An exegesis of transcendental syntax

Boris Eng

Laboratoire d'Informatique de Paris Nord (LIPN) – Université Sorbonne Paris Nord

An Exegesis of Transcendental Syntax

A journey into the logical machinery

Thèse de doctorat pour le diplôme de
Docteur en informatique de l'Université Sorbonne Paris Nord

Boris Eng

Thèse soutenue le 20 juin 2023 devant le jury composé de :

Thomas Seiller	CNRS – Université Sorbonne Paris Nord	Directeur de thèse
Damiano Mazza	CNRS – Université Sorbonne Paris Nord	Co-directeur de thèse
Lionel Vaux	Aix-Marseille Université	Rapporteur
Lorenzo Tortora de Falco	Università Roma Tre	Rapporteur
Claudia Faggian	CNRS – Université Sorbonne Paris Cité	Examinatrice
Jean-Baptiste Joinet	Université Lyon III	Président du jury

Octobre 2019 – Juin 2023

Abstract

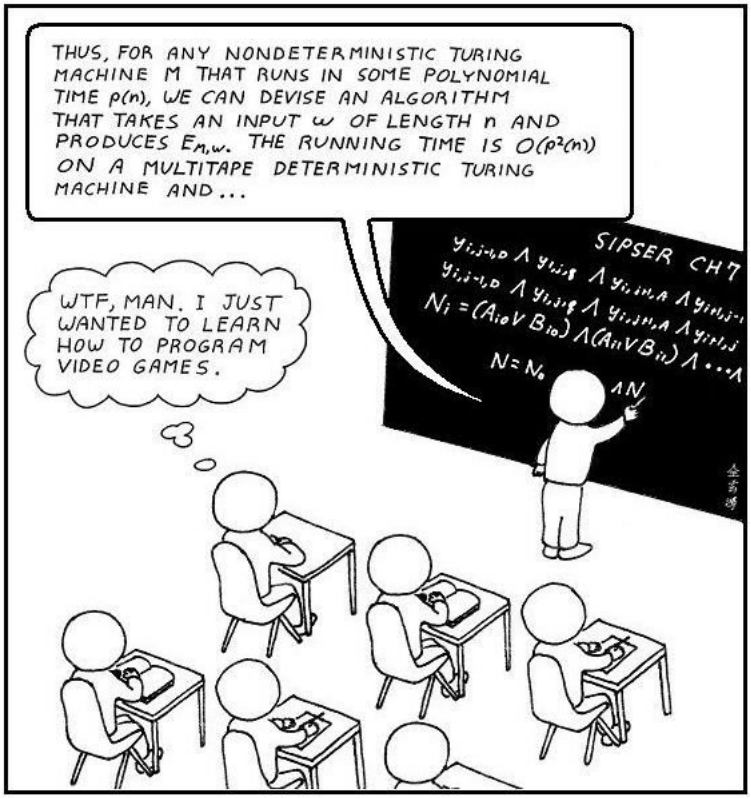
This thesis provides a clarification of Girard's recent work entitled "transcendental syntax". Girard suggests a reorganisation and a reinterpretation of concepts of mathematical logic coming from his previous works on linear logic, proof-nets, ludics and geometry of interaction. Unlike the approaches of semantic explanations based on the linguistic nature of logical entities and their evaluation, transcendental syntax suggests to start from the notion of computation as primitive in order to reconstruct mathematical logic, by starting with the proof-nets of linear logic. The Curry-Howard correspondence, now well-known among theoretical computer scientists, already established a formal correspondence between logic and computation. However, the links between logic and computation are still unclear. Are they two faces of the same thing? Are they different but intersecting notions? Or are they completely distinct? This thesis agrees with the last point of view: one is antique and almost mystical, the other is rather modern and pragmatic. In order to understand their differences and their role, transcendental syntax proposes a radical point of view: no primitive notion of mathematical proof, truth or formula is assumed. Only computation, through the model of computation called "stellar resolution" in this thesis, constitutes the elementary bricks from which logical concepts will be shaped. This model of computation, close to logic programming, uses asynchronous and very free mechanisms of term unification. The main problem that this thesis tackles is that Girard's proposal is rather informal and cryptic. A big objective is to clarify and to provide formal definitions. A particular effort is put in the accessibility of these ideas by suggesting a journey through the old and elementary roots of logic and computation to later reach the most modern works. Girard's works will then be put into context in both a historical and technical ways. The model of stellar resolution will be formally defined, illustrated and commented. The thesis concludes with a technical contributions realised with Thomas Seiller: a new interpretation of the multiplicative fragment of linear logic and a sketch of how it could be extended.

Résumé

Cette thèse apporte un éclaircissement sur le travail récent de Jean-Yves Girard intitulé "syntaxe transcendantale". Girard propose une réorganisation et une relecture des concepts de la logique mathématique venant de ses travaux précédents sur la logique linéaire, les réseaux de preuve, la ludique et la géométrie de l'interaction. À l'inverse des approches d'explications sémantiques reposant sur le caractère linguistique des entités logiques et leur évaluation, la syntaxe transcendantale propose de partir de la notion de calcul comme primitive afin de reconstruire la logique mathématique, en commençant par les réseaux de preuves de la logique linéaire. La correspondance de Curry-Howard, maintenant connue dans la recherche en informatique théorique, établissait déjà une correspondance formelle entre logique et calcul. Cependant, les liens entre logique et calcul sont encore mal compris. Sont-ils deux facettes de la même notion ? Deux notions différentes qui s'intersectent ? Ou alors deux choses complètement distinctes ? Cette thèse est en accord avec le dernier point de vue : l'une est antique et presque mystique, l'autre est plutôt moderne et pragmatique. Pour comprendre leurs différences et leur rôle, la syntaxe

transcendantale propose un point de vue radical : aucune notion primitive de preuve mathématique, de vérité ou de formule n'est supposée. Seul le calcul, à travers un modèle de calcul, appelé "résolution stellaire" dans cette thèse, constitue le matériau élémentaire à travers lequel s'incarnent les concepts logiques. Ce modèle de calcul, proche de la programmation logique, utilise des mécanismes asynchrones et très libres d'unification de termes. Le principal problème auquel s'attaque cette thèse est que la proposition de Girard est plutôt informelle et cryptique. Un grand objectif est donc de clarifier et de proposer des définitions formelles. Un effort particulier est mis dans l'accessibilité de ces idées en proposant un voyage à travers les racines les plus vieilles et élémentaires de la logique et du calcul pour arriver aux questionnements les plus modernes. Les travaux de Girard seront ainsi historiquement et techniquement contextualisés. Le modèle de résolution stellaire sera formellement défini, illustré et commenté. La thèse se conclut par une contribution technique réalisée avec Thomas Seiller : une nouvelle interprétation du fragment multiplicatif de la logique linéaire et une esquisse de comment elle pourrait être étendue.

Preface



Source: https://abstrusegoose.com/206.

What led to the existence of this document

When I began to learn computer programming for the first time, it was not especially out of intellectual curiosity or the (direct) consequence of some influence. What I wanted was very simple: to make video games. At the end of high school, I decided that I wanted to do computer programming because I enjoyed it (note that I was not aware of the existence of academic research in computer science and thought computer science was mostly about programming). I later became fascinated by how computation could be related to the "human mind". If I remember correctly, my first (superficial) interests were computational linguistics and computational neurosciences. How come that the

same things which make video games could be related to human thinking and language? My questions led me to Jérémy Ledent (student at École Normale Supérieure de Lyon at that time) and this was how I discovered the "Curry-Howard correspondence"; a formal correspondence in theoretical computer science relating computer programming and mathematical logic. This correspondence guided me for the following years as I was convinced that it was one of the current most important and overlooked scientific subjects. All my internships were chosen in order to pursue the understanding of this topic.

At one point, Jérémy mentioned Girard's book "The blind spot", which was apparently not a recommended course about logic. I tried to read it and decided that I wanted to work on linear logic (without even seriously knowing what it was really about).

At the end of university, I considered that computational complexity and Girard's "geometry of interaction" were the most profound subjects relating logic and computation. The only person I knew who was working on both (and that Paul-André Melliès suggested to me) was Thomas Seiller. Thomas proposed several subjects on computational complexity but I felt confident with none of them (I had a weak background in mathematics). He then suggested either to work on the "transcendental syntax" (Girard's most recent work) or linear dependent types. I chose transcendental syntax as I was more interested in fundamental questions. I actually did not know much about the transcendental syntax, barely read about it and could not understand a single paragraph that Girard wrote about it. For that reason, I never thought that it was worth considering and initially focussed on the geometry of interaction and computational complexity.

The design of this document

This long thesis has a chronological structure designed for the exploration of the roots of logic and computation and how they have led to Girard's transcendental syntax. We start from the premises of logic and computation that I associate with the intuition of space and time. I dare to start from the most natural intuitions of logic with Aristotle. Motivated by a search for a contextualisation of modern works in computer science, I then suggest to travel through time with the development of formal logic until Girard's linear logic. This logic, generally understood as a logic of resources or actions, is thought to be more primitive than the intuitive conception we had of logic in the 20th century. It seems to me that linear logic has not been adopted enough in the world-wide study of logic: it often appears that there are seminars of mainstream logic not mentioning linear logic at all. For that reason, I wrote most of what I know about linear logic in the simplest way I could. I mainly focus on intuition and pedagogy, forgetting technical details. I then tried to provide a simple presentation of Girard's geometry of interaction, a work of Girard which has only been partially accepted in the field of linear logic. At the end, this whole logico-computational journey should lead to a better understanding of the

transcendental syntax, which I consider the most conceptually mature (but technically immature) understanding of logic.

This thesis is qualified as "exegesis" mainly because that is how Lionel Vaux called my work with Thomas Seiller and I thought that it was a very accurate description since Girard's works are known to be very cryptic.

This thesis has been written with the following points in mind:

- *completeness*, justifying the length of the document. During my PhD thesis, I have met various people interested in Girard's recent ideas without being especially invested in the mathematics related to it. I found it necessary to provide resources so that the document would be as much accessible and self-sufficient as possible, even for philosophers or curious minds. I tried to provide a lot of external references as well for people willing to learn more;
- *pedagogy*, because I remarked that what was problematic in Girard's recent works was not really their technical difficulty but the lack of pedagogy and efficient communication. In particular, I tried to put efforts in the bridge between geometry of interaction (which is already known in the field of linear logic and some parts of theoretical computer science) and transcendental syntax;
- *accessibility*, as I initially wanted to write this document in French (my mother tongue, that I prefer over English) but after receiving few messages from non-French speakers interested in the transcendental syntax, I decided to write it in English instead;
- *entertainment* (probably odd for a PhD thesis), as I wanted to write something interesting and thought-provoking. For that reason, I included a lot of unknown or forgotten (mainly historical) facts. In most sections, a "discussion" part is included at the end in order to suggest non-technical comments on what has been written in the section.

How to read

Only the two first chapters are (almost) independent. The other chapters should be read in order until the end but not necessary from the beginning. Depending on your knowledge , you can choose to start from a specific point until the end.

All technical prerequisites are given at the end of this document, in the appendix.

Have a nice read!

Boris Eng.

Acknowledgements (in French)

Mon travail de thèse est le fruit d'un long parcours fait de rencontres et d'interactions. Cette partie de ma thèse est dédiée aux personnes qui m'ont apporté plus qu'elles ne le pensent et aux personnes qui ont indirectement contribué à ma thèse.

Je remercie mes directeurs de thèse Thomas Seiller et Damiano Mazza, que j'admirais déjà scientifiquement. Ils ont tous deux été successivement mes encadrants de stage de recherche en première et seconde année de master. Ils m'ont permis de satisfaire ma curiosité avec une grande liberté sur des sujets dont je ne savais en fait pas grand-chose.

Je remercie mes rapporteurs Lionel Vaux et Lorenzo Tortora de Falco pour leur exigence concernant la qualité technique de ma thèse et leur relecture particulièrement attentive et approfondie de mon manuscrit.

Je remercie les autres membres de mon jury: Jean-Baptiste Joinet pour son intérêt sincère envers mon sujet de thèse et Claudia Faggian pour ses conseils durant ma thèse. Claudia a notamment influencé mon choix de rediriger mon manuscrit en avance avec un parcours en largeur à moment où je ne savais pas où situer mon sujet. Un conseil que j'ai probablement poussé à l'extrême.

Je souhaite maintenant remercier, dans l'ordre chronologique, les personnes qui m'ont construit scientifiquement jusqu'à ma soutenance de thèse.

Ces premiers remerciements vont vers Monsieur Blot, mon professeur de mathématiques en terminale au lycée Dorian (filière technologique). Il a été le premier enseignant à croire en moi alors que j'avais toujours été un étudiant très moyen. Il avait remarqué et suivi mon intérêt pour la programmation mais aussi éveillé mon intérêt pour l'informatique théorique.

Je remercie Kostia Chardonnet, un de mes camarades de lycée qui m'a accompagné dans la programmation et mon humour le plus absurde. Je lui dois mon admission en DUT Informatique. Nous avons le point commun étonnant d'avoir suivi, ensemble, le même parcours jusqu'au doctorat en informatique théorique.

Je remercie Jérémy Ledent, rencontré par hasard sur un forum de programmation (mais jamais rencontré en personne !), qui a énormément guidé le lycéen naïf que j'étais et qui

m'a fait découvrir la recherche en informatique théorique et la passionnante correspondance de Curry-Howard. Il a influencé ma décision de poursuivre mes études en licence puis au Master de Recherche Parisien en Informatique (MPRI).

Je remercie Yannis Juglaret et Cătălin Hriţcu (qui était son directeur de thèse), qui m'ont très bien accueilli à l'Inria de Paris dans l'équipe Prosecco pour un stage avec l'assistant de preuve Coq en fin de DUT Informatique. Ils ont été tous deux particulièrement investis dans la confirmation de mon intérêt pour la recherche scientifique et ont été un moteur dans mon parcours.

Je remercie Michele Pagani et Delia Kesner qui m'ont accueilli en stage de L3 informatique sur la logique linéaire, sujet sur lequel je voulais travailler sans vraiment savoir de quoi il s'agissait. Ce stage m'a confronté à un mélange entre formalités techniques et théories abstraites qui est nécessaire dans la recherche en informatique théorique.

Je remercie Olivier Laurent (que je n'ai en fait jamais rencontré et qui ne me connaît probablement que de nom) qui a répondu, par e-mail, à mes questions techniques de la L3 Informatique jusqu'au doctorat (probablement sans le remarquer). J'ai toujours été impressionné et inspiré par son sens du détail et de la précision.

Je remercie Tito que j'ai considéré comme mon "grand-frère scientifique". Il m'a guidé en master et en thèse sur la logique linéaire, la géométrie de l'interaction et bien d'autres sujets de l'informatique théorique voire du monde académique en général.

Je remercie toutes les personnes qui ont accepté d'échanger avec moi concernant ma thèse : Matteo Acclavio, Benjamin Hellouin, Mccolm Gregory et Natasha Jonoska, Alexander Leitsch, Carlos Olarte, Félix Castro, Paolo Pistone, Arnaud Valence, Aleksey Gonus, François-René Rideau alias Fare (qui influencera la suite de mon parcours), Rémi Nollet ainsi que d'autres anonymes et personnes extérieures au monde académique.

Mes derniers remerciements vont au groupe de travail ReFL (Réflexions sur les Fondements de la Logique, et initialement "nouvelle église girardienne" puis "cercle transcendantaliste") que j'ai co-fondé avec Davide Barbarossa, Valentin Maestracci et Pablo Donato. Parmi les contributeurs principaux de ReFL figurent : Adrien Ragot, Ambroise Lafont, Baptiste Chanus, Jérémy Hervé, Kostia Chardonnet, Julien Marquet, Paul Séjourné, Sidney Congard, Tito et Xavier Denis. Ils m'ont tous permis de sortir de ma solitude de scientifique mais m'ont aussi apporté une grande stimulation intellectuelle : de nombreux passages de mon manuscrit viennent en fait de conversations avec des membres de ReFL. J'ai trouvé dans ReFL des personnes à la pensée profonde, sceptique et audacieuse. Parmi ces membres, Pablo Donato et Paul Séjourné ont particulièrement suivi mes réflexions sur la syntaxe transcendantale et j'ai beaucoup échangé avec eux sur les idées les plus expérimentales et spéculatives.

Contents

> *Tout commence en mystique et tout finit en politique.*
>
> *– Charles Péguy (Notre Jeunesse)*

Chapter 1

Logical traditions

In this chapter, instead of exposing the usual folklore of logic, I suggest a personal reading of the Western tradition of mathematical logic. This revision of logic is deeply influenced by modern works on Girard's linear logic and its connexions with computation. Despite these rather technical inspirations, I would like this chapter to be very accessible but I also wrote it so that it can be interesting for people who already received an education on formal logic.

I start with a historical background coming from a personal little investigation. My reference for the formal presentations of logic is Hedman's introduction to logic [Hed04]. I decided to modify a bit the usual definitions and notations according to my personal taste.

1 The experience of regularity

This section is inspired by Paolo Pistone's great presentation (in French) *"Aristote et l'électricien : le branchement des idées"* for *Treize Minutes Marseille* and Alain Lecomte's course on philosophy of language.

§1.1 We all have a vague idea of what logic is, as if it was either hard-wired in our brain or part of nature itself. Imagine that I say:

"All humans are mortal" and *"all Greeks are human"*, hence *"all Greeks are mortal"*.

"All humans are mortal" and "*all French are human"*, hence *"all French are mortal"*.

"All unicorns are cool" and *"all turtles are unicorns"*, hence *all "turtles are cool"*.

§1.2 These three sentences produce new knowledge from two premises (preceding the word "hence"). In the first two statements, we can *feel* that the words *"Greek"* and *"French"* are interchangeable and unimportant. No matter what word we replace these words with, the statement seems to hold. If we look at the more extreme third sentence,

Code	Name	Constructor
A	Universality	All S are P
E	Existence	Some S are P
I	Negative universality	No S are P
O	Negative existence	Some S are not P

Figure 1.1: Types of assertion in Aristotelian syllogisms.

the subjects and properties (called *predicates*) are replaced by pure nonsense and the whole sentence still makes sense. In each case, the two premises are only *hypothetically* considered valid but are not necessarily valid themselves. What matters is whether or not the conclusion follows from the two premises, hence validity is actually independent from our understanding of words or of any external reference. This is what we call *deductive reasoning*. These statements go without saying and we would simply say "it's logical!" to justify them.

§1.3 Notice that the three sentences above have the same *shape*, hence exhibiting some regularities in the production of knowledge. They are instance of a *pattern*. The concepts can be replaced by generic symbols, hence only the *spatial* arrangement of concepts matters and not the concepts themselves. We obtain the following more general statement:

"All M are P" and *"all S are M"*, therefore *"all S are P"*.

§1.4 **Aristotle's syllogism.** The letter "M" stands for *major premise*, "P" for *predicate* (a property of a subject) and "S" for *subject*. This type of deductive reasoning having two premises and a unique conclusion is what Aristotle called a *syllogism* in his work *Prior Analytics* (part of the *Organon*), during Ancient Greece. By analysing all the possible syllogisms constructible with the assertions of Figure 1.1, it is possible to create categories of logical patterns. At the time of Aristotle 19 valid types[1] of syllogisms were extracted from 256 logical combinations. In particular, our syllogism above has been given the mnemonic name[2] of B**A**RB**A**R**A** for the three occurrences of universal assertions (code *A* in Figure 1.1).

§1.5 Deduction does not come from nowhere. How can such rules exist and how can they produce valid knowledge? A modern hint would be that assertions such as "All M are P" are *usable entities* with an input and an output allowing connexion with other assertions. It is a machine producing P from M. In order to *prove* "all S are P", we connect the two premises seen as tools: I have "S" and know that it leads to "M" (second premise), itself leading to "P" (first premise). It is illustrated by the path of Figure 1.2 between the two occurrences of "P". This gives a *spatial* criterion for logical correctness.

[1] Leibniz added up to 25 types in his *De arte combinatoria* (1666).

[2] Mnemonic names and the notations A, E, I and O were given by medieval scholars [KKPS82] (associated with what we call *scholasticism*). The idea was to associate syllogisms with Latin names (Barbara, Cesare, Darii, Barbari, ...) so that vowels correspond to types of assertion.

"All M are P" and *"all S are M"*, therefore *"all S are P"*.

Figure 1.2: Path in the reasoning of an instance of syllogism.

Name	**Constructor**	**Modern notation**
Conditional	**If** A then B	$A \Rightarrow B$, $A \supset B$, $A \to B$
Disjunction	Either A **or** B (but not both)	$A \vee B$, $A + B$
Conjunction	A **and** B	$A \wedge B$, $A \cdot B$
Negation	**not** A	$\neg A$, $\overline{A}$

Figure 1.3: Logical connectives in stoic logic where A and B are assertibles.

§1.6 **Stoic logic.** Still in Ancient Greece, the *stoics* [GW04, Volume I] (and Chrysippus in particular) developed a different (and more modular) point of view on logic where a whole sentence such as "it's raining today" is a single object called *assertible*. Assertibles are either *true* or *false* and can be connected with *logical connectives* presented in Figure 1.3. In this case, the focus is not on the *structure of reasoning* but on the treatment and preservation of truth during the evaluation of a sentence. For instance, given two assertibles A and B, if A is true but B is false then "A and B" must be false. It suggests that logical validity is ruled by some mechanisms underlying logical connectives. Reasoning then relies on five indemonstrable arguments given in Figure 1.4.

§1.7 Although the study of logic can take several form, in the two cases described above (Aristotelian logic and stoic logic), logic takes place in the very specific and concrete space of *language*. Logic was thought to be about what could be expressed with written or spoken words. During Ancient Greece, syllogism was correlated with discussion, public speaking and the art of persuasion. If there are valid ways of reasoning, there are also wrong ways known as *(logical) fallacies*, which can be used on purpose to trick or convince people (about skills or politics for instance). Such wrong arguments look like valid ones but exploit the ambiguity of consequence:

"A better product is usually more expensive" and "this product is expensive",

Name	**Constructor**
Modus ponens	(A implies B) and A, therefore B
Modus tollens	(A implies B) and not B, therefore not A
Conjunctive syllogism	Not (both A and B) and A, therefore not B
Modus tollendo ponens	(Either A or B) and not A, therefore B
Modus ponendo tollens	(Either A or B) and B, therefore not A

Figure 1.4: Indemonstrable arguments in stoic logic.

therefore "it must be of a better quality".

§1.8 Using this argument, I can make you buy something at an unfair price. The art of "unreal wisdom" which was criticised during Ancient Greece is usually called *sophism*[3]. In *"On Sophistical Refutations"* (part of the *Organon*), Aristotle described several categories of fallacies and how to react to it. The list has later been developed several times by several authors [BC06, Sch31] and is now considered as a sort of *intellectual self-defence*[4].

§1.9 **Medieval logic.** Something I have rarely seen mentioned in logic is the medieval world[5] [GW04, Volume II]. Medieval schools were mainly interested in the study of Aristotle's syllogisms as if they were sort of multiplication tables we learn by heart (each valid syllogism is associated with a mnemonic Latin name). Medieval works on logic includes additional linguistic features such as plurality, tense and modality in the context of Aristotelian logic or theories about terms and predicates [Rea02, Spa02, Knu99]. Succeeding Greek's *Dialectic* (the discourse between several opponents wishing to establish truth through valid arguments, thus relating logic and dialogue) was considered part of logic in medieval schools [Abe06]. But beside this Aristotelian legacy, medieval logic was not void of innovation: interestingly, the first proof of the *principle of explosion*[6] is attributed to a French logician of the 12th century named William of Soissons [Mar86]. This principle states that from a contradiction such that "A and not A", it is possible to infer any statement, which should sound very surprising.

§1.10 It is generally agreed that Western logic mainly originates from Aristotle, stoics and medieval schools. However, something we do not always realise is that, despite the objective appearance of logic, our understanding of it is strongly influenced by seemingly insignificant things such as culture and geography. Indian logic [GW04, Volume I] (itself correlated with Buddhist logic [Fac83]) and Chinese logic [Chm09] (with the *School of Names*) were especially old forms of logic which were developed independently of Greek logic. In particular, Indian logic later had a direct influence on Western logic between the 18th and 19th century [Col58]. Something else which is notable is that Arabic logic [Bla98, Res64] had early forms of syllogisms which actually influenced Islamic law and theology, but also had an early access to Greek logic through Arabic translations, even before the Christian scholars of medieval schools. This led to Avicenna's alternative to Aristotelian logic including novelties such as *temporal modalities* [Res12]. Despite all those traditions of logic, this thesis will only be another brick in the story of Western logic, but I believe that it would be very nice to study old alternative traditions of logic in the light of the modern works that I will present in this thesis.

[3] Although it originally referred to the teaching of eloquence and the art of persuasion.
[4] Especially on social networks. Unfortunately, debates do not only evolve in the logical dimension.
[5] Which suffers from a lot of unfortunate prejudices.
[6] Although it seems to have been rejected at first [Mar86].

Syllogism	**Boolean equation**
All A is B	$AB = B$ or $A(1-B) = 0$
No A is B	$AB = 0$
Some A is B	$V = AB$
Some A is not B	$V = A(1-B)$

Figure 2.1: Algebraisation of Aristotle's syllogisms. Saying that all A is B is the same as saying that the concept of being both A and B is the same as the concept B (inclusion of A within B), written $AB = B$, or equivalently that being A but not B is a nonsense ($A(1-B) = 0$). Saying that no A is B is the same as saying that being both A and B is a nonsense ($AB = 0$). Saying that some A is B is the same as saying that there is some concept V equal to the concept of being both A and B. This works if concepts are non-empty (which is also the case in Aristotle's point of view). From these interpretations, it is then easy to understand the last syllogism.

2 Logical dreams and mathematical realisations of reasoning

§2.1 Between the 17th and 18th century, Gottfried Leibniz was a prominent scientific figure which contributed to several fields such as mathematics, philosophy, theology, ethics and more. However, it seems that his works on logic did not attract much attention at his time. For that reason, it is often considered that he foresaw the future developments of formal logic. Leibniz had the very ambitious project of designing a universal symbolic language called *characteristica universalis* [Jae96], which would be relevant for any rational discussions whether it be in metaphysics, physics, mathematics, music, or even everyday life conversations. Such a language would be based on an *alphabet of human thoughts* [WGRO93] which constitute rational thinking. Although this idea has been mentioned several times by Leibniz, it seems that it has never fully flourished and has even be considered too naive by its creator[7]. This great ambition resuscitated again after Leibniz, during the 19th century with a different philosophy.

§2.2 **Algebraic logic.** Still in this idea of a calculus of logic, the English mathematician George Boole initiated an algebraisation of logic[8] in *"The Laws of thought"* which aimed at giving a mathematical foundation to Aristotle's logic which would be simple and as close as possible to "high school algebra"[9] [Bur00]. In his algebra of logic, variables $x, y, z, ...$ represent classes or concepts over which we use arithmetic operations $+, -, \times$ and a symbol $=$ for equality. The intersection of concepts is written $x \times y$ or xy. This allows to express the fact that "good man" is the intersection of "good" and "man". In particular, logic has to satisfy $xx = x$ (which can also be written $x^2 = x$), illustrating

[7] According to an informal discussion with David Rabouin, Leibniz himself later considered this universal alphabet to be probably impossible to realise.
[8] Which is actually different from what we call *Boolean algebra* today.
[9] Which I never studied in high school.

the fact that being "good and good" is the same as being "good". Remark that the only numbers satisfying this equation are 0 ($0 \times 0 = 0$) and 1 ($1 \times 1 = 1$), which is coherent with the truth values *false* (concept of *emptiness*) and *true* (concept of *the whole universe*). Boole was able to represent Aristotle's assertions in this system: "all x are y" becomes $xy = x$ (all things such are both x and y are also x, hence the concept of x is contained in y) and "no x is y" becomes $xy = 0$ (it is impossible to be both x and y). Other syllogisms are presented in Figure 2.1. Interestingly, Aristotle's A (universal) statements looks like sort of *functions* taking A and producing B by associating a new "category" to a given individual coming from a given category. As for E statements, they are like *pairs* (A, B) reuniting an individual and the category it is part of. This intuition will actually be essential in modern developments of logic such as *(dependent) type theory*. Finally notice that subjects and predicates are of the same kind (unlike Aristotelian's syllogisms which distinguishes the two)[10].

§2.3 In this "algebraic period" [Boc61], several mathematicians such as Ernst Schröder, Augustus De Morgan, Hugh MacColl, Charles Sanders Peirce and John Venn contributed to the establishment of relations between an algebraic presentation of logic and other fields such as set theory, order theory or lattice theory. Works on algebraic logic were updated, extended and developed actively in that direction. At this time, Indian philosophy started to have an influence on Western scholars (especially English): Aristotelian logic and Indian logic were compared with the conclusion that they were not equivalent [Gan13]. De Morgan himself was also aware of Indian logic and even mentioned it explicitly:

> "*The two races which have founded the mathematics, those of the Sanskrit and Greek languages, have been the two which have independently formed systems of logic.*"
>
> – *Augustus De Morgan [DM60]*

§2.4 **Ideography.** Following these first mathematical presentations of logic and Leibniz's initial ideal of a universal language[11], Gottlob Frege invented a graphical language for logical sentences called *Begriffsschrift*[12] [F+79] (1879) or *ideography* in English. The construction of sentences is presented in Figure 2.2. Frege wanted it to be what Leibniz hoped for: a sort of microscope allowing the analysis of the relations between concepts. I would personally call it the "circuits of thought". This graphical language for logic constitutes the first known occurrence of *universal quantification* (expressing "for all" in logical sentences) with bounded variables. Something which is a bit less known is that

[10] Actually, this shared space for subjects and predicates is unusual. Even in more modern formal logic (including dependent type theory), they are usually distinguished.

[11] Although Frege places himself in Leibniz's scientific lineage, it seems that Frege's philosophy does not reflect Leibniz's philosophy. This relation to Leibniz apparently produced an exaggerated image of Leibniz as a logicist. See David Rabouin's works for more details.

[12] I was very fascinated by it when I started to learn logic.

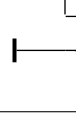
Name	**Frege's Begriffsschrift**	**Modern notation**
Assertion		A
Negation		$\neg A$
Tautology		$\models A$
Contradiction		$\not\models A$
Equality ($\neq$ logical equivalence)		$A = B$
Universal quantification		$\forall x.P(x)$
Conditional		$A \Rightarrow B$
		$\models \forall x.P(x) \Rightarrow P(x)$

Figure 2.2: Sentences written in Frege's Begriffsschrift. The representation of the formula $\forall x.P(x) \Rightarrow P(x)$ stated as a tautology is presented at the end as an example. A tautology is a statement which is always true whatever the content of its subformulas. A contradiction is always false.

Peirce[13] actually invented independently a graphical language called *existential graphs* [Pei79], which includes both *universal* and *existential* ("there exists") quantifications. Both these systems are anticipations of what we call *predicate calculus* or *first-order logic* [Fer01] in today's mathematical logic and which is now standard.

§2.5 **Frege's calculus.** Frege does not content himself only with a contemplative graphical language for logical sentences. In his article on Begriffsschrift [F$^+$79, §6], he already presented how to write inferences by stacking formulas. However, in order to do inferences, we need rules of inference to know what can follow from what, and we also need hypotheses left unproven (at the top on the inference). These statements are called *axioms*. Frege justifies them informally by their *self-evidence* and how they appear in the practice of mathematics. For instance, this is how Frege justifies the axiom $A \Rightarrow (B \Rightarrow A)$:

> "*This is evident, since A cannot at the same time be denied and affirmed. We can also express the judgment in words thus, "If a proposition A holds, then it also holds in case an arbitrary proposition B holds". Let A, for example, stand for the proposition that the sum of the angles of the triangle A BC is two right angles, and B for the proposition that the angle ABC is a right angle. Then we obtain the judgment "If the sum of the angles of the triangle A BC is two right angles, this also holds in case the angle A BC is a right angle.*"
>
> – *Gottlob Frege [F$^+$79, §14]*

Another notable axiom is the statement $c = c$ stating that all statements are equal

[13] You can ask Pablo Donato about it, he likes to talk about Peirce.

to themselves (because they refer to the same concept). All statements which are not axioms are inferred by using the following inference rules (in modern notations):

- ◇ **Modus ponens.** From A and $A \Rightarrow B$, we can infer B;
- ◇ **Generalisation.** From $A \Rightarrow P(x)$ where x does not occur in A, we can infer $A \Rightarrow \forall x.P(x)$.

§2.6 Formal consideration of linguistic features led to several new notations for logic. It is not so clear how these notations were introduced, how they have been propagated and how they evolved. The most common notations have already been presented in Figure 1.3. Hence, a sentence such as *"Either A is true or it is not"* would be written $A \vee \neg A$. The quantified formulas of Frege are written $\forall x.A$ for "for all x, A holds" and $\exists x.A$ for "there exists some x such that A holds", although these notations were introduced by different people at different times. Properties are expressed with *predicate symbols* which are used like function symbols. For instance, we write $P(t)$ to say that the property P holds for some term t and it produces a statement which is either true or false. Frege also had a notion now called *second-order quantification* [F+79, §27] which corresponds to a quantification over predicates or formulas (which are predicates with no argument). For instance, $\forall A.A \vee \neg A$ would be a second-order formula.

§2.7 **Logicism.** Frege's philosophical project is called *logicism*. Logicism's hope was to reduce some (or if possible, all) parts of mathematics (especially arithmetic) to logic, which is understood as "pure relations between concepts". This cold-hearted point of view sees logic as connexions between evidences. It attracted a great attention from mathematicians such as Giuseppe Peano, Richard Dedekind and Alfred North Whitehead which became active contributors but it is usually considered that the project has been mainly carried by Bertrand Russell. Logicism is also related to *analytic philosophy* which studies philosophy with an emphasis on linguistics and the use of mathematical tools.

§2.8 Logic turned into syntax. At this point, logic is defined as if humans were somehow able to experience logic and materialise intuitions of it into syntactic objects. Unfortunately, this approach is far from being trouble-free. What logicians have done was trying to frame the practice of mathematics into syntax. We will see that mathematical practice is not so obedient and can escape from the jails we design.

3 Paradoxes and the jails of the Format

§3.1 An antinomy is a mutual incompatibility between two rules. It is probably the most pathological phenomenon in logic. Several antinomies began to appear when trying to capture some parts of mathematics with formal logic. An early appearance of antinomy, even before these considerations is the Burali-Forti[14] paradox (1897) which came to the

[14] It's actually one person and not two.

conclusion that a naive formalisation of set theory leads to antinomies [BF97]. This paradox[15] concerns a generalisation of numbers called *ordinals*. Assuming the existence of a set of all ordinals leads to the existence of an ordinal greater than itself, which is absurd. It is thought that Cantor, considered as the inventor of set theory, was informally aware of this fact.

§3.2 The most famous example of antinomy is probably Russell's paradox [Rus03, Chapter X] which can be seen as a simpler version of the Burali-Forti paradox (it seems that it has even been suggested by Burali-Forti himself). We write $x \in y$ to formalise the fact that the entity x is contained in the entity y and $x \notin y$ when it is not the case. Sets can be written *extensively* as a collection of elements, such as $\{1, 3, 7\}$. We can for instance write $1 \in \{1, 3, 7\}$, $2 \notin \{1, 3, 7\}$ or even $\{1\} \in \{\{2\}, \{1\}\}$ (if we would like to base mathematics on set theory, then numbers have to be encoded as sets). It is then technically possible to write $x \notin x$ (although it seems absurd, it holds for the set $\{1\}$ as we do not have $\{1\} \in \{1\}$). Now, it is possible to collect all such elements not containing themselves, written $y = \{x \mid x \notin x\}$, to be read "the set of x such that $x \notin x$". The contradiction appears when we are trying to ask the forbidden question "Does y contain itself?" written "$y \in y$". If $y \in y$ holds, then y is part of the sets which do not contain themselves by definition of y. It is contradictory! Then $y \notin y$ must hold. However, if it is the case, it should have been be part of the collection of sets not containing themselves. Which is also contradictory with our hypothesis... This formulation of set theory is known as *naive set theory*.

§3.3 An immediate and natural solution to that antinomy, provided very early by Russell himself in his *The principles of mathematics* (1903), which has been then developed with Whitehead in *Principia Mathematica* (1910), is to say that our definitions were too permissive and that we need to put constraints on them. It makes sense because such contradictions do not reflect the practice of mathematics. These different ways of constraining the use of syntactic entities (typically, forcing categories to prevent some syntactic manipulations or inferences) led to different branches of logic with their own culture and practices.

§3.4 **Type theory.** This solution suggested by Russell corresponds to what is now called *type theory*. We classify entities so that unexpected uses are prevented. In this approach, the root of evil is the notion of *impredicative definition*. The Russell's paradox contains a vicious circle related to self-reference. The variable x in $\{x \mid x \notin x\}$ is a generic variable referring to the set of all sets, including itself. In order to avoid this illicit contact, Russell classified the entities of set theory with a hierarchy of types so that some entities cannot interact which each other. Russell's idea was to design a hierarchy of levels so that, when writing $x \in y$, x must be at level n and y at level $n+1$. It is then impossible to write $x \in x$ because x cannot be both at level n and $n+1$. Another way to understand this approach is that Russell's paradox can also be understood as the fact that there is

[15] As Girard exposed very often in his writings, we could say antinomy instead since it is a contradiction with the definitions and not something simply "out of dogma" as the name suggests.

no set of all sets. This hierarchy of levels actually has a consequence on mathematical definitions since several definitions are impredicative but used without problems (such as the definition of greatest lower bound) but also because the use of predicate becomes limited by our restrictions (we do not have access of all predicates anymore). Several attempts at having a more nuanced approach to those constraints came later, such as Russell's *ramified hierarchy* motivated by a fear of these vicious cycles rather than the initial need to avoid a leakage caused by antinomies.

§3.5 **Axiomatic.** Another solution for a logical foundations of set theory is to rely on axioms (*cf.* Paragraph 2.5) to clearly express what is allowed or not, which is indeed a quite *military* solution[16]. Axiomatic systems existed for a long time (*e.g.* Euclid's axioms for geometry) but they were given a greater importance at the time of the crisis of mathematical foundations. Also as early as Russell (1908), Ernst Zermelo suggested an axiomatic theory for set theory known as Z which has later been extended to ZF, then to the so-called ZFC axiomatic system. By choosing axioms in a clever way and on which mathematicians agree, it is possible to produce a trustworthy foundation for set theory and some (if not all) parts of mathematics. For instance, the first axiom is the *axiom of extensionality*, written $\forall x.\forall y.(\forall z.(z \in x \Leftrightarrow z \in y)) \Rightarrow (x = y)$ with modern notations of formal logic. It means that if two sets contain the same elements, then they are equal. The use of axioms forbids unwanted statements such as Russell's paradox. Adding more axioms allows to express more complex mathematical theory but coherence of the new set of axioms must be ensured. Actually, although axiomatic is interesting for foundational purposes, it barely affects mathematical practice.

4 The location of certainty: the mind or the paper?

§4.1 The appearance of paradoxes challenged the logicist project of capturing parts of (or all) mathematics with formal logic. This period of the early 20th century, known as the *foundational crisis of mathematics*, gave rise to *metamathematics* which is the study of mathematics by using mathematical methods. Two important schools of thought (both opposing the previous logicist project) appeared during this crisis: the *formalists* (often represented by Hilbert) and the *intuitionists* (often represented by Brouwer). Those two schools of thought, together with logicism, represented by Russell, were the most important schools of logic at that time.

§4.2 One thing on which those two schools of thought did not agree was the nature of mathematics and the *location* of certainty. Logicism is often associated with *semantic realism* which considers an external reality of mathematical meaning, independent from us and to which we can refer. For instance $x = y$ when x and y "refer to the same meaning". Intuitionism and formalism are two rivals opposing logicism and returning to Kant's seeking for an immanent logical meaning. A way to understand the difference between

[16] As often pointed out by Girard.

formalism and intuitionism is to ask whether the nature of mathematics is to be found in our mind or our paper.

§4.3 **Intuitionism.** For intuitionists, mathematics is a purely mental activity; a production of the human mind [BP84, Chapter 2]. Language is used as a way to communicate mental constructions. Certainty lies in what is accessible by the human mind and which can be constructed and communicated. Although innocent, this conception of logic has important consequences such as the doubt of some logical statements which were considered valid. Typically, the excluded middle "A or not A" which states that any statement is either true of false. If we are able to prove that A is false (by assuming it and showing that it leads to a contradiction), A must be true. In the particular case of existential statements "either there exists x such that P or there is no x such that P" for a property P, it happens that using the excluded middle allows us to infer the existence of some mathematical entities although they are mentally out of reach: we cannot construct them and yet we assert their existence. What we need is an explicit construct that we call an *existential witness* which is an intelligible proof of existence. This leads to serious doubts about all statements logically equivalent to the excluded middle. This divides logic into two categories: *intuitionistic logic* excluding these statements and *classical logic* which allows them.

§4.4 **Formalism.** Hilbert thought of logic as a *formula game*. His approach is more procedural and perhaps more *conventionalist*. All philosophical questions aside, mathematical practice mainly involves syntactic operations related to "fixed rules known to all mathematicians" [BP84, Chapter 3]. Certainty lies in the *purity of syntax* (the paper, in some sense). The idea of Hilbert's programme was to prove and verify statements by finite and mechanical means[17]. In formalism, there is no strong reason to reject the problematic classical statements which were rejected by intuitionism. These statements are part of mathematical practice. What is important are not the statements but the methods of proofs and then mathematics becomes a "combinatorial game" [BP84, Chapter 3]. Contradiction can be represented with a symbol $\bot$. Formalists are then especially interested in the property of *coherence* stating that we cannot produce the symbol $\bot$ with a given system. In some sense, formalists' logic is only about syntax and conventions.

§4.5 **Proof systems.** Frege's calculus of Paragraph 2.5 constitutes an example of proof system which is able to *prove* statements from previously assumed statements. Proof systems are the key for the Hilbertian ideal of a finite and mechanical verification of statements. Logic would then be *procedural* and *bureaucratic*. In its modern form, using modern notations for formulas instead of Frege's Begriffsschrift, Frege's calculus is called *Hilbert system*. A *proof* becomes a mathematical object which can be studied by itself. It is simply defined as a sequence of formulas following some given rules and which starts with a given set of axioms. It is also possible to begin with more axioms depending on the mathematical theory we would like to consider (which also allows for hypothetical

[17] In particular, he also had the ambition of reducing infinitary mathematics to finitary ones (one can argue that in some cases infinitary reasoning is not necessary).

Line	Statement	Justification
1	$((A \Rightarrow ((A \Rightarrow A) \Rightarrow A)) \Rightarrow ((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A)))$	by A_1
2	$A \Rightarrow ((A \Rightarrow A) \Rightarrow A)$	by A_2
3	$(A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A)$	by MD(1,2)
4	$A \Rightarrow (A \Rightarrow A)$	by A_1
5	$A \Rightarrow A$	by MD(3, 4)

Figure 4.1: Proof of $A \Rightarrow A$ from the two axioms $A_1 := A \Rightarrow (B \Rightarrow A)$ and $A_2 := (A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$ together with the modus ponens (MD) as only rule (from A and $A \Rightarrow B$, we can infer B). These rules are taken from a proof system called H_1 which is presented in a course of Stony Brook University (https://www3.cs.stonybrook.edu/~cse541/chapter8.pdf).

statements assumed to be true). An example of a proof of $(A \Rightarrow A)$ from a minimal set of axioms and rules is given in Figure 4.1. Representing mathematical proofs as mathematical objects allows for an analysis of reasoning but also allows results about proofs themselves (*cf.* Paragraph 4.1 for metamathematics). Several proof systems have been developed afterwards such as Gentzen's natural deduction and sequent calculus [Gen35a, Gen35b]. As we will see later in this thesis, proof systems became essential for their connexion with (typed) programming in computer science.

5 Syntax and semantics / Language and reality

§5.1 A dominant point of view in today's logic is to explain the logical activity by a separation between syntax (what we write) and semantics (what we mean). In a lot of beginner books or in university courses, logical systems are presented from their syntax then from their semantics. It became so systematic that it is now quite *cliché*. When I studied logic, I identified mathematical logic and *logic* in general but we should keep in mind that mathematical logic is *one possible* implementation of logic which is dependent of some traditions and history. Hence, mathematical logic is a *part* of logic, far from a definitive answer to the (philosophical) nature of logic. This dominant conception of mathematical logic assumes a direct access to reality (semantics) by language (syntax) but also an identification of reality with some mathematical semantic theories (typically, truth interpretation).

§5.2 It is possible to do logic both in the syntax and the semantic world. In the syntax, we typically have proof systems and axiomatic systems. We have formulas and infer other formulas according to some given rules. We write $\Gamma \vdash_S A$ when A can be inferred from a set of formulas $\Gamma = \{B_1, ..., B_n\}$ in some system S. We can write $\vdash_S A$ when A is provable without any assumptions (for instance simply from the axioms of S). In semantics, we are interested in the meaning of symbols by evaluating expressions. For instance, the expression $A \vee \neg A$ where A is true can be evaluated to the truth value 1

(representing truth). The evaluation gives value of the whole expression by computing the truth value of its components. We write $\Gamma \models A$ when the truth of formulas in Γ implies the truth of A. We simply write $\models A$ when A is a tautology (vacuously true for any truth value of its components). Mathematical theorems correspond to tautologies. Several semantic theories with more complex mathematical interpretations also exist such as having three truth values, theory of all possible worlds (Kripke semantics) etc.

§5.3 **Soundness and Completeness.** Now that we can do logic in two different worlds, we need results about the correspondence between them: that the logic of syntax exactly corresponds to what we mathematically mean.

- The first property we usually require is *soundness* which states that $\Gamma \vdash_S A$ implies $\Gamma \models A$ or in other words that our procedural system of symbols is correct (sound) *w.r.t.* our conception of meaning.
- The second property is *completeness* which says that $\Gamma \models A$ implies $\Gamma \vdash_S A$, or in other words that all what we mean is faithfully captured by syntactic operations.

These properties appear in a lot (if not most) papers about logic. As for the *consistency* property of Paragraph 4.4, it is indeed a syntactic property: the fact that we cannot infer a special symbol $\bot$ representing contradiction, or more formally $\nvdash_S \bot$.

§5.4 **First incompleteness.** Despite the ambitions of formalism and logicism, it is considered that Gödel's *incompleteness theorems* [Göd31] made these two projects almost obsolete. There are often confusions about what Gödel's theorems really say. The informal idea is that Gödel's result is a sophisticated version of the *liar paradox* (explicitly mentioned by Gödel). In natural language, we can assert that "the statement X is false". In the space of all statements which can replace X, there is... the previous statement itself! We obtain "*this* statement is false". If it is true then it is contradictory because it says it is false. But if it is false then it is consistent with what the statement says, then it must be true. There is a mismatch between the external commentary on sentences and the meaning associated with syntactic expressions.

§5.5 I refer to Smith's book for more technical details [Smi13]. To give more context, there was hopes at giving logical foundations to arithmetic for natural numbers. Peano's arithmetic was an attempt at providing a finite set of axioms from which all valid arithmetic statements would be inferred. Let T be a set of axioms capturing arithmetic. By only using natural numbers, it is possible to encode various statements. For instance, it is known that there is a bijection between $\mathbf{N}$ and $\mathbf{N} \times \mathbf{N}$, hence natural numbers can encode pair of natural numbers. With this idea of encoding, Gödel managed to construct a statement G_T of T stating "G_T is unprovable in T". Gödel's first incompleteness theorem is the following:

1. neither G_T nor $\neg G_T$ are provable in T. Hence, there exist statements for which provability is impossible to decide with a finite and mechanical procedure. This ends Hilbert's formalist ambition;

2. if T is consistent then G_T is true (the contradiction $\bot$ cannot follow from the rules of T).

If we assume T to be consistent, then G_T is true (2) but not provable (1). Hence there exists at least one statement in T which is true and not provable. This exhibit a gap between truth and provability. In some sense, T is *incomplete* (hence the term "incompleteness"). But more than incomplete, it is *incompletable* (to quote Smith) because providing T is expressive enough to express arithmetic, it is always possible to construct such unprovable formulas even if we extend it with more axioms. Although it sounds serious, it mostly affects the logical foundations of arithmetic and not mathematical practice (since we usually do not end up with those weird problematic statements).

§5.6 **Second incompleteness.** We write C_T for "T is consistent". The second condition of Gödel's first incompleteness theorem says that C_T implies G_T. The reasoning behind the proof of the implication $C_T \Rightarrow G_T$ can actually be encoded in arithmetic. It appears that $C_T \Rightarrow G_T$ is provable in T (by using arguments we do not detail but which are given by Smith [Smi13]). When T is assumed to be consistent, then by the second condition of Gödel's first incompleteness, G_T cannot be proven. It follows by $C_T \Rightarrow G_T$, that C_T cannot be proven as well, otherwise we would be able to prove G_T (notice that this does not affect the provability of the implication). Gödel's second incompleteness theorem is a corollary coming from this reasoning, which says that if T is consistent then it cannot prove its own consistency (the statement C_T). This is again a barrier against the total, mechanical and finitistic justification of mathematics hoped by formalists.

§5.7 **Gödel's hierarchy.** It would be exaggerated to say that this is the end of formalism. If Hilbert's ideal cannot be fully realised, some parts of it can still be saved. Even though the consistency of a formal system cannot be shown inside it (if it is expressive enough), it can be shown in stronger systems. There is a hierarchy of systems: we say that $T < T'$ when the consistency of T can be shown in T'. It is then possible to design several variant of arithmetic or axiomatic theories and form a whole hierarchy of formal systems. Reverse mathematics, founded by Harvey Friedman [Fri75, Fri76], searches for the sufficient axioms to express some mathematical theories (it is a *reverse* approach in the sense that it starts from mathematical practice to reach axioms instead of starting from given axioms). From this new approach, it is possible to study the mathematics which are "finitistically" meaningful or reducible to finite mathematics [Sim88]. In particular, a system known as PRA (Primitive Recursive Arithmetic) embodies all of finitistic mathematics [Tai81].

§5.8 This is the end of this historical introduction. Now, I suggest formal presentations of the ideas described until now. These presentations are strongly influenced by my own education on logic. I start from a modern formalisation of stoic logic known as *propositional calculus* and extend it to obtain richer logical systems. Then proof systems for these systems are formally presented.

6 Propositional calculus

§6.1 The propositional calculus can be understood as turning stoic logic (*cf.* Paragraph 1.6) into a system of symbols. We fix a set `Atoms` of atomic statements called *atoms* (these are stoics' assertibles), written with lower case letters a, b, c, d etc. From propositions, it is possible to construct more complex statements called *formulas* or *propositions*:

- if $a \in$ `Atoms`, then a is a formula;
  - ◇ **Example of sentence:** "the sky is blue".
- if A is a formula, then $\neg A$ is a formula (negation);
  - ◇ **Example of sentence:** "the sky is **not** blue".
- if A and B are formulas, then $A \wedge B$ is a formula (conjunction);
  - ◇ **Example of sentence:** "the sky is blue **and** the grass is green".
- if A and B are formulas, then $A \vee B$ is a formula (inclusive[18] disjunction);
  - ◇ **Example of sentence:** "the sky is blue **or** the grass is green" (both can be true).
- if A and B are formulas, then $A \Rightarrow B$ is a formula (implication).
  - ◇ **Example of sentence:** "**if** the glass falls **then** it breaks".

§6.2 This description of formulas is often compactly described by what computer scientists call a *grammar* (*cf.* Appendix A.3), defining how a formula (written A or B) can be constructed:

$$A, B ::= \quad a \mid \neg A \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \qquad \text{(Formulas)}$$

where a is an atom.

§6.3 The problem now is that formulas such as $\neg A \vee B$, $A \Rightarrow B \Rightarrow C$ or $A \vee B \wedge C$ are ambiguous and can be read in different non-equivalent ways. It is then common to proceed to a *militarisation* of syntax by specifying exactly what string of characters are considered well-formed formulas. We will simply allow to put parentheses around connectives to write clearer presentations of formulas and define priorities between connectives:

- If A and B are formulas, then $(\neg A), (A \wedge B), (A \vee B)$ and $(A \Rightarrow B)$ are formulas.
- The order of priority is: $\neg, \wedge, \vee, \Rightarrow$ (for instance $\neg A \vee B$ is equivalent to $(\neg A) \vee B$).
- Binary connectives are *right-associative*, meaning that $A \Rightarrow B \Rightarrow C$ is equivalent to $A \Rightarrow (B \Rightarrow C)$ and idem for the connectives $\wedge$ and $\vee$.

[18] Stoics' original "or" connective was *exclusive*, meaning that only one statement has to be true and not both. With the *inclusive* disjunction, both can be true.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$
0	0	1	0	0	1
0	1		0	1	1
1	0	0	0	1	0
1	1		1	1	1

Figure 6.1: Truth table defining the values of $\neg A, A \wedge B, A \vee B$ and $A \Rightarrow B$ with respects to the value of A and B.

§6.4 Now that we have symbols, we have to interpret them (and thus define their semantics). A proposition is expected to be either true or false (respectively represented by the numbers 1 and 0). Formulas are evaluated by a function $[\![\cdot]\!]$ providing the meaning of symbols. For instance, if both a and b are true, we expect $a \wedge b$ to be true as well. Let $\Omega : \texttt{Atoms} \to \{0, 1\}$ be a function called *valuation* associating a truth value to propositions.

§6.5 **Definition** (Interpretation of propositional formulas)**.** We define the *interpretation* of a formula with respects to its components as follows:

- $[\![a]\!]_\Omega = \Omega(a)$ with $a \in \texttt{Atoms}$ (we directly access the truth value of the atom);
- $[\![\neg A]\!]_\Omega = 1 - [\![A]\!]_\Omega$ (the value is inversed);
- $[\![A \wedge B]\!]_\Omega = \texttt{min}([\![A]\!]_\Omega, [\![B]\!]_\Omega)$ (true only when both values are true);
- $[\![A \vee B]\!]_\Omega = \texttt{max}([\![A]\!]_\Omega, [\![B]\!]_\Omega)$ (only false when everything is false);
- $[\![A \Rightarrow B]\!]_\Omega = [\![\neg(A \wedge \neg B)]\!]_\Omega$.

§6.6 There is an alternative way to describe the meaning of formulas called *truth tables* (Figure 6.1). It is a mechanical way to compute the truth of a formula from the truth values of its components. However, if you have n variables in a formula A, then you need 2^n steps to compute $[\![A]\!]_\Omega$. If you have only 8 variables, it is already 256 steps! Fortunately, there are now several ways to evaluate formulas more efficiently for a computer (SAT solvers).

§6.7 **Implication.** The case of implication has to be made clear because it looks quite mysterious (and very confusing for beginners). The intended meaning for $A \Rightarrow B$ is that it is a law of causality from A to B. It is expected that $A \Rightarrow B$ is false only when this causality is violated. If both A and B are true then the causality holds. If A is false then it does not violate the fact that A being true must implies the truth of B follows. The only possible violation of causality is that A is false but B is true. Hence, the meaning of $A \Rightarrow B$ is that it is impossible to have both A true and B false, which can be written as the incompatibility $\neg(A \wedge \neg B)$.

§6.8 From this simple presentation of logic, it is possible to exhibit some interesting equivalences appearing in logic (for instance, the fact that $\neg\neg A$ is equivalent to A, which should not be too surprising):

- ◇ **Involution of negation.** $[\![\neg\neg A]\!]_\Omega = 1 - (1 - [\![A]\!]_\Omega) = 1 - 1 + [\![A]\!]_\Omega = [\![A]\!]_\Omega$;
- ◇ **De Morgan law I.** $[\![\neg(A \vee B)]\!]_\Omega = 1 - \mathtt{max}([\![A]\!]_\Omega, [\![B]\!]_\Omega) = \mathtt{min}(1 - [\![A]\!]_\Omega, 1 - [\![B]\!]_\Omega) = \mathtt{min}([\![\neg A]\!]_\Omega, [\![\neg B]\!]_\Omega) = [\![\neg A \wedge \neg B]\!]_\Omega$;
- ◇ **De Morgan law II.** $[\![\neg(A \wedge B)]\!]_\Omega = 1 - \mathtt{min}([\![A]\!]_\Omega, [\![B]\!]_\Omega) = \mathtt{max}(1 - [\![A]\!]_\Omega, 1 - [\![B]\!]_\Omega) = \mathtt{max}([\![\neg A]\!]_\Omega, [\![\neg B]\!]_\Omega) = [\![\neg A \vee \neg B]\!]_\Omega$;
- ◇ **Commutativity.** $[\![A \wedge B]\!]_\Omega = [\![B \wedge A]\!]_\Omega$ (and same for $\vee$);
- ◇ **Associativity.** $[\![(A \wedge B) \wedge C]\!]_\Omega = [\![A \wedge (B \wedge C)]\!]_\Omega$ (and same for $\vee$);
- ◇ **Contraposition.** $[\![A \Rightarrow B]\!]_\Omega = [\![\neg B \Rightarrow \neg A]\!]_\Omega$;

§6.9 A question which naturally comes to mind is "do we capture all possible functions on truth values?" How can we express more complex formulas such as exclusive disjunction or equivalence between formulas? Actually, our system is already powerful enough to speak about any combination of truth we wish. For instance, we could also define:

- ◇ **Exclusive disjunction.** $A \oplus B := (A \vee B) \wedge \neg(A \wedge B)$;
- ◇ **Equivalence.** $A \Leftrightarrow B := (A \Rightarrow B) \wedge (B \Rightarrow A)$.

§6.10 Restricting connectives to few ones (for instance negation and conjunction) is sufficient to retrieve the other connectives. Such set of connectives are said to be *functionally complete*. Here are some examples of functionally complete sets of connectives:

- From $\{\neg, \wedge\}$: $A \vee B := \neg(\neg A \wedge \neg B)$ and $A \Rightarrow B := \neg(A \wedge \neg B)$;
- From $\{\neg, \vee\}$: $A \wedge B := \neg(\neg A \vee \neg B)$ and $A \Rightarrow B := \neg A \vee B$;
- From $\{\neg, \Rightarrow\}$: $A \wedge B := \neg(A \Rightarrow \neg B)$ and $A \vee B := \neg A \Rightarrow B$.

It is even possible to have a unique connective: $\{\uparrow\}$ is functionally complete for $A \uparrow B := \neg(A \wedge B) \equiv \neg A \vee \neg B$ called "Scheffer stroke". Such functions are usually attributed to Henry M. Sheffer in 1913 [Sch65] although Peirce[19] had similar (unpublished) results in 1880.

§6.11 Formulas true for any valuations are called *tautologies*, *theorems* or *valid formulas*, and the ones which are always false are called *antilogies* or *contradictions*. For instance, it is easy to check with a truth table that $a \Rightarrow a$ or $\neg\neg a \Leftrightarrow a$ are tautologies, and that $\neg(a \Rightarrow a)$ which is equivalent to $\neg a \wedge a$ is an antilogy. In particular, $a \Rightarrow a$ which states that a statement implies itself, is equivalent to $\neg a \vee a$ stating that any statement is either true of false. We write $\models A$ when A is a tautology and $\not\models A$ when it is not (note that it does not necessarily mean that it is an antilogy).

[19] It seems that Peirce independently anticipated a lot of things.

§6.12 If $\not\models A$, then we must have some valuation which makes the formula A false. This corresponds to the idea of *counter-example*. To show that a statement is not true, we have to provide a situation for which it is false. For instance, $\not\models a \Rightarrow b$ because if a is true but b is false, it makes the whole formula $a \Rightarrow b$ false.

§6.13 Tautologies are actually generic statements which are true whatever the truth values we put on the atoms. Actually, for that reason, it is even possible to replace atoms by *any formula* and it would still hold. Mathematically speaking, if $\models A$ then $\models A\{a := B\}$ where a is an atom of A and $A\{a := B\}$ is A after replacing all occurrences of a by some formula B. This is called the *substitution lemma*. This shows that there is a slight confusion about the generic nature of tautologies. Although we state tautologies about specific atoms, they actually holds for any statements replacing the atoms. This idea of genericity will only be clear later, in second-order logic.

§6.14 **Logical constants.** A common extension of the propositional calculus is to turn the truth values into constants of the language of formulas itself. We consider two constants 1 (true) and 0 (false) such that $[\![1]\!]_\Omega = 1$ and $[\![0]\!]_\Omega = 0$ for any Ω. Truth values behaves like neutral elements: we have $[\![A \vee 0]\!]_\Omega = [\![A]\!]_\Omega$ and $[\![A \wedge 1]\!]_\Omega = [\![A]\!]_\Omega$. This allows to consider *closed formulas* which are independent of an interpretation of variables and which can be interpreted as such.

7 Predicate calculus

§7.1 Predicate calculus is the logic underlying Frege's Begriffsschrift (*cf.* Paragraph 2.4) and it embodies the paradigm of Frege's philosophy in the sense that we distinguish a class of *individuals* represented by *terms* with which we associate a *meaning* in some given *universe*. For instance, the symbol a may represent *Aristotle* and $f(a)$ may refer to Aristotle's father. We then reason about properties on individuals with symbol called *predicates* associated with a boolean function. For instance, we can define a predicate H to represent the property of being human, which returns either *true* or *false* when given an individual as argument. The formula $H(a)$ then expresses the fact that Aristotle is a human.

§7.2 **Definition** (Relational signature)**.** A *relational signature* is a tuple $\mathbb{R} = (V, PR, F, \mathtt{ar})$ where $(V, F, \mathtt{ar})$ is a signature (*cf.* Appendix B) and PR is a set of function symbols called *predicate symbols* such that $\mathtt{ar}$ is defined on $PR \uplus F$.

§7.3 **Definition** (Formula of predicate calculus)**.** Let $\mathbb{R} = (V, PR, F, \mathtt{ar})$ be a relational signature. Formulas are defined by the following grammar where $t_1, ..., t_n$ are terms (*cf.* Appendix B), $P \in PR$ is a predicate symbol such that $\mathtt{ar}(P) = n$ and $x \in V$ is a variable:

$$A, B ::= \quad P(t_1, ..., t_n) \mid \neg A \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \mid \forall x.A \mid \exists x.A \qquad \text{(Formulas)}$$

§7.4 **Example.** The term $f(a)$ presented above can be defined in a signature $\mathbb{S} = (V, F, \mathtt{ar})$ where $V = \emptyset$, $F = \{f, a\}$, $\mathtt{ar}(f) = 1$ and $\mathtt{ar}(a) = 0$. The formula $H(a) \Rightarrow H(f(a))$ is defined on relational signature $\mathbb{R} = (V, PR, F, \mathtt{ar})$ where $V = \emptyset$, $PR = \{P\}$, $F = \{f\}$, $\mathtt{ar}(H) = 1$, $\mathtt{ar}(f) = 1$ and $\mathtt{ar}(a) = 0$.

§7.5 Because of the presence of variables, it can happen that a variable is not related to a quantifier such as in $\forall x.P(x, y)$. Such variables are said to be *free*. It is still possible to interpret these formulas by providing an interpretation of free variables but to keep things simple, we will simply exclude formulas containing free variables. Such formulas are called *closed formulas*.

§7.6 **Definition** (Free variables and closed formulas)**.** Let A be a formula. We define inductively the set $\mathtt{fv}(A)$ of its free variables:

$$\mathtt{fv}(P(t_1, ..., t_n)) = \bigcup_{i=1}^{n} \mathtt{vars}(t_i) \qquad \mathtt{fv}(\neg A) = \mathtt{fv}(A) \qquad \mathtt{fv}(A \wedge B) = \mathtt{fv}(A) \cup \mathtt{fv}(B)$$

$$\mathtt{fv}(A \vee B) = \mathtt{fv}(A) \cup \mathtt{fv}(B) \qquad \mathtt{fv}(A \Rightarrow B) = \mathtt{fv}(A) \cup \mathtt{fv}(B)$$

$$\mathtt{fv}(\forall x.A) = \mathtt{fv}(A) - \{x\} \qquad \mathtt{fv}(\exists x.A) = \mathtt{fv}(A) - \{x\}$$

We say that A is *closed* when $\mathtt{fv}(A) = \emptyset$.

§7.7 What we do in predicate calculus is turning some linguistic features into symbols in a more articulate way than in propositional calculus. For instance, the expression *"all flowers are beautiful"* would be represented by a single atom in propositional calculus but could be represented by the more accurate $\forall x.F(x) \Rightarrow B(x)$ in predicate logic. Reality is thought to be accessed by putting into words what we can express by language. Accordingly to Frege's philosophy, symbols refer to some reality. This is represented by the notion of *structure* which interprets symbols in a specific universe (*e.g.* the universe of numbers, humans etc).

§7.8 **Definition** (Structure)**.** Let $\mathbb{R} = (V, PR, F, \mathtt{ar})$ be a relational signature. An $\mathbb{R}$-*structure* S (or simply a *structure* when the signature is implicit) is a tuple (U, I) where U is a non-empty set of elements called the *universe* (or *domain*) and I is an *interpretation function* defined on $PR \uplus F$ such that it associates a function $I(f) : U^n \to U$ for function symbols $f \in F$ such that $\mathtt{ar}(f) = n$ and a relation $I(P) : U^n \to \{0, 1\}$ for predicate symbols $P \in PR$ such that $\mathtt{ar}(P) = n$.

§7.9 **Example.** The atom $H(a)$ can be interpreted in the structure $S = (U, I)$ where $U = \{\mathtt{Aristotle}, \mathtt{Plato}, \mathtt{Rock}\}$ and the interpretation is defined by $I(a) = \mathtt{Aristotle}$ and $I(H)(x) = 1$ when $x \in \{\mathtt{Aristotle}, \mathtt{Plato}\}$ and 0 otherwise. It can also be interpreted in a structure $S' = (U', I')$ where $U' = \mathbf{N}$, $I'(a) = 0$ and $I'(H)$ is the parity predicate.

§7.10 We usually consider formulas implicitly closed since they are the only ones which will be considered in our definitions.

§7.11 **Definition** (Interpretation of closed formulas)**.** The *interpretation* of a closed formulas *w.r.t.* a structure $S = (U, I)$ is defined as follows:

- $[\![P(t_1, ..., t_n)]\!]_S = I(P)(I(t_1), ..., I(t_n))$;
- $[\![\forall x.A]\!]_S = \mathtt{min}_{t \in U}[\![\{x := t\}A]\!]_S$;
- $[\![\exists x.A]\!]_S = \mathtt{max}_{t \in U}[\![\{x := t\}A]\!]_S$;
- the other connectives are defined as in propositional logic.

We require that $\{x := t\}A$ is A for which all occurrences of x are replaced by a term t.

§7.12 **Remark.** We have $[\![\forall x.A]\!]_S = [\![\neg(\exists x.\neg A)]\!]_S$ and $[\![\exists x.A]\!]_S = [\![\neg(\forall x.\neg A)]\!]_S$.

§7.13 We write $M \models A$ for a structure M and a formula A when $[\![A]\!]_M = 1$ and say that M is a *model* of A. In case it is not a model of A, we simply write $M \not\models A$. For instance, given the model M of the previous example, we have $M \models H(a)$ because $[\![H(a)]\!]_M = I(H)(I(a)) = I(H)(\mathtt{Aristotle}) = 1$. If for a given formula A, there exists a model M such that $M \models A$, we say that A is *satisfiable*. If there exists M such that $M \not\models A$, we say that A has a *counter-model*. As in propositional calculus, a formula of predicate calculus is a *tautology* or *valid* when $M \models A$ for any model M, which can be written $\models A$.

§7.14 **Equality.** The *equality predicate* which is essential in mathematics becomes a mere binary predicate $=$ which is true if and only if the inputs correspond to identical objects in a given structure, *i.e.* $\models [\![a = b]\!]_S$ for $S = (U, I)$ when $I(a) = I(b)$. It shows that equality in syntax is just the reflection of equality in semantics. It is fair to ask whether it is a right definition or not considering how important equality is in mathematics. Is it right to define it simply as a symbol like other ones? It looks like we are only delegating the problem of equality in syntax to the implicit equality of semantics.

§7.15 Do quantifiers really add something more than propositional calculus? Universal quantification can naturally be seen as a n-ary conjunction over the n elements of a structure. For instance, the formula $\forall x.P(x)$ can be encoded by the formula $\bigwedge_{i=1}^{n} P(a_i)$ for a structure of n elements $a_1, ..., a_n$. This poses no problem for the finite case and the two formulas are equivalent. However, for the case of infinite models, we have to switch to *infinitary propositional logic* [Moo97] instead. Even in this case, universal quantification and infinitary conjunction are different. For instance, the formula $\bigwedge_{i=1}^{\infty} P(a_i)$ only has infinite models for constants $a_1, a_2, ...$, but $\forall x.P(x)$ can be true in finite models as well as infinite models. Predicate calculus actually subsumes propositional calculus. If we remove quantifiers and restrict predicates to the nullary case only (no argument), we exactly obtain propositional calculus. Universes must also be restricted to the set $\{0, 1\}$ so that structures corresponds to a truth interpretation.

§7.16 **Empty models.** Let us now discuss about a small but important detail: the exclusion of *non-empty* models. Actually, this is a technical requirement. Consider an empty model $M_\emptyset = (\emptyset, I)$. If we have a universal formula $\forall x.P(x)$, it states that P is true for all x in the model. But what happens if the model is empty? It is usually considered that $M_\emptyset \models \forall x.P(x)$. This can be justified by the fact that 1 is the neutral elements for $\wedge$ or that it is impossible to find an element of the model for which $[\![P(x)]\!]_{M_\emptyset} = 0$ which would contradict the formula. In particular, we have $M_\emptyset \not\models (\forall x.P(x)) \Rightarrow (\exists x.P(x))$ because the premise is true but not the conclusion. The problem is that this formula is actually provable in usual proof systems (in syntax). This makes the correctness formula *false* in this case. The usual (ad-hoc) solution is simply to forbid the empty model. This choice does not seem to be philosophically justified.

8 Second-order logic

§8.1 It happens in the practice of mathematics that we need to quantify over predicates. For instance the formula $\exists P.P(a)$ states that there is some predicate P which holds for the constant a. Since unary predicates can be seen as set of individuals (for which the predicate holds), extending quantification to predicates corresponds to quantifying over sets, which is strictly stronger than predicate calculus. In Tarskian terms, set theory can be chosen as a metalanguage for second-order logic (but other semantic theory can be chosen as well).

§8.2 Second-order logic[20] corresponds to predicate calculus extended to quantification over predicates. It has also been introduced by Frege who introduced the expression of *second-order* ("zweiter Ordnung" in German) [Fre82, §53]. When considering the restriction to the propositional case (using nullary predicates), the quantification applies on propositions and the atomic formulas correspond to variables. For this reason we use a notation of variable in the following definition of the second-order formulas.

§8.3 The formulas of second-order logic are defined by the following grammar:

$$A, B ::= \quad X_i(t_1, ..., t_n) \mid \neg A \mid A \wedge B \mid \forall x.A \mid \forall X_i.A \qquad \text{(Formulas)}$$

where X_i is a predicate symbol, $t_1, ..., t_n$ are terms and $i \in \mathbf{N}$. It is sufficient to restrict the formulas to universal quantification since existential quantification can be retrieved by Remark 7.12. In the same fashion, the set of connectives $\{\neg, \wedge\}$ is functionally complete (*cf.* Paragraph 6.10) and is thus sufficient to express $\vee$ or $\Rightarrow$.

§8.4 For the interpretation of closed second-order formulas, we extend interpretation of predicate calculus (*cf.* Definition 7.11) to an interpretation of predicate symbols by sets (the

[20] I remember a presentation where Gilles Dowek (jokingly) said that he did not know what second-order logic was since he did not understand what *first-order* logic was.

set of all terms for which the represented property holds). However, as we saw in Section 3, invoking the set of all sets is problematic. Either some properties cannot be represented or we have to choose another foundation (using type theory for instance). We leave all these foundational discussions aside.

§8.5 **Example.** In second-order logic, the predicate `even` corresponding to even natural numbers is interpreted by the set of all even natural numbers $\{n \mid \exists k \in \mathbf{N}, n = 2k\}$.

§8.6 **Definition** (Interpretation of closed second-order formulas)**.** The *interpretation* of a closed second-order formula *w.r.t.* a structure $S = (U, I)$ is defined as follows:

- $[\![\forall X_i.A]\!]_S = \mathtt{min}_P[\![\{X_i := P\}A]\!]_S$ where P is a predicate, and
- other connectives are interpreted in the same way as for predicate calculus.

§8.7 **Indiscernibility principle.** Second-order logic enjoys a different (and more interesting notion of equality). Instead of a simple binary predicate, equality can be defined with the following second-order formula:

$$a = b \text{ is defined as } \forall X.X(a) \leftrightarrow X(b)$$

meaning that a and b are considered equal when they cannot be distinguished for all properties, or in other words, that all property satisfied by a are satisfied for b and vice-versa. This is usually attributed to Leibniz and even called *Leibniz's law* for that reason:

- the implication $(a = b) \Rightarrow \big(\forall X.X(a) \leftrightarrow X(b)\big)$ is known as *indiscernibility of identities* and is trivial[21] (replace a by b or the converse and you obtain the identity $A \Leftrightarrow A$);
- the converse implication $\big(\forall X.X(a) \leftrightarrow X(b)\big) \Rightarrow (a = b)$ is more controversial because of whether or not it is a sufficient principle of individuation. Distinguishing individuals strongly depends on what we mean by property. Are we considering all possible properties? If so, then we are implicitly invoking the forbidden set of all sets. If not, how are we choosing our properties? As Girard remarked [Gir18b, Section 2.1], in $a = a$, one symbol is on the left and the other on the right but no predicate takes spatial position into account. Is it considered logically irrelevant? Why?

In fact, Leibniz's notion of indiscernibility was a metaphysical concept which had nothing to do with formal logic and the correspondence with Leibniz's original thought may be exaggerated. For that reason, I choose to not call this logical principle Leibniz's law but indiscernability principle instead.

[21] In its philosophical forms, it is rejected by Peter Geach who consider a notion of relative equality.

§8.8 **Peano arithmetic.** Peano arithmetic which has often been discussed in the context of formal logic is naturally expressed as a second-order theory. All axioms except the last one are statements of predicate calculus. The last axiom expresses *induction* over natural numbers, which allows to prove a general statement P for all natural numbers only from the base case $P(0)$ and the propagation $P(n) \Rightarrow P(n+1)$. In other words, we have:

$$\forall X.\big(X(0) \wedge (\forall n.X(n) \Rightarrow X(n+1))\big) \Rightarrow \forall n.X(n)$$

We call PA2 Peano arithmetic with second-order induction. A consequence of Gödel's incompleteness theorem is that second-order logic cannot be both sound and complete since it can express Peano arithmetic. However, these properties are valid for predicate calculus which also enjoys additional properties (*e.g.* compactness theorem). For that reason, the last axiom is often replaced by infinitely many axioms of predicate calculus for each predicate P (countably many):

$$\big(P(0) \wedge (\forall n.P(n) \Rightarrow P(n+1))\big) \Rightarrow \forall n.P(n)$$

Peano arithmetic with this infinite axiom scheme is called PA. Since there are countably many predicates in predicate calculus and uncountably many ones in second-order logic, PA2 is theoretically more expressive than PA (some second-order predicate cannot be represented in predicate calculus). However this has no serious effect in practice.

§8.9 Although full second-order logic is too powerful, it is possible to have alternative weaker semantics for which we can retrieve properties we wish for. Reverse mathematics (*cf.* Paragraph 5.7), for instance, studies subsystems of PA2. For more details about the power and problems related to second-order logic, we refer to Väänänen's article on second-order logic [Vää19].

§8.10 It is possible to consider even stronger extensions of logic such as *higher-order logic* which quantifies over nested sets of arbitrary depth instead of simply sets of elements as in second-order logic. In particular, mathematics using quantification over general sets should be expressed in higher-order logic as predicate calculus is not sufficient.

9 Natural deduction

§9.1 In the following sections, I will only mention Gentzen's natural deduction and sequent calculus since they are the most relevant proof systems for this thesis. The idea is that we would like to represent a proof of a statement as a mathematical object. In Figure 4.1, proofs are sequences of operations following some rules but a more structural representation are *trees* since some rules can create several branches (the modus ponens referring to two previous assumptions). This section is inspired by Olivier Laurent's great course notes on proof theory (https://perso.ens-lyon.fr/olivier.laurent/thdem11.pdf).

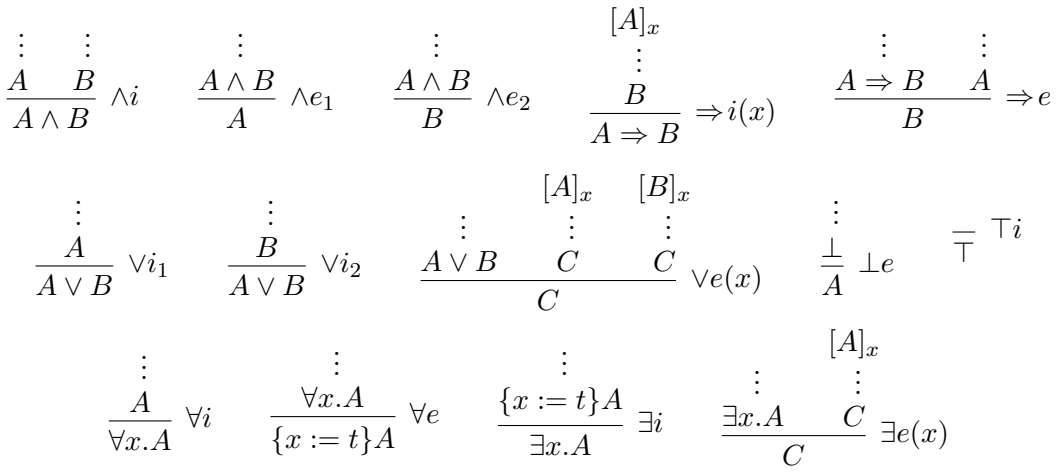


Figure 9.1: Rules of intuitionistic natural deduction (NJ). We require that x is not free in any hypothesis of the rule $\forall i$ and that x is not free in neither C nor any hypothesis for the rule $\exists e$. These requirements avoid weird behaviours related to variables such as identifying the reference of two variables which were previously pointing at different things. The expression $[A]_x$ corresponds to an infinite supply of occurrences of an assumed formula A. These assumptions have to be justified by specifying which rule introduced them. The x in $[A]_x$ refers to a rule identified by a label x. A rule can be linked to any number of assumptions (including 0). These assumptions are called *discharged assumptions*.

§9.2 We will only focus on logic without second or higher-order quantification because this is how proof systems are usually introduced.

Intuitionistic natural deduction

§9.3 Natural deduction has been introduced by Gentzen [Gen35a] as an alternative to axiomatic systems which would be more natural. Such a system should look like as much as possible as the "natural rules of logic". The formulas we consider are defined by the following grammar:

$$A, B ::= X_i \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \mid \forall x.A \mid \exists x.A \qquad i \in \mathbf{N}$$

where $\top$ and $\bot$ corresponds to constants for truth and contradiction. Rules are written several premise formulas on the top of a conclusion formula, separated by a horizontal line. The rules of natural deduction are presented in Figure 9.1. The rules are divided into two classes: *introduction* rules and *elimination* rules. An intuitive way to understand this separation is that introduction rules correspond to *defining* or *constructing* a logical

$$\dfrac{\dfrac{}{[A \vee B]_1} \quad \dfrac{\dfrac{[A]_2}{B \vee A} \vee i_2 \quad \dfrac{[B]_3}{B \vee A} \vee i_2}{B \vee A} \vee e(2,3)}{(A \vee B) \Rightarrow (B \vee A)} \Rightarrow i(1)$$

Figure 9.2: A proof of $(A \vee B) \Rightarrow (B \vee A)$ in natural deduction. The application of the rule $\Rightarrow i(1)$ needs to justify that there is some $A \vee B$ above which would imply $B \vee A$. The 1 shows that we found the required formula in the formula supply $[A \vee B]_1$ of identifier 1.

symbol (if you have A and B, you can construct $A \wedge B$) and elimination rules define the *use* of logical symbols (if you have $A \wedge B$ then you can extract either A or B). The rule $\Rightarrow e$ corresponds to the so-called modus ponens and show how we can use an implication.

§9.4 **Defined negation.** Remark that there is no rule for the negation $\neg A$. This is because the negation can be defined by $\neg A := A \Rightarrow \bot$. It is then sufficient to have rules for implication and contradiction. If we want to explicitly give the rules for negation (which would be redundant), we would have an introduction rule reaching $\bot$ from $[A]$ and inferring $\neg A$, and an elimination rule producing $\bot$ from A and $\neg A$.

§9.5 I comment the rules of NJ (*cf.* Figure 9.1).

- For $\wedge$, we can either *construct* a pair or *destruct* it by extracting a component;
- The rule $\Rightarrow i$ allows to infer $A \Rightarrow B$ when you can reach B with some A existing somewhere above, and $\Rightarrow e$ is the modus ponens which allows to use an implication $A \Rightarrow B$ providing you can feed it with an argument A;
- The rules $\vee i_1$ and $\vee i_2$ make a statement more confusing[22] by encapsulating it in a more general structure so that they can be treated in a uniform way. For instance, a proof of A and B would be treated as the same entity $A \vee B$, thus hiding if it was a proof of A or B. The rule $\vee e$ corresponds to *case analysis*. If we have proven $A \vee B$ then we can look at the possible case: either A or B is true but we do not know which one. If both lead to the same conclusion C then C can be inferred from the whole reasoning;
- The rule $\bot e$, corresponding to the explosion principle (*cf.* Paragraph 1.9), says that from a contradiction, anything can be said[23];
- The rule $\top i$ introduces a statement considered trivially provable;

[22] Why would we do that, logically speaking? This rule will later be justified by its connexion with programming.

[23] I give a brief explanation because people often find it confusing. Assume A and $\neg A$ are true. Then $A \vee B$ must be true since A is true in particular. But since $\neg A$ is true (and hence A false), it follows that B must be true.

- The rule $\forall i$ says that a statement can be *generalised*. The rule $\forall e$ says that a general statement can be *instantiated* to a more specific case;
- The rule $\exists i$ says that if we have a witness t for some formula A then there exists some x (which is t) such that A holds for it (it is similar to the introduction of $\vee$). The rule $\exists e$ which is a bit more confusing behaves exactly like a generalised $\vee e$ rule.

§9.6 Proofs are trees constructed with those rules such that their leaves are formulas corresponding to assumptions. An example of proof is given in Figure 9.2. In particular, the formula A alone (written $[A]_x$ for some x) is a proof by itself because it corresponds to assuming A, then having to prove A, which is trivial[24]. We can finally remark that the only axiom that natural deduction allows is the identity $A \Rightarrow A$.

§9.7 When trying to construct a proof, there are two strategies which imply two different reading of rules. The bottom-up method starts from the conclusion and use rules from the bottom until reaching justified hypotheses only. It is especially natural for implications since the goal tells what hypothesis you need to introduce. This strategy is also called *goal-directed*. The other strategy uses a top-down reading of rules (it is also probably the most natural for mathematicians). We start from assumptions and try to reach a wanted goal. We then have to guess what hypotheses are necessary.

§9.8 Although this system is called "natural", it is not exactly natural in the sense that it does not reflect the practice of mathematicians. Look at the rule $\vee i_1$. What mathematician would prove A then forget that he proved A by making it a proof of $A \vee B$? The same problem happens with $\exists i$. We actually lose information with these operations. We can also observe a duality between rules such as the elimination of conjunction and the introduction disjunction or the elimination and introduction of the universal and existential quantifiers. Those rules are not so innocent since they are what we call *irreversible rules*. When we wish to prove $A \vee B$ in a bottom-up strategy and *choose* either A or B, we can do a mistake and obtain an unprovable goal. The same phenomena happens with $\exists i$ when choosing a wrong existential witness t. The provability is not preserved in a bottom-up use of rules. Moreover, the elimination for these connectives also has a quite weird shape. If we look at the rule $\vee e$ in Figure 9.1, the "flow of reasoning" seems bended: there is a path ending on $A \vee B$ then continuing to the conclusion C and going up to the hypotheses $[A]_x$ and $[B]_x$ which are the connectives subject to the formula $A \vee B$ on which the rule is focussing. This *detour* looks like a compensation for the inability to produce A and B on the bottom (which would break the shape of tree) because of the rigid distinction between hypothesis and conclusion.

[24] When I learned logic for the first time, I was very confused by these discharged assumptions. I thought that since you could start from any assumption without justification, anything could be trivially proven. You want to prove A? Well take a node A and it is done! But actually, either you have unjustified assumptions and they represent hypotheses in the context of an hypothetical reasoning, or assumptions are justified which means they have been correctly introduced by some rule

$$\frac{}{A \vdash A}\ \text{ax} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B}\ \wedge i \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge e_1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge e_2$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee i_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee i_2 \qquad \frac{\Gamma \vdash A \vee B \quad \Delta_1, A \vdash C \quad \Delta_2, B \vdash C}{\Gamma, \Delta_1, \Delta_2 \vdash C}\ \vee e$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow i \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Delta \vdash A}{\Gamma, \Delta \vdash B}\ \Rightarrow e \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot e \qquad \frac{}{\vdash \top}\ \top i$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ \forall i \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash \{x := t\}A}\ \forall e \qquad \frac{\Gamma \vdash \{x := t\}A}{\Gamma \vdash \exists x.A}\ \exists i \qquad \frac{\Gamma \vdash \exists x.A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}\ \exists e$$

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ w_L \qquad \frac{\Gamma, B, B \vdash A}{\Gamma, B \vdash A}\ c_L \qquad \frac{\Gamma \vdash A}{\sigma(\Gamma) \vdash A}\ ex_L$$

Figure 9.3: Rules of intuitionistic natural deduction (NJ). We require $x \notin \mathtt{fv}(\Gamma)$ for $\forall i$ and $x \notin \mathtt{fv}(\Delta \cup \{C\})$ for $\exists e$. The expression $\sigma(\Gamma)$ is the permutation of the order of the formulas in Γ given by some function σ.

$$\begin{array}{c} A_1, ..., A_n \\ \vdots \\ B \end{array} \qquad \rightsquigarrow \qquad \begin{array}{c} \vdots \\ A_1, ..., A_n \vdash B \end{array}$$

Figure 9.4: Shape shifting from the vertical and horizontal (sequents) presentations of rules. Inspired by a picture I once saw in a presentation of Elaine Pimentel.

§9.9 **Sequents.** If you try to prove some statements by yourself, you will probably find that the rules are not very handy to write proof trees. Consider that you want to prove $(A \vee B) \Rightarrow (B \vee A)$ as in Figure 9.2. Either you start from the top and assume $A \vee B$ and try to reach $B \vee A$, which is not very convenient when you want to write a tree (from the root to the leaves) or you start from the bottom but then you have to keep somewhere that you assumed $A \vee B$. We will see that it is even less convenient when considering *classical logic*. Another style of rules also suggested by Gentzen [Gen35a] is to use *sequents* which are expressions $\Gamma \vdash A$ where $\Gamma := \{A_1, ..., A_n\}$ is a set of formulas representing hypotheses. This expression asserts that from Γ, it is possible to infer A. It is expected to have the same meaning as the formula $(A_1 \wedge ... \wedge A_n) \Rightarrow A$. The rules now handle sequents instead of formulas. By doing so, we can keep track of the hypotheses at the left of the symbol $\vdash$ (called "turnstile"). This change of proofs' shape is illustrated in Figure 9.4. The rules for natural deduction in a sequent presentation are given in Figure 9.3. You can compare the two set of rules to convince yourself that they have the same meaning. As expected, the rule $\Rightarrow i$ only introduce the premise on the left part of $\vdash$. Now, since the assumptions are now located in the left part of $\vdash$ and not leaves anymore, we need another terminal rule which is the axiom $A \vdash A$ (as we said before in Paragraph 9.6, the identity is the only axiom considered). Because of this change we also need to represent the fact that we can use as many occurrences of a hypothesis as we want (this corresponds to the fact that a rule can be linked with several occurrences

$$\frac{\neg\neg A}{A}\ \text{dne} \qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}\ \vee i^m \qquad \frac{\Gamma \vdash \Delta, A \vee B}{\Gamma \vdash \Delta, A, B}\ \vee e^m$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ w_R \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ c_R \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \sigma(\Delta)}\ ex_R$$

Figure 9.5: Additional rules of classical natural deduction (NK) for the two presentations (the first rule for the presentation without sequents and the others for the presentation with sequents). As for ex_L, the use of ex_R is usually implicit.

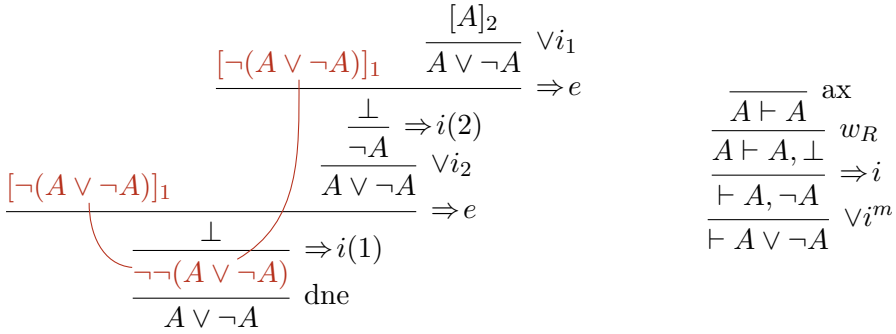


Figure 9.6: Proof of excluded middle for the two presentations of natural deduction (without and with sequents). Recall that we write $\neg A$ for $A \Rightarrow \bot$. Look at how it is much more complicated without sequent where it is not natural to write multiple conclusions. In the left proof, remark that we come back again to the goal $A \vee \neg A$ but together with the hypothesis $\neg(A \vee \neg A)$ which makes it provable.

of a discharged assumption). For that reason, the structural rules w_L (left weakening) and c_L (left contraction) are introduced. We also have a rule ex_L which exchanges the order of formulas (if we want to be very formal) but its use is usually left implicit.

§9.10 **Structural rules.** The weakening rule can be interpreted as making the goal "weaker" by adding useless hypotheses. In a bottom-up reading, it corresponds to erasing useless information when trying to prove a statement. The contraction corresponds to identifying two copies of a formula (top-down) or duplicating formulas (bottom-up). These operations are very common and natural: when trying to prove a statement, it happens that some information are not relevant or that we use several times the same hypothesis. This happens when proving the sequents $A, B \vdash A$ and $A \vdash A \wedge A$.

Classical natural deduction

§9.11 We presented a system for intuitionistic logic, a logic rejecting *classical rules* such as the excluded middle $A \vee \neg A$ (*cf.* Paragraph 4.3). If we try to give a derivation of $\vdash A \vee \neg A$, then we have to prove either A or B, which is impossible for a generic statement A we know nothing about. It is still possible to get the classical statements back:

- in the original notation without sequent, we can add a rule called *double-negation elimination* which infer A from $\neg\neg A$ (which is equivalent to the excluded middle). It is also possible to simply add an axiomatic rule for the excluded middle;
- in the notation with sequent, we allow multiple conclusions and the meaning of a sequent $A_1, ..., A_n \vdash B_1, ..., B_m$ is $(A_1 \wedge ... \wedge A_n) \Rightarrow (B_1 \vee ... \vee B_m)$. In order to prove the excluded middle, we must consider an alternative (but equivalent) rule $\vee i^m$ for disjunction which infers $\Gamma \vdash A \vee B$ from $\Gamma \vdash A, B$.

This yields a system of *classical natural deduction* (NK). The additional rules to add for classical logic are given in Figure 9.5 and a proof excluded middle for the two presentations of NJ is given in Figure 9.6.

§9.12 But what changed by adding these rules? If we look at Figure 9.6, the rule dne over a formula A (with conclusion $\neg\neg A$) allows the introduction of several occurrences of A so that the proof can keep going without "loss of information" (if we use $\vee i$, it forces us to prove either A or $\neg A$ and we lose the possibility to use the other). Similarly, for the sequent presentation, the new rules allow an occurrence of A to be introduced as a hypothesis. Classical logic is related to the ability to save a goal for later and switch to another goal, or to *retract* during an attempt to prove a statement[25].

§9.13 This ability to introduce hypotheses can actually be simulated in intuitionistic logic as well by considering Gödel's *double-negation translation*. Even though the excluded middle $A \vee \neg A$ is not provable, it appears that its double-negated version $\neg\neg(A \vee \neg A)$ is provable in intuitionistic logic. The first proof of Figure 9.6 corresponds to an intuitionistic proof of $\neg\neg(A \vee \neg A)$.

Proof reduction

§9.14 In natural deduction, if we introduce a connective then eliminate it, we obtain a redundancy since it is the same as just introducing it (as if we were computing $n + 1 - 1$). It should then be possible to eliminate such redundancies in order to obtain a simpler and irreducible final proof called the *normal form* of a proof. However, it is not so simple to

[25] This idea has been illustrated by Wadler's story of the Devil of excluded middle. The devil suggests an offer: (a) either he gives you a billion dollars (b) or anything you wish provided you give him a billion dollars. He chooses (b). You try to get a billion dollars after few years but then the Devil changes his choice and say that he chooses (a) instead and gives you a billion dollars.

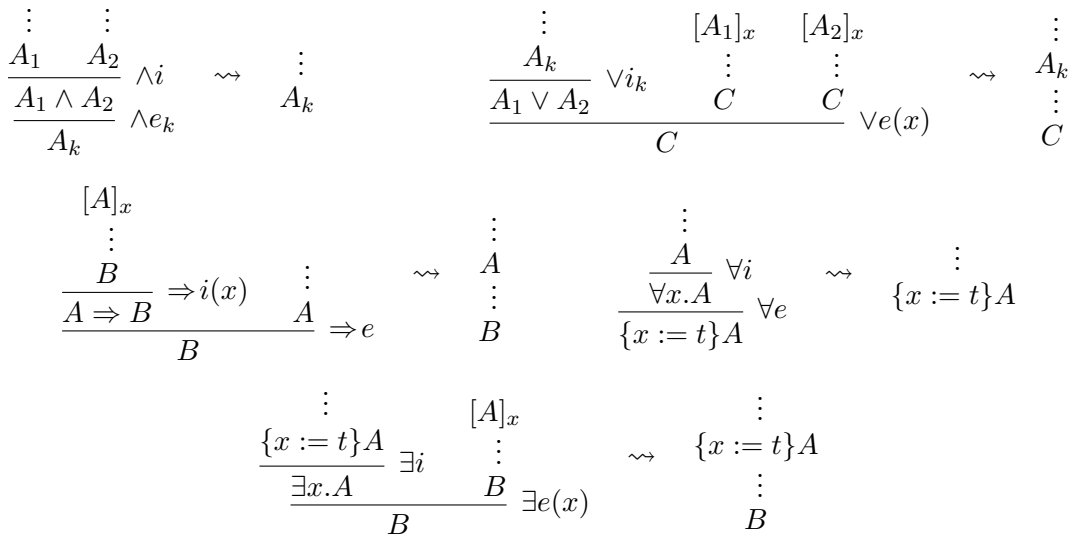


Figure 9.7: Proof reduction for NJ (presentation without sequent) with $k \in \{1, 2\}$.

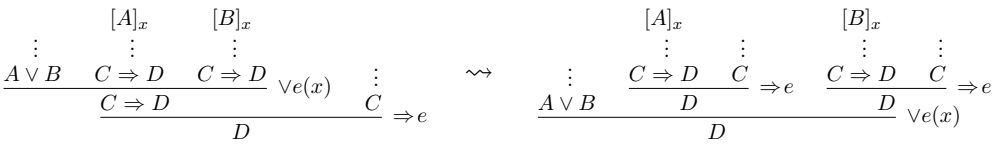


Figure 9.8: Example of commutative case of reduction for the rules $\vee e / \Rightarrow e$ of NJ.

do these proof reductions[26]. Reduction rules for NJ are given in Figure 9.7. The case of $\wedge$ corresponds to extracting a component from a pair we constructed. In the case of $\Rightarrow$, the proof is replaced by of proof of A but instead of coming from the assumption $[A]_x$, it comes from the proof of A on the right. All assumptions $[A]_x$ are then replaced by a proof of A: we have a substitution of proofs which multiply the size of the proof by the number of occurrences of $[A]_x$. In the case of $\forall$, we know that A holds for some possible variables x so we generalise it into $\forall x.A$ but then specialise it with some term t. We can reduce this to a proof of $\{x := t\}A$ which takes the proof of A and replaces its occurrences of variables x by t. The other rules behaves in the same ways as the rules previously described.

§9.15 There exists more subtle cases of reduction which does not correspond to redundancies introduced with an elimination rule following and introduction rule. They correspond to exchange of rules and are called *commutation cases* for that reason. An example of reduction for a commutative case is presented in Figure 9.8. The (problematic) rules

[26] Although it is called *reduction*, it is possible for the normal form to be bigger.

concerned by these commutative cases are the irreversible rules $\vee e$, $\exists e$ and $\bot e$.

§9.16 **Links with computation.** Although this procedure of proof reduction looks innocent, it has deep connexions with computation as we will see later in this thesis. The reduction rule for $\wedge$ corresponds to constructing a pair and extracting a component out of it. The reduction for $\Rightarrow$ corresponds to activating a function by *inlining*, by copying its code and pasting it in the place of all the corresponding function calls. This will establish a formal correspondence between (intuitionistic) proof and (functional typed) programs, presented later in this thesis.

10 Sequent calculus

Classical sequent calculus

§10.1 Gentzen's original goal was to establish consistency results for number theory but it seems that natural deduction was not very convenient for that purpose. For that reason, he introduced another proof system called *classical sequent calculus* (LK). In this system we also have rules handling sequents but instead of introduction and elimination rules, we only have introduction rules but which introduce formulas either on the left (as hypothesis) or on the right (as conclusion) of the symbol $\vdash$. Sequent calculus has redundancies since several sequent calculus proofs can correspond to a single natural deduction proof. As we will see, sequent calculus proofs allow a more fine-grained analysis of reasoning. As for the sequent presentation of NJ, a restriction of conclusion to a single formula yields intuitionistic sequent calculus (LJ). The rules of LK are presented in Figure 10.1a. As for natural deduction, we have identity rules (although one is added) and structural rules. Logical rules are divided into two equivalent variants: additive and multiplicative rules.

§10.2 **Additive and multiplicative rules.** Additive (marked with a) and multiplicative rules (marked with m) are two variants of the same logical concept. If we look at $\wedge R^m$ and $\wedge R^a$ in bottom-up reading, the former *splits* its contexts and the latter *shares* its contexts. Since structural rules (which are the same as for natural deduction) allow the erasure and duplication formulas, the two variants can be shown to be equivalent. This equivalence is illustrated by an example in Figure 10.3. This distinction emphasises the point of view of formulas as resources which is essential in linear logic. Duplicating formulas ensures that we will not have a shortage of resources and erasing formulas ensures that we get rid of superfluous information so that we can reach the axiom $A \vdash A$. In strictly linear logic which see formulas as *limited* resources, structural rules are forbidden[27] and multiplicative and additive rules become truly distinct logical concepts with their own handling of formulas, seen as resources.

[27] But allowed with more control in full linear logic.

$$\frac{}{A \vdash A}\ \mathrm{ax}^m \qquad \frac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2, A \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \mathrm{cut}^m \qquad \frac{}{\Gamma, A \vdash A, \Delta}\ \mathrm{ax}^a \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}\ \mathrm{cut}^a$$

(a) Multiplicative and additive identity rules.

$$\frac{\Gamma \vdash \Delta}{\sigma(\Gamma) \vdash \sigma'(\Delta)}\ \mathrm{ex} \qquad \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ w_L \qquad \frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ c_L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ w_R \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ c_R$$

(b) Structural rules.

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ \neg L \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ \neg R$$

(c) Rules for negation.

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L^m \qquad \frac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash A \wedge B, \Delta_1, \Delta_2}\ \wedge R^m$$

$$\frac{\Gamma_1, A \vdash \Delta_1 \qquad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \vee B \vdash \Delta_1, \Delta_2}\ \vee L^m \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R^m$$

$$\frac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \Rightarrow B \vdash \Delta_1, \Delta_2}\ \Rightarrow L^m \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\ \Rightarrow R^m$$

$$\frac{}{\vdash \top}\ \top R^m \qquad \frac{}{\bot \vdash}\ \bot L^m \qquad \frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta}\ \top L^m \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \bot R^m$$

(d) Multiplicative logical rules.

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L_1^a \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L_2^a \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge R^a$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee L^a \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R_1^a \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R_2^a$$

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \Rightarrow B \vdash \Delta}\ \Rightarrow L^a \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\ \Rightarrow R_1^a \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\ \Rightarrow R_2^a$$

$$\frac{}{\Gamma, \bot \vdash \Delta}\ \bot L^a \qquad \frac{}{\Gamma \vdash \top, \Delta}\ \top R^a$$

(e) Additive logical rules.

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}\ \exists L \qquad \frac{\Gamma \vdash \{x := t\}A, \Delta}{\Gamma \vdash \exists x.A, \Delta}\ \exists R \qquad \frac{\Gamma, \{x := t\}A \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\ \forall L \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x.A, \Delta}\ \forall R$$

(f) Rules for quantifiers. We require $x \notin \mathtt{fv}(\Gamma)$ for $\forall R$ and $\exists L$.

Figure 10.1: Rules of classical sequent calculus (LK).

$$\frac{\Gamma \vdash A, \Delta \qquad \dfrac{}{\bot \vdash}\ \bot L^m}{\Gamma, \neg A \vdash \Delta}\ \Rightarrow L^m \qquad\qquad \frac{\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A \vdash \bot, \Delta}\ \bot R^m}{\Gamma \vdash \neg A, \Delta}\ \Rightarrow R^m$$

Figure 10.2: Simulation of negation rules with other LK rules.

$$\frac{\dfrac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, \Gamma, A \wedge B}\ \wedge^m}{\vdash \Gamma, A \wedge B}\ \mathrm{c} \qquad\qquad \frac{\dfrac{\vdash \Gamma, A}{\vdash \Gamma, \Delta, A}\ \mathrm{w} \qquad \dfrac{\vdash \Delta, B}{\vdash \Gamma, \Delta, B}\ \mathrm{w}}{\vdash \Gamma, \Delta, A \wedge B}\ \wedge^a$$

(a) Simulation of $\wedge^a$ with $\wedge^m$. (b) Simulation of $\wedge^m$ with $\wedge^a$.

Figure 10.3: Equivalence between additive and multiplicative conjunction. The double line represents multiple applications of a same rule.

§10.3 I comment the interesting rules of LK (*cf.* Figure 10.1). It is mostly the rules of "problematic" connectives such as $\vee$ and $\exists$ that change. The rules for $\wedge$ and $\forall$ are the same as in natural deduction.

- Notice the presence of a new rule: the *cut rule*. The cut rule takes its name from the fact that, in a top-down reading, the formula A disappears during inference in order to connect two goals. The meaning of this rule is that if A is proven in some context (left sequent) and that can be used in another context (right sequent), then we can continue the proof in the composition of these two contexts by using A. Hence, the cut is a short*cut* using A as a bridge in reasoning. This can also be understood as the use of a lemma A in a bottom-up reading.
- The rules for negation $\neg$ can be confusing. They can actually be defined with $\neg A := A \Rightarrow \bot$ and other LK rules as shown in Figure 10.2 (hence they are redundant). The rule $\neg L$ says that if you can prove A among other alternative goals Δ, then it is valid to turn A into an hypothesis $\neg A$. If A was a provable goal, then the provability is given by contradiction, and otherwise a goal of Δ must be provable by validity of the disjunction. As for $\neg R$, if a goal is provable with A as hypothesis, then it is valid to turn A into a goal $\neg A := A \Rightarrow \bot$ which would introduce a hypothesis A and an unprovable goal $\bot$ (but it does not matter since Δ was provable by itself);
- For the rule $\vee L^a$, remark that it is nicer than $\vee e$ because $A \vee B$ is allowed to appear on the bottom sequent (since it is introduced on the left and not eliminated from the top). Hence, we can do the same reasoning without "bending" the structure of the rule as for $\vee e$. The multiplicative variant $\vee R^m$ corresponds to classical disjunction which keeps the two goals available (hence removing the need for "data saving").
- The rule $\exists L_a$ is similar to $\vee L^a$. If you can prove your goal with a generic A then you should be able to prove it with the fact that there is some x such that A holds;

- The rule $\Rightarrow L^m$ still is modus ponens but in another form. If you are able to prove A in some context and know that B is the key argument to prove your goal Δ, then you know that $A \Rightarrow B$ is a relevant tool to infer Δ.
- The rule $\bot L^a$ corresponds to the explosion principle. If you have $\bot$ as hypothesis then your statement is vacuously proven.

§10.4 **Subformula property.** It appears that proofs of a sequent $\Gamma \vdash \Delta$ only use subformulas occurring in $\Gamma \cup \Delta$ *except* when the cut rule is present. This property is called *subformula property*[28]. The meaning of this property is rather profound: it means that everything we need is *already there*. A proof enjoying the subformula property does not rely on something else than subformulas of its conclusion. There is no need to summon something external to the proof (in our intuition or culture of mathematics for instance).

§10.5 **Cut-elimination theorem.** Gentzen's showed that the cut rule, representing the use of lemma in mathematical practice was in fact... useless. It can be eliminated, exactly like the redundancies of natural deduction. The proof of the cut-elimination theorem (also called *Hauptsatz*) gives rise to a procedure of proof reduction. Let us now mention why sequent calculus is interesting regarding semantic results. If there is a proof of $\bot$, then there must be a cut-free proof of $\bot$ (by cut-elimination). However, there is no cut-free proof of $\bot$ because of the subformula property (which is a key tool for the cut-elimination theorem) and because no rule introduces $\bot$. It follows that there cannot be a proof of $\bot$, which is the consistency property. We do not give the reduction rules since they are quite complicated. A simplification of LK will be presented later, in which reduction rules are way simpler.

§10.6 **Symmetry.** Notice how LK rules enjoy a very beautiful symmetry. The origin of this symmetry is the separation between left and right introduction rules. The rules $\neg L$ and $\neg R$ show that the symbol $\vdash$ acts as an inversion of *point of view*. This is due to involution of negation occurring in classical logic: we have $A = \neg\neg A$ (we can transfer a formula on the left then on the right or vice-versa and obtain the formula we started with). This duality is appears in the left/right distinction of rules. The rule $\wedge R^m$ and $\vee L^m$ have the same operational behaviour except that the former works on the right and the latter on the left. Logical operations behave exactly like how our movements are reproduced by a mirror. A conjunction on the left is a disjunction on the right and vice-versa. The same phenomena occurs with the other rules of the pair $\wedge/\vee$ and $\forall/\exists$. It shows that it is technically possible to just put every formulas on the right with negation and forget half the rules we presented. This symmetry was not present in NJ because NJ enjoys a *top-bottom* symmetry. In Figure 9.1, look at how the elimination rules for $\wedge$ are upside-down versions of the introduction rules for $\vee$. However, as mentioned in Paragraph 9.8, this symmetry is not natural for elimination rules since we cannot produce two conclusions A

[28] It does not hold for NJ since some rules such as the elimination of disjunction $\vee e$ can introduce formulas which are not necessarily subformulas of C. For instance, if C is always provable (*e.g.* $D \Rightarrow D$) then any formulas can take the place of A and B, including formulas more complex than C.

$$\frac{}{\vdash A^\perp, A}\ \text{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta}\ \text{cut}^m \qquad \frac{\vdash \Gamma, A, A}{\vdash \Gamma, A}\ \text{c} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, A}\ \text{w}$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B}\ \wedge^m \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\ \wedge^a \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B}\ \vee^m$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \vee B}\ \vee_1^a \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \vee B}\ \vee_2^a \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x.A}\ \forall \qquad \frac{\vdash \Gamma, \{x := t\}A}{\vdash \Gamma, \exists x.A}\ \exists$$

$$\frac{}{\vdash \top}\ \top^m \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ \bot^m \qquad \frac{}{\vdash \Gamma, \top}\ \top^a$$

Figure 11.1: (Right) Monolateral rules of classical sequent calculus (LK). We require $x \notin \mathtt{fv}(\Gamma)$ for $\forall$. I choose to exclude the additive axiom rule because we will not need it.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg A, A}\ \text{ax} \quad \dfrac{\dfrac{}{\vdash \neg B, B}\ \text{ax} \quad \dfrac{}{\vdash \neg C, C}\ \text{ax}}{\vdash B \wedge \neg C, \neg B, C}\ \wedge^m}{\vdash A \wedge (B \wedge \neg C), \neg A, \neg B, C}\ \wedge^m}{\vdash A \wedge (B \wedge \neg C), \neg A, \neg B \vee C}\ \vee^m}{\vdash A \wedge (B \wedge \neg C), \neg A \vee (\neg B \vee C)}\ \vee^m}{\vdash (A \wedge (B \wedge \neg C)) \vee (\neg A \vee (\neg B \vee C))}\ \vee^m$$

Figure 11.2: Proof of $((A \wedge B) \Rightarrow C) \Rightarrow (A \Rightarrow (B \Rightarrow C))$ in (right) monolateral sequent calculus. If we push negations, the formula becomes $\neg((A \wedge B) \Rightarrow C) \vee (A \Rightarrow (B \Rightarrow C)) = \neg(\neg(A \wedge B) \vee C) \vee (\neg A \vee (\neg B \vee C)) = (\neg(\neg A \vee \neg B) \wedge \neg C) \vee \neg A \vee \neg B \vee C = (A \wedge B \wedge \neg C) \vee \neg A \vee \neg B \vee C$.

and B (which would break to shape of tree) and have to *bend* the rule with an auxiliary formula C.

11 Monolateral sequent calculus

§11.1 **The logical space of interaction.** Thanks to the nice symmetries of LK and the fact that the symbol $\vdash$ acts like a mirror, it is possible to forget one side and just consider the other as suggested in Paragraph 10.6. If we take the remark seriously, this yields a *monolateral* sequent calculus. The rules of this new system are presented in Figure 11.1. Although it is possible to only consider formulas on the left, we usually consider formulas on the right. We obtain a unique rule for each connective and consider a multiplicative cut which is more standard. Sequents are therefore collections of formulas $\vdash \Gamma$. Working on this new calculus has few interesting consequences:

- there is no rule for negation (because no way to go the left of $\vdash$ anymore) and negation only appears on atomic formulas by pushing the negation with double-negation elimination ($\neg\neg A \rightsquigarrow A$) and De Morgan's laws. The set of formulas becomes:

$$A, B ::= X_i \mid \neg X_i \mid A \wedge B \mid A \vee B \mid \forall x.A \mid \exists x.A \qquad i \in \mathbf{N};$$

- a sequent is a sort of *space of interaction* where atomic formulas interact with their negation by the axiom rule;
- there is no distinction between a negated conclusion and a hypothesis. Input hypotheses can then be seen as *demand* for a formula and the axiom rule connects supply and demand;
- the implication $A \Rightarrow B$ must be defined as $\neg A \vee B$ since there is no hypotheses anymore.

I give an example of proof of monolateral sequent calculus in Figure 11.2.

§11.2 **A simpler analysis of sequent calculus.** Thanks to this monolateral version, it is possible to provide a simple interpretation of rules and logical principles.

◇ **Implication** If we look at $A \Rightarrow B = \neg A \vee B$, the bottom-up reading of the rule $\vee^m$ gives $\vdash \neg A, B$ which can be understood as the demand for A with B waiting in the space of interactions.

◇ **Cut rule** The cut rule joins two spaces of interaction along compatible formulas. It is similar as plugging a male cable with a female cable in electronics.

◇ **Conjunction** Multiplicative conjunction *splits* by distributing its interaction context whereas additive conjunction *shares* it. They are both equivalent thanks to structural rules.

◇ **Disjunction** Multiplicative disjunction $A \vee B$ keeps A and B separated but living in the same space of interaction whereas additive disjunction only keeps one side and deletes the other. They are both equivalent thanks to structural rules.

◇ **Neutral elements** The constant $\top$ corresponds to having a trivial goal and the constant $\bot$ corresponds to having an unprovable goal which can be discarded.

§11.3 **Comparing natural deduction and sequent calculus.** Something I find very instructive is how natural deduction rules can be translated into sequent calculus. First, introduction rules are exactly the same (nothing surprising) and only elimination rules are translated. Some cases of translation are presented in Figure 11.3. First, remark that a cut always appears. If elimination rules symbolise the *use* of a symbol, then cuts are also related to use. Actually, sequent calculus concentrates all the use of symbols in the cut rules and everything else is definition (introduction of symbols). Recall that cuts represent the use of a lemma, then sequent calculus turns the use of symbols into the use

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge e_1 \quad \rightsquigarrow \quad \frac{\Gamma \vdash A \wedge B \qquad \dfrac{\dfrac{}{A \vdash A}\,\mathrm{ax}}{A \wedge B \vdash A}\,\wedge L_1^a}{\Gamma \vdash A}\,\mathrm{cut}^m$$

$$\frac{\Gamma \vdash A \vee B \qquad \Delta_1, A \vdash C \qquad \Delta_2, B \vdash C}{\Gamma, \Delta_1, \Delta_2 \vdash C}\,\vee e \quad \rightsquigarrow \quad \frac{\Gamma \vdash A \vee B \qquad \dfrac{\dfrac{\Delta_1, A \vdash C}{\Delta_1, \Delta_2, A \vdash C}\,w_L \qquad \dfrac{\dfrac{\Delta_2, B \vdash C}{\Delta_1, \Delta_2, B \vdash C}\,w_L}{\Delta_1, \Delta_2, A \vee B \vdash C}\,\vee L_1^a}{\Gamma_1, \Delta_1, \Delta_2 \vdash C}\,\mathrm{cut}^m$$

$$\frac{\Gamma \vdash A \Rightarrow B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}\,\Rightarrow e \quad \rightsquigarrow \quad \frac{\Gamma \vdash A \Rightarrow B \qquad \dfrac{\Delta \vdash A \qquad \dfrac{}{B \vdash B}\,\mathrm{ax}}{\Delta, A \Rightarrow B \vdash B}\,\Rightarrow L^m}{\Gamma, \Delta \vdash B}\,\mathrm{cut}^m$$

Figure 11.3: Translation of the rules $\wedge e_1$ and $\Rightarrow e$ of NJ (sequent presentation) into sequent calculus proofs. The double line represents multiple applications of a same rule.

of lemmas. The left premise of the cut defines/introduces a symbol and the right premise is a lemma explaining how that symbol is able to prove our goal. Sequent calculus is often said to be *analytic*: it decomposes the steps of natural deduction into more elementary operations. In particular, a natural deduction proof potentially corresponds to several different sequent calculus proofs[29] (permuting the order of rules for instance) while there are not so many possible choices in natural deduction proofs.

Cut-elimination procedure

§11.4 In natural deduction, we are interested in removing redundancies coming from the elimination of previously introduced symbols. In sequent calculus, we only have introductions, hence these redundancies disappear. However, as we have seen in the previous translation of NJ into LJ (*cf.* Figure 11.3), elimination rules introduce cuts, hence the elimination of redundancies becomes the elimination of cuts. But more than that, consistently with the fact that redundancies somehow turned into uses of lemmas (which are essentially implications), cut-elimination will naturally looks like the elimination of redundancy for implication: lemmas will be *inlined*, *i.e.* their call is replaced by their proof, similarly to how convenient mathematical notations can be replaced by their more primitive definition. Cut-elimination is a procedure of *explicitation*. The main cases of cut-elimination are presented in Figure 11.4. A very intuitive computational explanation of these reduction rules can be given:

- the elimination ax/cut is an annihilation occurring when an axiom rule interacts with a cut rule, leaving only the other premise of the cut which was unrelated to the interaction;

[29] For that reason, natural deduction is seen as a quotient over sequent calculus proofs.

$$\dfrac{\dfrac{}{\vdash \neg A, A}\,\mathrm{ax} \qquad \vdash A, \Gamma}{\vdash A, \Gamma}\,\mathrm{cut}^m \quad \rightsquigarrow \quad \vdash A, \Gamma$$

$$\dfrac{\dfrac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B}\,\wedge^m \qquad \dfrac{\vdash \Xi, \neg A, \neg B}{\vdash \Xi, \neg A \vee \neg B}\,\vee^m}{\vdash \Gamma, \Delta, \Xi}\,\mathrm{cut}^m \quad \rightsquigarrow \quad \dfrac{\vdash \Gamma, A \qquad \dfrac{\vdash \Delta, B \quad \vdash \Xi, \neg A, \neg B}{\vdash \Delta, \Xi, \neg A}\,\mathrm{cut}^m}{\vdash \Gamma, \Delta, \Xi}\,\mathrm{cut}^m$$

$$\dfrac{\dfrac{\vdash \Gamma, A_1 \quad \vdash \Gamma, A_2}{\vdash \Gamma, A_1 \wedge A_2}\,\wedge^a \qquad \dfrac{\vdash \Delta, \neg A_k}{\vdash \Delta, \neg A_1 \vee \neg A_2}\,\vee^a_k}{\vdash \Gamma, \Delta}\,\mathrm{cut}^m \quad \rightsquigarrow \quad \dfrac{\vdash \Gamma, A_k \quad \vdash \Delta, \neg A_k}{\vdash \Gamma, \Delta}\,\mathrm{cut}^m$$

$$\dfrac{\dfrac{\vdash \Gamma, A}{\vdash \Gamma, \forall x.A}\,\forall \qquad \dfrac{\vdash \Delta, \neg\{x := t\}A}{\vdash \Delta, \exists x.\neg A}\,\exists}{\vdash \Gamma, \Delta}\,\mathrm{cut}^m \quad \rightsquigarrow \quad \dfrac{\vdash \Gamma, \{x := t\}A \quad \vdash \Delta, \neg\{x := t\}A}{\vdash \Gamma, \Delta}\,\mathrm{cut}^m$$

Figure 11.4: Main cases of cut-elimination for the (right) monolateral sequent calculus.

$$\dfrac{\dfrac{\vdash \Gamma, B, C \quad \vdash \Gamma_1, A, C,}{\vdash \Gamma, A \wedge B, C}\,\wedge^a \qquad \vdash \Delta, \neg C}{\vdash \Gamma, \Delta, A \wedge B}\,\mathrm{cut}^m \quad \rightsquigarrow$$

$$\dfrac{\dfrac{\vdash \Gamma, B, C \quad \vdash \Delta, \neg C}{\vdash \Gamma, \Delta, A}\,\mathrm{cut}^m \qquad \dfrac{\vdash \Gamma, A, C \quad \vdash \Delta, \neg C}{\vdash \Gamma, \Delta, B}\,\mathrm{cut}^m}{\vdash \Gamma, \Delta, A \wedge B}\,\wedge^a$$

Figure 11.5: An example of commutation case in cut-elimination for LK.

- the elimination $\wedge^m/\vee^m$ is a rewiring on the premises: the left and right premises respectively interact with each other through an application of cut rule;
- the elimination $\wedge^a/\wedge^a$ corresponds to a choice. Two parallel possibilities are on the left and a selector is given on the right of the cut rule;
- the elimination $\forall/\exists$ corresponds to shifting the interaction between general statements to specific cases.

§11.5 In sequent calculus (independently from NJ's translation), most cases of cut-elimination are *commutations*. They occur in case a cut is applied on a formula which is not introduced by the last rule. For instance, in Figure 11.5, the cut rule is applied on C but it is $A \wedge B$ which has been introduced in the left sequent. It intuitively corresponds to an interaction *at distance*, and which can be resolved by *pushing* the cuts up so that a direct interaction can be done, as if the interaction was obtruded.

§11.6 There are several ways to eliminate cuts and we just suggested one set of rules among others. The idea is always the same: suggesting another proof derivation with the same premises (possibly with duplications or unused premises) and the same conclusion. The reduced proof has to be "simpler". Some sets of cut-elimination rules are "better"

than others for some properties. An important property is *termination* which states that applying the rules will ultimately ends on an irreducible cut-free proof. Another property is *confluence* with states that when there are two ways to reduce a proof, they ultimately ends on the same irreducible cut-free proof. This usually fails with LK for the typical cut-elimination rules but it is possible to hack sequent calculus so to get the desired properties. If you want more about confluence in sequent calculus, I wrote a blog article with Farzad Jafarrahmani (https://prooftheory.blog/2021/09/23/confluence-in-the-sequent-calculus/).

§11.7 **Difference between implication and turnstile.** What is the difference between $\vdash$ and $\Rightarrow$? Did not we say that they had the same meaning? Are we just adding superfluous notations? The expression $A \vdash B$ could be written $A \Rightarrow B$. It is semantically the same. However, sequent calculus would not exist anymore (not in this form at least) as $\vdash$ is a horizontal materialisation of the vertical distinction between hypothesis and conclusion occurring in natural deduction. This distinction is of a *procedural* nature, rooted in the dynamics of the *action of deduction*. The symbol $\vdash$ is part of the inference system or the "machine" while $A \Rightarrow B$ is just a syntactic object handled by the system which is moved back and forth. This difference can be observed in LK rules Figure 10.1. The rule $\Rightarrow R^m$ (which is the same as $\Rightarrow i$ in natural deduction) shows that it is possible to *freeze* the current reasoning $A \vdash B$ and turn it into a static and handleable token $A \Rightarrow B$. Moreover, cut-elimination (which holds all the computational dynamics of logic) only applies on sequents by annihilating formulas in two interacting sequents while the implication $A \Rightarrow B$ is only a *resource* which is added to this inference machine like adding an ingredient to a cooking pot. You can think of it as taking a photo: an instant is captured into something we can exchange, throw away or even duplicate. You can find a small discussion on this subject in Barbarossa's PhD thesis [Bar21, Section 5.3] (he is very fascinated by this distinction).

§11.8 **What about second-order?.** Second-order logic has not been mentioned in the previous technical chapters. If we would like to work with second-order logic, it is sufficient to reuse the rules for first-order quantification and replace the first-order variable by a second-order one. It works correctly when considering the syntactic part of logic but is a bit more complicated and awkward if we would like to consider its relation with a semantic interpretation. Moreover, it appears that we lose analyticity (sub-formula property).

§11.9 **Returning to axiomatic.** If you still want to retrieve an axiomatic system using Gentzen's systems (which would betray his original goal), it is possible to a consider a theory $\mathcal{T}$ (set of axioms). We then have additional axiom rules $\vdash_{\mathcal{T}} \Gamma, A$ for $A \in \mathcal{T}$.

12 Discussion: doubting traditions

§12.1 I presented the big lines of what I call *traditions of logic*. However, some other people speak another "logical dialect": they will say that their tradition of logic is *model theory* (that I did not mention much since it is not my field of expertise) or *computability theory*. Keep in mind that I am deeply influenced by my background of computer scientist.

§12.2 The formal vessel of logic emerged from empirical phenomena such as the *experience of regularities*; the fact that some phenomena we are able to describe by language have a consistent shape which can be freed from contingent information. Aristotle's logic was more concerned about the *shape* of reasoning while Stoic's logic was interested in the *decomposition* of statements. This already shows that there are several ways to formally capture the logical activity. And yet, we tend to take our formalisation of logic for granted. What makes our encapsulation of logic *right*? The explanation of the convenient syntax/semantics separation is that it is correct when the syntactic operations we defined conform to the semantics, *i.e.* the meaning we expect. Gödel's incompleteness shows that mismatches can occur between syntax and semantics[30]. This seems unsatisfying. On the top of that, semantics is basically just another syntax in an upper level, judging another lower-level syntax by a procedure of evaluation. To quote Girard's [Gir07, Section 5.1] cliché (and rather brutal) remark, such semantic explanations sound rather silly when we think about it: $A \wedge B$ is true when the interpretation of A and B is 1 (being true). This shows how semantics mirrors syntax. Then, what justifies semantics apart from our hermetic and unintelligible intuitions? After all, where does our intuitions on logic even come from? With this semantic justification, we are not able to justify what we are doing unless we say that it is how the *laws of nature* are but then we are stuck with a definitive and limited answer about what logic is.

§12.3 **The role of intuition.** Any tradition should be doubted but their systematic rejection in a sort of *tabula rasa* is too extreme. We should ask ourselves if this logical folklore has a *reason to be*, if it can be *justified* at all. It makes no sense to get rid of traditions only because they are traditions (even when they are perpetuated without serious self-questioning). All these developments of logic are not a bunch of nonsense. They led to the developments of various practical tools and ideas now used in computer science. But even without this utilitarian point of view on logic, we *feel* that it is meaningful even if we cannot put words on it. Moreover, foundational issues actually barely affect mathematical practice. It is still impressive how natural deduction and sequent calculus, with their beautiful structures and properties, mainly emerged from rather vague intuitions about logic. I believe that our vague intuitions and our prejudices are still essential after all, even though logic is often seen as cold, exact and void of any feeling. Obviously, it is an error to take our intuition for reality as if logic was the circuits of reality, hard-coded in our brain, which we are somehow fully conscious of. Our vague intuitions, either right or wrong still guide us pretty well providing we learn to tame them.

[30] If completeness is the fact that syntax is correct with respects to semantics, then incompleteness is the fact that it is not; that some parts of syntax exist without conforming to semantics.

$$\frac{A}{A \text{ tonk } B}\ \text{tonk } i \qquad \frac{A \text{ tonk } B}{B}\ \text{tonk } e \qquad \frac{\dfrac{A}{A \text{ tonk } B}\ \text{tonk } i}{B}\ \text{tonk } e$$

Figure 12.1: Natural deduction rules for tonk and a proof containing a redundancy. This proof can infer everything we want from any statement. Is it logical? If not, then why?

§12.4 **The search for justifications.** The culprit is the *evaluative* nature of semantics. To evaluate logical objects means that we already have in mind how they should be and what they should mean. If an external and explanation of logic apparently cannot be found[31], it may be possible to look for an *internal* explanation instead. But how can syntax explains itself? At a first glance, it makes absolutely no sense at all. The question of justification of logic has already been discussed by several philosophers. Regarding natural deduction, Arthur Prior [Ste61] asked why would not it be possible to consider a new connective "tonk" with the introduction rule of $\vee$ and the elimination rule of $\wedge$. We would then be able to prove any statement (*cf.* Figure 12.1). We can conclude, like Prior did, that inference rules by themselves are not able to provide any logical meaning. Nuel Belnap [Bel62] and Michael Dummett [Dum91] thought that not all combinations of rules were *"meaning-constituting"*. The rules has to follow some constraints. Gentzen (although it was mainly studied by Dummett) introduced the term of *"harmony"* to describe the property of those inference rules which are meaning-constituting. The ideal objective would be to formalise Wittgenstein's [Wit10] famous slogan *"meaning is use"* implying that the procedural operations done on logical symbols are sufficient to convey their meaning: we would then reach our internal explanation.

§12.5 We already saw in the presentation of natural deduction that elimination rules intuitively correspond to the use of symbols. Then the possibility of reducing a redundancy might mean that the use of a symbol is *sound*, that it is possible to do something out of it (while preserving hypotheses and conclusion). Harmony is then probably related to a balance between introduction rule (definitions) and elimination rules (use of definitions). This idea gives an *utilitarian* and *computational* meaning to logical symbols. For instance, the meaning of conjunction would be its computational ability to construct pairs and extract its components. But then, our question reduces to "what proof reductions steps are valid?". It is possible to hack the system so that our tonk connective can be reduced (for instance with a reduction similar to the case of $\Rightarrow$ in Figure 9.7). In this thesis I show that a more satisfying answer is possible, providing we work in another logical architecture. If the subjective evaluative aspect of semantics asserting objectivity is faulty, then we shall go beyond evaluation. The key is to consider *mutual interaction* from which meaning will emerge.

§12.6 **Doubts about the unicity and plurality of logic.** In this chapter, I presented

[31] Unless we are able to directly communicate with God.

classical and intuitionistic logic. But when we think about it, is logic not *unique*? When we say "it's logical!" in our everyday life, to what logic do we refer to? Do not we refer to a *unique* logic? I now expose a personal and speculative point of view (which may be completely wrong) I have about the nature of classical and intuitionistic logic:

- *classical logic* can be reduced to a single connective $A \uparrow B := \neg(A \wedge B)$ called Scheffer's stroke. This connective basically express the *incompatibility* of two formulas *w.r.t.* to truth values; they cannot be true at the same time. Negation corresponds to the fact that a formula is incompatible with itself: $\neg A := A \uparrow A$. For the disjunction, it is false only when both inputs are false. This can naturally be expressed as the incompatibility $A \vee B := \neg A \uparrow \neg B$. Similarly, implication is the incompatibility between a true assumption and a false conclusion. Hence, we have $A \Rightarrow B := A \uparrow \neg B$ (as already explained in Paragraph 6.7). As for the fundamental equivalence $A \equiv \neg\neg A$, it is justified by the fact that the incompatibility of $\neg A$ and $\neg A$ is consistent with A. However, classical logic is not a mere calculus of incompatibility between individuals. If it is about incompatibility then it should be a *manichean* incompatibility: the excluded middle $A \vee \neg A$, says that anything is either true or false and it is expressed by the incompatibility $A \uparrow \neg A$ between A and its alter ego $\neg A$. The world is divided into two hermetic categories: whoever is not with me is against me[32]. Girard's alternative definition [Gir18b, Section 1.2] of the excluded middle $(A \equiv B) \vee (B \equiv C) \vee (C \equiv A)$ accentuates this *social segregation*: among 3 propositions, two must be equivalent, you have no other choice. Although I do not investigate it, it is even possible to consider a logical system with Scheffer's stroke [Ten79]. Finally, what is a *classical proof*? Is classical provability simply asserting/resolving (in)compatibilities?
- In *intuitionistic logic*, the implication $A \Rightarrow B$ is primitive and cannot be defined by $\neg A \vee B$ since the translation actually relies on classical axioms. Intuitionism cannot be reduced to incompatibility. Forgetting $A \vee \neg A$ is like forgetting that some statements are incompatible as if we could not be sure about it. Instead, this limitation has the very nice property of forcing the unique conclusion formula to be justified/constructed: the limitation becomes an ability, a requirement for more information. This makes intuitionism about *functionality*: we focus, not on incompatibility but on the *process* or *actions* leading to some result. Intuitionistic logic can then be seen as actually stronger than classical logic in some sense since classical axioms can be encoded (*cf.* Paragraph 9.12). This surprising encoding seems related to the fact that functionality may be able to encode a space of incompatibility, as shown in the natural deduction proof on Figure 9.6 which is completed with two functional operations: *saving* the data $\neg(A \vee \neg A)$ (which is only possible with the wrapping of double negation) then *loading* it afterwards in the rightmost branch.

§12.7 **Doubts about the elementary logical operations.** A reasonable doubt about logic

[32] Which is apparently something shared by both Jesus and antifascist activists.

is whether these natural notions of conjunction, disjunction, negation, implication and quantification constitute the *bedrock of logic*. Or can logical operations can be decomposed to reveal more primitive constructions? Girard's linear logic [Gir87a] shows that the implication $A \Rightarrow B$ can actually be decomposed as $!A \multimap B$ where $!A$ represent a potentially infinite amount of A and $A \multimap B$ is a *linear implication* using its argument exactly once as if it was a limited resource. This offers a point of view of logic as *resources handling*. Moreover, if logic is related to natural language at all, it is fair to ask whether it is possible to represent other aspects of natural language (such as references, indexicals etc) without resorting to ad-hoc constructions (such as temporal modalities but what is ad-hoc, by the way?).

§12.8 **Doubts about predicate calculus.** As already remarked by Girard [Gir18b, Section 1], predicate calculus (*cf.* Section 7) is not a so natural extension of propositional calculus. A class of *individual*, distinct from logical entities, is introduced. They cannot hold a truth value but refer to some entity in a given universe. It is not clear what logically justifies such a distinct category apart from our own point of view (or prejudice in Girardian terms) on what logic should be. Another suspicious point is the exclusion of empty model (*cf.* Paragraph 7.16) which seems ad-hoc. Finally, equality is treated as a mere binary predicate but should not equality be a more fundamental notion?

§12.9 **Doubts about the undoubtable.** Finally, Girard dares to doubt about the most elementary things: the logical identity $A \Rightarrow A$ and the equality $a = a$. Doubting about such things (which were often considered axioms) makes no sense for most logicians and mathematicians. However, Girard suggests the idea that they actually reflect some preconceptions of logic [Gir18b, Section 2].

- For $A \Rightarrow A$, the idea is that it reflects Frege's philosophy of denotation. The two occurrences of A refer to the same entity. However, such paradigm is not sufficient to explain some phenomena occurring in sequent calculus such as the distinction between $\Rightarrow$ and $\vdash$ (*cf.* Paragraph 11.7). We have to take into account the cold aspect of sequent calculus and consider that it speaks about *actions on some locations*. It then appears that the two occurrences of A are two distinct entities at different locations but which are related. This is similar to how printers can copy a sheet of paper. The two sheets of paper are *distinct* but we can verify that one has the same content as the other: identity becomes something one has to prove. This will be more explicit when introducing Girard's proof-nets and transcendental syntax;
- as for $a = a$, in the same fashion, the two occurrences of a are thought to refer to the same entity in a given universe. If we consider second-order equality (*cf.* Paragraph 8.7), in order to prove $a = a$, we would have to prove $X(a) \Leftrightarrow X(a)$ for any X but this is trivial since $X(a) \Rightarrow X(a)$ by identity. However, as Girard pointed out, some properties are implicitly forbidden such as the fact of "being on the right" which indeed distinguishes the two occurrences of a. Ironically, it is as

if the "licit properties" were the ones satisfying our preconceptions about equality, hence making equality trivial.

§12.10 **A criticism of Truth and Reality.** Everything being either true or false is a very natural statement but if we take formal logic as presented in this chapter and forget the metaphysical connexion between truth values (just symbols) and the philosophical concept of truth, then what distinguishes 0 and 1? What makes 1 more *virtuous* than 0? They are distinguished because we choose that some statements are true (typically, axioms). But is truth a matter of choice when I say that $1+1=2$ is true and $1+1\neq 2$ is not? There is an idea[33] that axioms could be *proven* instead of simply *given*. But even with this idea, absoluteness of truth cannot be found. Computer science could probably give answers, for instance by saying that what is true is what can be computationally materialised, *i.e.* what corresponds to some material reality (computation being the ground of reality). If we step back from all our logical formalities, our formal conception of logic presupposes a specific shape of reality. However, some parts of computational reality are not captured by neither classical nor intuitionistic logic (as pointed out by Girard in his commentary on Turing [TGBB95, Chapter 3] and the beginning of his geometry of interaction project [Gir89b]).

- Chemical equations such as $2H_2 + O_2 \mapsto 2H_2O$ *consumes* matter. But these equations abstract from details such as time or the environment so that what is kept is only the relation between concepts. Both classical and intuitionistic logic violates chemical principles as $A \Rightarrow A\wedge A$ (matter can be produced out of nowhere). Should we say that chemical reactions are not logical?
- In the same fashion, banks do a lot of operations which have to keep a certain consistency so that clients do not suddenly obtain or loose money by mistake. However, money is also an entity which can be consumed. Can we say that it has nothing to do with logic?
- Databases are able to store data such as clients registered in a bank. It is possible to update, add or remove data. However, if two data A and B are related in the database, either we have the truth of the statement expressing this relation (classical world) or a constructive proof of it (intuitionistic world). But in both cases, truth and provability are static and eternal properties which never die. What is true, is true forever (mathematical truth). How to treat the removal or update of data? But can we say that databases are simply not logical?

Girard's linear logic shows that it is possible to consider a logic of resources, thus giving a formal logical account to these ideas[34] but his last project of transcendental syntax, presented in this thesis, shows that it is not the last answer yet.

[33] Already there at the time of Leibniz, according to David Rabouin.

[34] Actually, it is an a posteriori justification of linear logic. He discovered linear logic first in another context then tried to justify how interesting it actually was.

Chapter 2

Computational panorama

After all this logical history of conflicts, ideals and failures, I would like to tell a whole new story: the story of *computation*. Like logic, computation is a natural phenomenon which has always "been there". But it is only recently that it took a great importance. Unlike logic, it is a bit (only a bit) more clear what computation is about. Computation in its mature form also came way later. An amusing fact that computer scientists like to tell is that the theory of computation (and computers) existed way before computers themselves. Those computers being *everywhere* nowadays, it shows how of a great importance the study of computation is.

I start with few historical facts about computation then present the main classes of models of computation. Most definitions are inspired from Sipser's introduction to the theory of computation [Sip06]. This chapter sometimes refers to the previous chapter about logical foundations and is thus not completely independent.

13 The experience of procedurality

My main reference for this section is Dowek's book *"Les métamorphoses du calcul"* (in French).

§13.1 We only need time to realise an order in the succession of events. We sometimes are subject to some events. We are sometimes responsible of some events. We are sometimes also responsible of some events to which we are subject, or in other words: we can make some events happen. A typical example are methods described by a list of actions. It can be a cooking recipe or a method to solve a specific problem. Such methods can be shared so that other people can reproduce it, so there is a social aspect to it. It can be more or less precise and sensitive to an environment. For instance, for a cooking recipe, we may give more details about ingredients or leave the choice free. Those methods are called *algorithms*[1].

[1] Term thought to be derived from Al-Khwârizmî's name (a Persian mathematician) but the reality seems more nuanced than that, as stated by McLarty [McL08] (himself quoting Theseus Logic's website): *"The term algorithm was not, apparently, a commonly used mathematical term in America or Europe before Markov, a Russian, introduced it. None of the other investigators, Herbrand and*

§13.2 A common example of an old algorithm[2] is Euclid's algorithm which is the method computing the greatest common divisor of two integers. For instance, the set of common divisors of 36 and 30 is $\{1, 2, 3, 6\}$ and the greatest is 6. Euclid's method is defined by the following steps for two positive integers a and b:

1. if $b = 0$ then the result is a;
2. otherwise, we compute the remainder r of the integer division $a \div b$ and do the step 1 again with $a := b$ and $b := r$.

We apply this algorithm on the inputs $a := 36$ and $b := 30$:

- the remainder of $36 \div 30$ is $r = 6$ since $36 = 30 \times 1 + 6$;
- we have $a := 30$ and $b := 6$. The remainder of $30 \div 6$ is $r = 0$ since $30 = 6 \times 5 + 0$;
- we have $a := 6$ and $b := 0$. Since $b = 0$, the final result is 6.

Algorithms are often related to the practice of mathematics as they provide a way to solve problems. However, notice that algorithms are actually *sensitive* to a context: I do not explain the method to do the integer division $a \div b$. You are free to choose any method you want providing it does *what it should do*. It is similar to not giving too many details about the tools in a cooking recipe, you just have to find something which does the job. In the case of the integer division $\div$, your choice has to follow a particular specification implicitly given by the algorithm which defines what we mean by "integer division".

§13.3 Logically speaking, algorithms provide an alternative way to prove mathematical statements. When having an existential statement $\exists x.P(x)$, I need to find some element x that satisfies the property P. It is possible to prove it only by using logical means without ever knowing what that x looks like. But it is also possible to *construct* that x, called *existential witness*. Those constructions give rise to what we call *constructive proofs*. On the other hand, non-constructive proofs usually occur when using proof by contradiction (or equivalent rules such as the excluded middle). Proof by contradiction says that whenever you need to prove some statement A, you can virtually assume it to be wrong (assume $\neg A$) and show that it leads to a contradiction only to infer at the end that A must be true (and that our initial assumption was impossible). By doing so, we do not construct an actual proof of A but take a sort of *logical detour*. This is connected to the ideas of *intuitionistic logic* (*cf.* Paragraph 4.3): proofs in intuitionistic logic correspond to constructive proofs and algorithms seems to be a way to construct such proofs.

Gödel, Post, Turing or Church used the term. The term however caught on very quickly in the computing community.". Al-Khwârizmî's name indeed produced terms similar to "algorithm" but it initially referred to algebraic/numerical systems.

[2] It has to be noted that although Euclid's algorithm is already a relatively old notion of computation, computational methods were not uncommon. I have heard from a presentation of Gilles Dowek in the Château de Goutelas (Summer 2022) that reasoning was exclusively computational before the Ancient Greeks.

Those ideas of algorithms and constructive proofs were developed independently until the middle of the 20th century.

§13.4 Similarly to logic, the theory of algorithms has been seriously developed rather late; around the 20th century when predicate calculus (*cf.* Section 7) was rising. But unlike logic which was studied since at least since the Ancient Greek, the theory of algorithms is a rather recent subject which took a great importance from its connexions with formal logic. Using predicate calculus, it is possible to represent mathematical statements with logical formulas. In particular, Euclid's algorithm is a way to prove the statement "c is the greatest common divisor of a and b" which can be represented by the atomic formula $GCD(a, b, c)$ or equivalently, by the formula $gcd(a, b) = c$ where GCD is a predicate and gcd is a function giving the greatest common divisor of two inputs. Algorithms become procedural ways to *prove logical statements*. Several other algorithms were developed to prove logical statements for arithmetic (or a fragment of arithmetic). For instance, Presburger developed a method to prove arithmetical statements for integers without multiplication.

§13.5 **Decision problem.** Given the procedural philosophy of *formalists* (*cf.* Paragraph 4.4), it is not surprising that questions between logic and algorithms were explicitly mentioned by Hilbert. Hilbert's *decision problem*[3] asks for a mechanical and *effective* procedure to solve statements of predicate calculus. This method should always terminate with an answer in finite time. If such method exists, then mathematics (thought to be fully expressible in formal logic at this time) could be *mechanised*. Logic and mathematics would then be reduced to *pure computation*. More generally, given a function f taking a formula as input and answering either true (1) or false (0) is there a mechanical way to compute this function? This can also be generalised to any mathematical function. The main problem is that no mature notion of *computation* existed at that time.

§13.6 **Computation.** Alonzo Church and Alan Turing both independently came up (around the 30s) with a formalisation of computation, now called *model of computation*. Their answer was apparently influenced by Gödel's incompleteness theorems Paragraph 5.4.

- Church's proposal, is the λ-calculus [Chu32], a model of computation corresponding to a sort of *realisation* of mathematical functions. It is the ancestor of *functional programming*. It works with a functional language expliciting the relation between input x and output $f(x)$ for a function f;
- Turing's model is the famous Turing machine [Tur36] which formalises the intuitive idea of a mathematician writing on a piece of paper. Symbols can be written on paper depending on what is read on that same paper and the pencil can move either to the left or the right. To correctly model theoretical computation, we assume that we will never run out of paper. This is the ancestor of *imperative programming* where programs are sequences of instructions doing a specific task.

[3] *Entscheidungsproblem* in German.

Both models have been shown to be equivalent for functions over rational numbers. A function which can be computed in finite time by an expression of λ-calculus or a Turing machine is said to be *computable*. A formula A is *decidable* when its associated function returning its truth value $f(A) \in \{0, 1\}$ is decidable. A lot of models of computation have been considered but the most powerful ones (Herbrand-Gödel's equations and Kleene's recursive functions) were all shown to be equivalent to Church and Turing's proposal. This defines a space of *computable functions*. Models equivalent to Turing machines are called *Turing-complete*.

§13.7 **Church's thesis.** Until now, no model of computation has been shown to be stronger than Turing-complete models in the sense that there would be a problem solved by some model but not by a Turing-complete one. This is *Church's thesis* (a term introduced by Kleene who was one of Church's students), also called *Church-Turing thesis*. Two notions are identified: Turing-complete models and the vague and intuitive notion of *computation*. This thesis (which is not a theorem) asserts that Turing-complete defines *the* notion of computation. Two versions of Church's thesis can actually be distinguished:

- the *physical variant* asserts that Turing-completeness captures the intuitive of computation by a physical system, hence it captures the concept of *natural* computation;
- the *psychological variant* asserts that Turing-completeness captures what is computable by the human mind, hence it captures the concept of *mental* computation.

The two variants are not necessarily equivalent depending on whether Nature computes more than humans or the opposite. This is the subject of a debate between Longo and Dowek known as the *Longo-Dowek controversy* explained in Yannis Hausberg's PhD thesis (not completed yet as I am writing this thesis). Other interesting related references are Longo and Paul's papers [LP11, PL09].

§13.8 **Partial recursive functions.** Still in the 30s, Kleene constructed (from works of Gödel and Herbrand) a mathematical theory of computable functions (which can be implemented by "effective procedures" such as λ-calculus and Turing machines). These functions, called *partial recursive functions* or (μ-recursive functions), are constructed from the following primitive functions:

- constant functions $\texttt{Const}_n^k(x_1, ..., x_k) = n$;
- successor function $\texttt{Succ}(x) = x + 1$;
- projection functions $\Pi_i^k(x_1, ..., x_k) = x_i$

and closed under the following operations:

- n-ary function composition $f \circ (g_1, ..., g_n) = h$ where:

$$h(x_1, ..., x_k) = f(g_1(x_1, ..., x_k), ..., g_n(x_1, ..., x_k))$$

- primitive recursion defined on functions *base* and *ind* by:

$$\texttt{Rec}(base, ind)(0, x_1, ..., x_k) = base(x_1, ..., x_k);$$

$$\texttt{Rec}(base, ind)(\texttt{Succ}(x), x_1, ..., x_k) = ind(x, \texttt{Rec}(base, ind)(x, x_1, ..., x_k), x_1, ..., x_k);$$

- minimisation operator μ defined on a function f by:

$$\mu(f)(x_1, ..., x_k) = a \text{ such that } f(a, x_1, ..., x_k) = 0$$

$$\text{and } f(i, x_1, ..., x_n) > 0 \text{ otherwise.}$$

The minimisation operator searches for the smallest natural number for which the input function f returns 0 on it. In case there is no such argument, it is not defined. The implementation of minimisation corresponds to a procedure which searches with a counter (initialised to 0) on the first argument of f and which loops by incrementing its counter until finding the right a cancelling the function. Functions without minimisation are called *primitive recursive functions* and are only implemented by terminating procedures. This corresponds to a weaker form of potentially looping procedures.

§13.9 **Undecidability.** Unfortunately, the answer to Hilbert quest for mechanised mathematics is *negative*. It is impossible to always provide an answer for any logical statement of predicate calculus. This can also be formalised by the fact that we cannot provide a constructive proof of $\forall X.X \vee \neg X$ (corresponding to the excluded middle) because we would need to either construct a proof of X or a proof of $\neg X$, which is impossible without any information about X. But this can be decidable for some specific choice of X. In terms of Turing machine, it can be shown that some problems are impossible to solve by a finite succession of steps. In particular, the famous *Halting problem* asks for a way to tell whether or not a given Turing machine will halt (only by looking at how it is constructed and without simulating it). It is possible to show (and we do in a dedicated section of this chapter) that assuming the existence of a Turing machine deciding this problem leads to a contradiction. Undecidability can be seen as a consequence of Cantor's *diagonal method*. Since Gödel's incompleteness (*cf.* Paragraph 5.4) is also related to this method, undecidability can be seen as a *computational version* of incompleteness: it expresses the gap between a program/machine and its result, the former cannot be reduced to the latter (since an answer may not come at all depending on inputs).

14 Realisation of machines

§14.1 The story of machines is a parallel story. There are methods that we can share and use but executing them can require an important amount of time or work. Tools has always been made all around the world to reduce the workload of people and increase the efficiency of work for various activities (including very early forms of programmable

machines for music [Koe01]). But what about purely *mental* operations? Can we automatically reproduce some operations occurring in our mind such as arithmetic calculation?

§14.2 **Pascal's calculator.** In the Western world, it is considered that Blaise Pascal, a French scientist, was the first to design a mechanical calculator called the *Pascaline* [Pas97]. This machine was required by his father who was a tax commissioner. Important calculations can take a lot of time and such a machine would be indeed very useful, especially in scientific professions. Pascal's calculator had wheel to write the digit of input numbers. His machine could do basic arithmetic operations such as addition and subtraction with a carry mechanism. Unfortunately, it seems that because of social relations with craftsmen and other factors of his time, Pascal could not expand his project and only 20 machines were constructor over 10 years [Mou88].

§14.3 **Calculus Ratiocinator.** Several different machines were designed after Pascal such as Gottfried Leibniz or Thomas de Colmar's calculators. But let us focus on Leibniz's idea. In his "De arte combinatoria" (1666), he imagined a machine (or algorithm) which could determine whether a sentence, expressed in an idealised language called *characterisca universalis*, was true or false. Such a machine, called *calculus ratiocinator*, would be able to answer everything, any philosophical or scientific question. A famous quote of Louis Couturat illustrate this idea:

> "*Pour résoudre une question ou terminer une controverse, les adversaires n'auront qu'à prendre la plume, en s'adjoignant au besoin un ami comme arbitre, et à dire : « Calculons ! »*"
>
> – *Louis Couturat*

meaning that two persons discussing would only need to *compute* in order to conclude the conversation. A common caricature is that Leibniz's idea corresponds more or less to Hilbert's ideal of mechanised mathematics (*cf.* Paragraph 4.4). However, it seems that Leibniz idea was more nuanced and that he later considered himself that those ideas were too naive and probably impossible to realise (see David Rabouin's works).

§14.4 **Babbage's engine.** To tell the story briefly, Charles Babbage, a British inventor, remarked that some minor errors were present in calculus tables. These human errors are usually caused by unawareness. Inspired by Pascal's engine and his own knowledge of logarithm table, he presented in 1821, the *difference engine* for the purpose of computing polynomials. However, for mechanical reasons, it has never been completed. Another machine, the *analytical engine* is even closer to today's computer. It is designed for more general purposes, and would include arithmetic operations, conditional branching, memory and loops, making it Turing-complete a century before Turing. But again, Babbage was ahead of his time's technology and his machine could not be completed

[4]. His machine would be theoretically be fed with "punched cards" with holes encoding instructions to execute. This is indeed a form of program.

§14.5 **Lovelace's implication.** His engine has been developed jointly with Ada Lovelace, with whom he was corresponding. Ada Lovelace was a countess who received a mathematical education and who was able to interact with great scientists of her time. She called her approach "poetical science", probably in reference to her mathematician mother and her poet father. Lovelace was very interested in Babbage's analytical engine and it is thought that she had a great vision of the potential of these machines [FF03]. Babbage's himself was fascinated by her mathematical skills. She was especially known for her contribution to the analytical engine with a program computing Bernoulli numbers. Today, she is often considered as the *first* programmer[5].

15 Computation as functional process

My personal reference for this chapter is Hammache's PhD thesis (in French) [Ham21, Chapter 4]. A lot of historical information of this section are more or less translated summaries of parts of Hammache's thesis.

Now, I suggest to dive into the λ-calculus, one of the most important models of computation. The λ-calculus comes from logical considerations and from philosophical questioning about the notion of function.

The notion of function

§15.1 A very natural way to approach computation is to start from *(mathematical) functions*. Functions are often defined as associations of two values (input and output). A first intuition of computation is then to give a concrete realisation which simulates this association. But since we are still in mathematics and not in concrete physical realisations, it is not clear what is a satisfying set of steps mechanically simulating a function.

§15.2 Frege, in his ideal of meaning, developed the notion of function together with some other authors such as Schönfinkel. Although functions were already widely used by mathematicians, definitions of function were neither satisfying nor accurate for Frege. A first characteristic of functions is to distinguish their *argument* (input) and *result* (output) but the point was to understand what could be an argument and what could be a result. Frege wanted to enrich logic with a more mature idea of function, probably because it

[4] It even inspired sci-fi novels such as William Gibson and Bruce Sterling's "The Difference Engine" which imagines a world in which Babbage was able to realise and popularise his inventions.

[5] Although there are actually historical contestations showing that Babbage was actually the first programmer (see Doron Swade's affirmations) and controversies about how much Lovelace played a role in "programs" for analytical engine. I cannot tell who is wrong or right but this is a rather convenient myth today.

would materialise his personal philosophy. In Frege's philosophy, logic was about relations between "pure" concepts. In his mind, a *concept* was a function returning a *truth value* and a *relation* was a binary function returning a truth value. He distinguished a hierarchy of concepts so that a concept of first level is a function from a class of *"objects"* to a truth value.

§15.3 The common notation for functions was $f(x) = y$, expressing the *product* or *result* y of a function f when applied to an input x. We can then plug a value as input and obtain a result: $f(a) = \{x := a\}y$ where $\{x := a\}y$ is a replacement of x by a in y. However, as remarked by Frege, it only expresses the result of the function, hence what the function *does*: we have a black box producing y from x. We have no mean to express what the function *is*, independently of any *use* of it and of any argument. Frege wanted to find the *essence* of functions, and for that reason, he introduced a notation called *course-of-values*. The function f defined by $f(x) = x + 2$ can be represented by the distinguished expression $\dot{\alpha}(\alpha + 2)$. The application of the function $\dot{\alpha}(f(\alpha))$ (representing a function f) to an argument c is seen as an intersection written $c \cap \dot{\alpha}(f(\alpha)) = f(c)$. We now have two expressions, one to refer to the result produced by a function and another one referring to the function itself.

§15.4 Schönfinkel had the ambition of reducing logic to the mechanisms of functions and eventually establishing a "calculus of functions" [Sch24]. It was already remarked in logic that most connectives were superfluous and could be defined by a small set of connectives (*cf.* Paragraph 6.10). For instance, Whitehead and Russell wanted a reduction to negation and conjunction, and Frege to negation and implication. However, Scheffer[6] showed that it was possible to consider a unique connective $A \mid B := \neg(A \wedge B)$ (sometimes written $A \uparrow B$) which would define all the others. Schönfinkel wanted to extend this idea to predicate calculus. It was already known that every predicate calculus formulas could be turned into a formula $\forall x_1 ... \forall x_n . A$ for a quantifier-free formula A. All Schönfinkel needed to do was to get rid of universal quantification. A natural solution was to extend Sheffer's stroke with a variable in order to define the formula $f(x) \mid^x g(x) := \forall x . f(x) \mid g(x)$. But the variable x is not even so essential. We can forget it since $A \mid^x B$ expressed the incompatibility between two predicates (written with a notation of function), independently of the variable x itself. Hence, he introduced the functional notation $Ufg := f(x) \mid^x g(x)$ and was led to introduce other such functions like $Ix = x$ or $Sfgx = fx(gx)$. These expressions are now known as *"combinators"* and they later have been developed by Haskell Curry around 1930.

§15.5 Remark that in Schönfinkel's expression Ufg, we have an occurrence of function taking another function as argument. It is also technically possible to consider functions returning other functions. Nowadays, it is known as the idea that functions of several arguments can be represented by unary functions only[7]. The idea is that a function

[6] Although it was already present in Peirce's works [Ham21, Section 4.3.1]. Peirce again!

[7] This feature is usually called *curryfication* but it seems that there are reasons to call it *fregeing* since the idea was originally introduced by Frege [AK12, Chapter 3].

$f : (x, y) \mapsto a$ of two arguments can be represented as a unary function $f' : x \mapsto (y \mapsto a)$ returning another unary function $g : y \mapsto a$ as a result.

Functional foundations for logic

§15.6 In 1932, Alonzo Church [Chu32] was interested in alternative foundations for logic which would be more satisfying than Russell's type theory and Zermelo's axiomatic theory. Instead of set theory, he was interested in functions as a basis for logical foundations[8]. Church had a representation of functions which could both represent the function itself and their application to arguments. The function $f(x) = x + 2$ was written $\lambda x.x + 2$ and its application to the argument 2 (corresponding to $f(2) = 4$) was written $(\lambda x.x + 2)\ 2$ which could be reduced to $\{x := 2\}(x + 2)$. Such an expression is called λ-*abstraction*. Everything can be considered a function providing we have a mature notion of function. Using λ-abstractions, the *mechanisms* linking the input and the output of a function are explicited by reducible syntactic expressions taking into account the notion of *function application* within the system itself [9].

§15.7 Church then designed a logical system where all logical constructions are unary functions. The grammar of Church's formulas is the following:

$$M, N ::= x \mid \lambda x.M \mid M\ N \mid \Pi \mid \Sigma \mid \& \mid\sim\mid \iota \mid A$$

where x is a variable and both A and ι are special operators we will not talk about. The application $M\ N$ will sometimes be written MN We leave parentheses implicit and application is considered left-associative, *i.e.* $M_1M_2M_3 = (M_1M_2)M_3$. We also write $\lambda x.\lambda y.M$ or even $\lambda xy.M$ for $\lambda x.(\lambda y.M)$. Remark that predicates are replaced by functions (this is consistent with Frege's idea of concepts/predicates as functions). The symbols Π, Σ are respectively the universal and existential quantification and the symbols $\&$ and $\sim$ are respectively the conjunction and negation. A very important construction in the function application MN which corresponds to $M(N)$ in mathematics. Remark the unusual presentation of connectives. This is due to the fact that connectives are primitive unary functions. For instance, the formula $A \wedge B$ would be written $(\&A)B$ representing a function $\&$ applied to a first argument A and returning a function to which we feed a second argument B. The main reduction rule called β-reduction is the following:

$$(\lambda x.M)N \rightsquigarrow_\beta \{x := M\}N$$

For instance, we have $\vee := \lambda x.\lambda y.\ \sim((\&(\sim x))(\sim y))$ which corresponds to the disjunction and $(\vee A)B \rightsquigarrow_\beta \sim((\&(\sim A))(\sim B))$ for some formulas A and B.

[8] The use of a calculus of functions to solve foundational problems was apparently not considered by Schönfinkel.

[9] In modern programming we sometimes talk about *nameless functions* and see Church language as programming with mathematical functions

§15.8 Around the same period, Haskell Curry was also interested in a functional point of view on the foundations of logic. He introduced *combinatory logic* which can be seen as a further development of Schönfinkel's calculus of functions. Curry wanted to develop what he called an *Urlogik*, a sort of *proto-logic* which would precede any logical construction. Another interest of Curry was to study the notion of *substitution* which had not been clearly formalised although widely used, sometimes implicitly (by Frege and Russell in particular). Combinators are constants representing functions without explicit bound variables (unlike the x attached to λ and $\forall$), and are associated with a reduction rule. Curry considered the following combinators:

- $Bxyz \rightsquigarrow x(yz)$;
- $Cxyz \rightsquigarrow (xz)y$;
- $Kxy \rightsquigarrow x$;
- $Wxy \rightsquigarrow xyy$.

It is possible to retrieve Schönfinkel's combinators with $I := SKK = WK$ and $S := B(B(BW)C)(BB)$. In particular, S was thought to be counter-intuitive and Curry provided a decomposition with more intuitive combinators. It appears that combinatory logic is equivalent to λ-calculus:

- not so surprisingly, combinators can be easily represented with λ-terms;
- more surprisingly, any computable function can be represented with only the combinators S, K and I.

Chris Barker showed (around 2000) that it was possible to consider only one combinator ι [Bar01] defined by $\iota := \lambda f.((fS)K)$ such that the SKI combinators could be defined with $I := \iota\iota$, $K := \iota(\iota(\iota\iota))$ and $S := \iota(\iota(\iota(\iota\iota)))$.

§15.9 **Paradoxes.** It has quickly been remarked that Church's system was inconsistent and that a paradox similar to Russell's one is present Section 3. It is possible to construct the expression $\lambda f. \sim(ff)$ which will be sometimes true or false depending on its argument (Church considered his language a logic and hence any statement could be either true or false). But even weirder, it is possible to feed it with... itself and obtain $R := (\lambda f. \sim (ff))(\lambda f. \sim(ff))$. Now we obtain the following reductions:

$$R \quad \rightsquigarrow_\beta \quad \sim((\lambda f. \sim(ff))(\lambda f. \sim(ff))) = \sim R$$

$$\sim R \quad \rightsquigarrow_\beta \quad \sim\sim R \equiv R$$

The last reduction relies on the equivalence $\neg\neg A \equiv A$ that Church allowed. The main problem of Church's logic is not the presence of this paradox itself but the requirement that formulas must be either true or false. The expressions already have a *computational* meaning independent of any logical meaning. Unlike Church, Curry did not necessarily want to get rid of paradoxes at any cost. His Urlogik was more permissive and he thought

$$\begin{aligned}
&\texttt{disj}\ (\texttt{is_zero}\ \overline{1})\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&= \texttt{cond}\ (\texttt{is_zero}\ \overline{1})\ \texttt{true}\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&= (\lambda c.\lambda x.\lambda y.cxy)\ (\texttt{is_zero}\ \overline{1})\ \texttt{true}\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&\rightsquigarrow_\beta (\{c := \texttt{is_zero}\ \overline{1}\}(\lambda x.\lambda y.cxy))\ \texttt{true}\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&= (\lambda x.\lambda y.(\texttt{is_zero}\ \overline{1})xy)\ \texttt{true}\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&\rightsquigarrow^*_\beta (\texttt{is_zero}\ \overline{1})\ \texttt{true}\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&\rightsquigarrow^*_\beta \texttt{false}\ \texttt{true}\ (\texttt{neg}\ (\texttt{is_zero}\ \overline{1}))\\
&\rightsquigarrow^*_\beta \texttt{neg}\ (\texttt{is_zero}\ \overline{1})\\
&\rightsquigarrow^*_\beta \texttt{neg}\ \texttt{false}\\
&= (\lambda x.\texttt{cond}\ x\ \texttt{false}\ \texttt{true})\ \texttt{false}\\
&\rightsquigarrow_\beta \{x := \texttt{false}\}(\texttt{cond}\ x\ \texttt{false}\ \texttt{true})\\
&= \texttt{cond}\ \texttt{false}\ \texttt{false}\ \texttt{true}\\
&\rightsquigarrow^*_\beta \texttt{false}\ \texttt{false}\ \texttt{true}\\
&\rightsquigarrow^*_\beta \texttt{true}
\end{aligned}$$

Figure 15.1: Example of computation of the truth value of the statement "either 1 is null or it is not"

that it could be interesting to study systems containing paradoxes and the meaning of those paradoxes.

Mathematical functions as a model of computation

§15.10 **Lambda-calculus.** Forgetting the functional point of view of logic, we can consider Church's system as a model of computation now known as (pure or untyped) λ-*calculus*. The expressions are called λ-terms and are defined by the following grammar:

$$M, N ::= \quad x \mid \lambda x.M \mid MN \qquad \text{(Lambda-terms)}$$

And the only reduction rule, as expected, is still the β-reduction[10]:

$$(\lambda x.M)N \rightsquigarrow_\beta \{x := M\}N$$

[10] Although other rules are often considered, this is the most important rule.

We say that two terms M and N are β-equivalent, written $M \equiv_\beta N$ when there is a sequence of terms $M_0 = M, M_1, ..., M_n, M_{n+1} = N$ such that either $M_i \rightsquigarrow_\beta M_{i+1}$ or $M_{i+1} \rightsquigarrow_\beta M_i$. We write $M \rightsquigarrow_\beta^* N$ when M can be reduced in zero or at least one step to N. When we have $M \rightsquigarrow_\beta N$ and N is irreducible then we say that N is a *normal form* of M. This model of computation is very powerful since it has been shown that it is expressive enough to expressive all the computable functions. It is also powerful for its simplicity: it only needs three constructors and a single reduction rule. Here are some common example of terms:

- the identity function $\lambda x.x$;
- the left and right projection $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$;
- $\lambda x.xx$;
- $(\lambda x.xx)(\lambda x.xx)$ which reduces into itself.

It is also possible to encode boolean operators or natural numbers so that the reductions behaves as expected:

- the truth value 0 is represented by $\texttt{true} := \lambda x.\lambda y.x$;
- the truth value f is represented by $\texttt{false} := \lambda x.\lambda y.y$;
- conditionals are represented by $\texttt{cond} := \lambda c.\lambda x.\lambda y.cxy$ (c will be a truth value seen as a selector selecting either x or y);
- negation is represented by $\texttt{neg} := \lambda x.\texttt{cond}\ x\ \texttt{false}\ \texttt{true}$;
- conjunction is represented by $\texttt{conj} := \lambda x.\lambda y.\texttt{cond}\ x\ y\ \texttt{false}$;
- disjunction is represented by $\texttt{disj} := \lambda x.\lambda y.\texttt{cond}\ x\ \texttt{true}\ y$;
- the natural number n is represented by $\overline{n} := \lambda s.\lambda z.s^n z$ where s^n corresponds to n successive applications of s (representing the successor function over the variable z representing zero);
- the function $\texttt{is\_zero} := \lambda n.n\ (\lambda x.\texttt{false})\ \texttt{true}$ which returns true if its input is 0 and false otherwise.

A simple example of computation with λ-terms is $(\lambda x.\lambda y.y)\ (\lambda x.x)\ (\lambda x.x) \rightsquigarrow_\beta (\{x := \lambda x.x\}(\lambda y.y))\ (\lambda x.x) = (\lambda y.y)\ (\lambda x.x) \rightsquigarrow \{y := \lambda x.x\}y = \lambda x.x$. A more complex example of detailed computation is given in Figure 15.1.

§15.11 **Conflicts of names.** There is a small technical and often neglected (and also often painful to treat) detail to take into account with λ-terms. In the expression $\lambda x.M$, the variable x is *bound* in M, meaning that all occurrences of x appearing in M are virtually linked to λx. This is similar to the mathematical function $f(x) = M$. When an argument is provided, all those occurrences of x are replaced. But in λ-calculus, it is technically possible to use several occurrences of a variable which are unrelated. For instance, in $\lambda x.M(\lambda x.N)$, the first x is bound in M only and the second one in N only.

Moreover, names are not so relevant since $\lambda x.x$ is equivalent to $\lambda y.y$. There are two classical solutions to avoid conflicts of variables:

- to use a rule called α-*conversion* which allows to rename bound variables. This rule (which defines an equivalence relation called α-equivalence) is defined by:

  $$\lambda x.M \equiv_\alpha \lambda y.\{x := y\}M$$

  for any variable y. We would then have $\lambda x.M(\lambda x.N) \equiv_\alpha \lambda x.M(\lambda y.\{x := y\}N)$ and $\lambda x.x \equiv_\alpha \lambda y.y$;

- another solution suggested by De Bruijn[11] is to use *relative indexes*. Instead of variable names and λx, we would have natural numbers and a nameless λ. Natural numbers, called *De Bruijn indexes*, refer to the distance (to the left) of the λ symbol they are linked to. For instance, $\lambda x.M(\lambda x.N)$ would be rewritten $\lambda M'(\lambda N')$. Any 1 in N' is linked to the second λ and any 2 is linked to the first λ. The left projection $\lambda x.\lambda y.x$ would simply be written $\lambda\lambda 2$. Note that when we say that these indexes refers to a distance, we are not speaking about *literal distance* in terms of characters but in term of *scope*. For instance, in the term $\lambda(\lambda 1(\lambda 1))(\lambda 21)$, the index 2 actually refers to the first λ.

§15.12 It is possible to consider recursion in λ-calculus (which can be seen as a way to loop in a program by calling itself) by using fixpoint operators. A fixpoint of a function f in mathematics is a value c such that $f(c) = c$. Fixpoint operators of λ-calculus are λ-terms `fix` such that `fix` F reduces to F. A common fixpoint operator is the Y operator defined by:

$$Y := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

which can be used to define the *factorial function* $f(x) = x! = \prod_{i=2}^{x} i$. The template for any recursive function is to define a function $\lambda f.\lambda n.M$ where f will correspond to the recursive call (which is a necessary trick since we cannot refer to the function being defined), n is the argument of the function and M is the body of the function. For the factorial function, we have

$$F := \lambda f.\lambda n.\texttt{cond}\ (\texttt{is\_zero}\ n)\ \overline{1}\ (f\ (n-1)).$$

We assume the existence of a subtraction operator on $\mathbf{N}$ such that $0 - 1 = 0$. We define `fact` $:= Y\ F$ and we have the following reduction:

$$\texttt{fact} = Y\ F \rightsquigarrow_\beta (\lambda x.F(xx))(\lambda x.F(xx))$$

$$\rightsquigarrow_\beta F((\lambda x.F(xx))(\lambda x.F(xx)))) = F\ (Y\ F) = F\ \texttt{fact}$$

[11] There are a lot of confusions about how to pronounce this name. I suggest that you listen to the pronunciation at least one.

The notion of (simple) type

§15.13 Type theories have been introduced in logic mainly for the purpose of avoiding paradoxes making logical systems contradictory. Russell and Whitehead in particular contributed to the developments of these theories (*cf.* Paragraph 3.4). Since Church's system was originally a logical system, type theory can be applied to it. According to Hammache [Ham21, Section 9.1.1], Church wanted to keep the computational power of his system so he did not introduce typed λ-calculus for the only purpose of avoiding paradoxes. He especially wanted to constrain the space of λ-terms to a *relevant subset* such as the subset of normalisable terms. Church's type theory is a theory of *control* of computation.

§15.14 Church's original type theory [Chu40] has the following grammar of types:

$$\alpha, \beta ::= \qquad \iota \mid o \mid \alpha\beta \qquad \text{(Types)}$$

where ι is the type of individuals, o is the type of propositions and $\alpha\beta$ is the type of functions from the type β to the type α (a function type applied to an argument type). Types are directly integrated within λ-terms with the following grammar:

$$M_\alpha, N_\alpha ::= \qquad x \mid \lambda x_\beta.M_\alpha \mid (MN)_\alpha \qquad \text{(Terms)}$$

where $\lambda x_\beta.M_\alpha$ is of type $\alpha\beta$, and in $(MN)_\alpha$, M is of type $\alpha\beta$ and N of type β. The reduction now takes types into account:

$$(\lambda x_\beta.M_\alpha)_{\alpha\beta}N_\alpha \rightsquigarrow_\beta \{x := N\}M_\beta$$

Notice that types have a functional notations, exactly like λ-terms. It suggest the idea that types themselves can behave like functions as well. This elementary type system, now known as *simple types* is sufficient to exclude unwanted terms. For instance, the term $\lambda x.xx$ cannot be typed:

- it has to be of type $\alpha\beta$ because it is a function. We have the partial typing $\lambda x_\beta.(xx)_\alpha$;
- having $(xx)_\alpha$ implies that we must have the left x of type $\alpha\alpha'$ and the right x of type α';
- however, a same variable x cannot be both of type α' and $\alpha\alpha'$, this is contradictory.

This term was not so innocent anyway, since it can form $(\lambda x.xx)(\lambda x.xx)$ which is known to normalise into itself and hence corresponding to an infinite loop.

§15.15 **Application in linguistics.** This is rather anecdotal in this thesis but I just would like to mention that Church's original type theory has applications in linguistics. Montague grammar for instance, makes use of λ-calculus and higher order predicate calculus to express meaning in natural language. A verb phrase (VP) such as "The man is walking"

$$\frac{}{\Gamma, x : A \vdash x : A}\ \text{var} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ \text{abs} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ \text{app}$$

Figure 15.2: Typing rules for simply typed λ-calculus.

$$\dfrac{\dfrac{\dfrac{}{x : \alpha, y : \beta \vdash x : \alpha}\ \text{var}}{x : \alpha \vdash \lambda y.x : \beta \to \alpha}\ \text{abs}}{\vdash \lambda x.\lambda y.x : \alpha \to \beta \to \alpha}\ \text{abs}$$

Figure 15.3: Typing derivation for a λ-term.

for instance is of type $\iota \to o$, it takes a subject (the man) and turns it into a proposition. It shows that types can also speak about the behaviour of computational entities.

§15.16 Curry considers untyped λ-terms as independent from types. We write $M : A$ for a type M of type A. Types are written A, B and atomic type variables α, β are considered instead of ι and o. We have the following grammar for types:

$$A, B ::= \qquad \alpha, \beta, ... \mid A \to B \qquad \text{(Terms)}$$

To still represent the fact that typed λ-terms cannot exist without types, the construction of terms have to be constructed by some rules. These rules use the notation of sequents of Paragraph 9.9. A sequent is an expression asserting the type of an expression. The sequent $\Gamma \vdash M : A$ says that the term M is of type A for a context or typing environment Γ made of type assertions $x_1 : A_1, ..., x_n : A_n$. The typing rules are given in Figure 15.2. Using typing rules, it is possible to construct trees exactly like we do in proof theory. The typing derivation of Figure 15.3 guarantees that $\lambda x.\lambda y.x$ can be attached to the type $\alpha \to \beta \to \alpha$.

§15.17 Types can be seen as constraints over computation so to prevent some behaviours from happening, but they also express a computational behaviour. A type derivation is a very important object since it acts like a certificate for a computational behaviour. The existence of such a certificate ensures that our computational object will have a specific behaviour and not another unexpected one. After Church, a lot of type systems have been developed in order to capture other behaviours. For instance, *intersection types* are able to give several types to terms so that the x in $\lambda x.xx$ (which terminates) can be typed with the intersection $\langle A \to B, A \rangle$ because it acts both as a function and as an argument. It is also possible to extend λ-calculus with other constructions such as pairs $(M, N) : A \times B$ with $M : A$ and $N : B$.

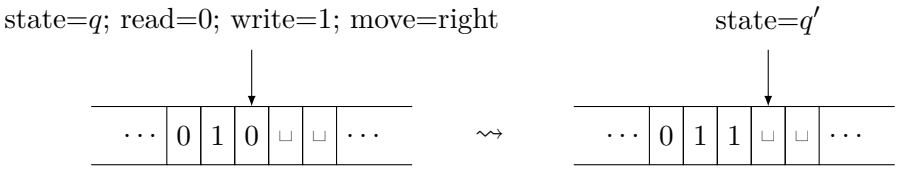


Figure 16.1: Transition in a tape for a Turing machine.

16 Computation as state machine

My references for this section are Petzold's book *"The annotated Turing"*[12] (which is a commentary on Turing's seminal paper) and Sipser's introduction to the theory of computation [Sip06] for formal definitions.

The mathematician-machine

§16.1 Alan Turing is without doubt one of the most famous figures of computer science. During his youth, it seems that Turing was fascinated by machines and more especially with their relation to the human mind [Pet08, Chapter 4]. Already interested in science as a young student, Russell's work on mathematical logic caught his interest very early. Hilbert's decision problem came to him through a course of Maxwell Herman Alexander Newman that he took in 1935. At this time, the idea of *mechanical process* did not refer to machines. This is after this course that he began to work on Hilbert's problem. He tried to design a machine which would reproduce the steps done by a mathematician. His machine could compute numbers in binary form (only with 0 and 1) but also well-known mathematical constants such as π and e. More than that, he also designed a *universal machine* able to simulate the operations of another machine, exactly like how a program (such as a compiler) would read another program in today's programming. Turing was an independent researcher in the beginning[13], which is probably part of why he came up with this unusual solution. Church already published about this problem at this time but his works were very connected to the literature, unlike Turing. This interests led to a collaboration with Church himself who became his PhD advisor.

§16.2 To quote Turing himself, his machines [Tur36] (now called *Turing machines*) consist of a "tape" (representing a piece of paper) divided into sections called "squares". Each square can bear a symbol, and the machine can read the square of a tape and move to the left or to the right. Turing says that a square read by the machine is a *scanned square* which is the only square the machine is aware of. The design was motivated by the *use* of machines (*e.g.* computing numbers). A machine can be in a specific state

[12] One of the first books about computer science I have read!

[13] According to Petzold's book, he even reinvented the binomial theory and developed his own notation for calculus.

Configuration		Behaviour		
State	Symbol read	Symbol written	Movement	New state
q	0	1	right	q'
q'	0	1	left	q
q'	1	0	right	q''
...	...	...	...	...

Figure 16.2: Transition table for a Turing machine.

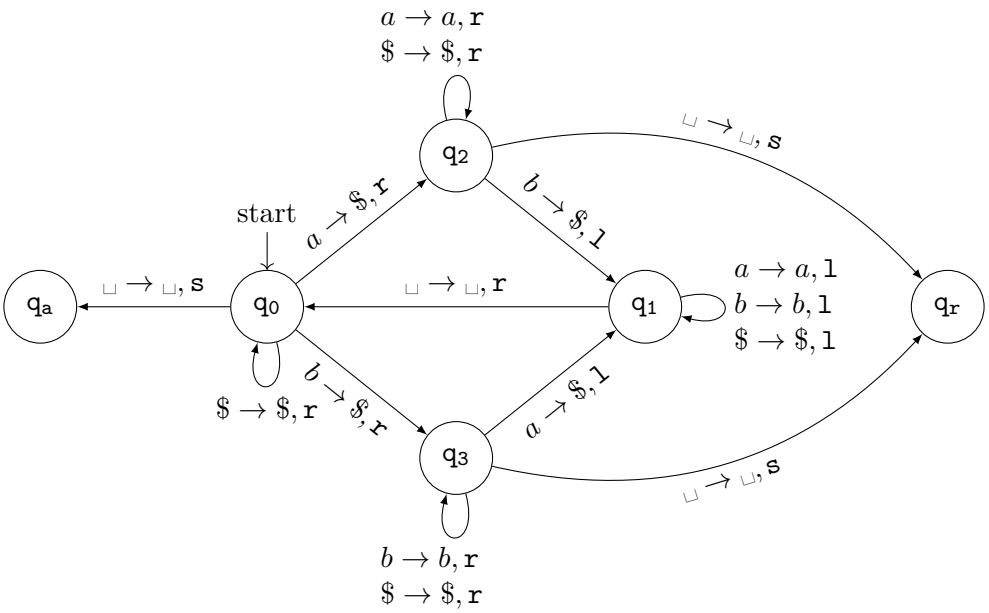


Figure 16.3: State graph of a Turing machine recognising the language of words with as many a as b.

called *m-configuration* (or simply *state*) which is a classification of situations the machine can be in. An intuitive example of states are the two or three colours of traffic lights. The machine goes from an m-configuration to another one depending on what symbol is read. Then, a symbol is written and the machine moves left or right. An example of move is given in Figure 16.1. The machines move accordingly to a *transition table* such as the table of Figure 16.2.

§16.3 Turing wanted to compute numbers and he had in mind *sequences* of digits such as π or the decimals of $1 \div 3$. Interestingly, he wanted his machines to compute forever and distinguished two classes of machines: *circular* machines which get stuck and the others are *circle-free*. Although very powerful, Turing did not provide a lot of relevant examples illustrating the computational power of his machines.

§16.4 We now give a modern definition of *non-deterministic Turing machine* (NTM) with more

modern definitions and notations. Modern Turing machines can be either *deterministic* or *non-deterministic*. Deterministic ones have only one possible transition at every step, so no choices are possible whereas non-deterministic machines have several choices of transition. Turing machines can be used for two purposes (which differ from Turing's original machines):

- *accepting or rejecting* an input word, and then the machine is seen as a *recogniser* of a language;
- *computing a mathematical function* by stopping with a result left on the tape (which is closer to what Turing had in mind). In the case of partial functions, the machine is expected to loop on undefined inputs.

Our definition will include special states q_a and q_r, respectively representing the state of acceptation and the state of rejection. Moreover, unlike Turing, we will be especially interested in machine computing mathematical functions in finite time instead of printing a sequence of digits. Looping will usually be seen as an *ill* behaviour. We refer to Appendix A.3 for technical definitions of language theory used in this section.

§16.5 A non-deterministic Turing machine is a tuple $M = (Q, \Gamma, \Delta, q_0, q_a, q_r)$ where Q is the set of states, Γ is the alphabet of the tape (*cf.* Appendix A.3), $\Delta : Q \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon \times \{\mathtt{l}, \mathtt{r}\})$ is the transition function (which can be written as a transition table as in Figure 16.2), q_0 is the initial state, $\Gamma_\varepsilon := \Gamma \cup \{\varepsilon\}$ and finally, q_a and $q_r \neq q_a$ are respectively the state of acceptation and rejection. We require that $\Delta(q_a, c) = \emptyset$ and $\Delta(q_r, c) = \emptyset$ for all $c \in \Gamma_\varepsilon$. We assume the existence of a symbol $\sqcup \in \Gamma$ representing the emptiness. An example of *state graph* of a Turing machine is illustrated in Figure 16.3. Vertices are states and arrows represent transitions.

§16.6 A configuration[14] is a triple (l, q, r) where $q \in Q$ and l, r are tapes (sequence of characters in Γ). Configurations represent the position of the head of the machine on the tape and the associated state. We say that a configuration C leads to C' when moving the head in C accordingly to Δ leads to C'. A word $w = c_1 ... c_n$ is *accepted* by M, written $M(w) = 1$, when there is an *accepting* sequence of configurations $C_1, ..., C_n$ such that:

1. $C_1 = (\sqcup, q_0, w)$;
2. C_i leads to C_{i+1};
3. $C_n = (l, q_a, r)$ for some l and r.

If the last configuration has a state q_r instead, we say that M *rejects* w, which is written $M(w) = 0$. When M *loops* infinitely on w by never reaching a final state for any sequence of configuration, we write $M(w) = \infty$. We require that an NTM necessarily ends on q_a or q_r when it stops. For instance, the Turing machine of Figure 16.3 accepts *aabb* but rejects *aba*. The machine is designed so to accept all words having as many a as b.

[14] Note that this is different from what Turing called configuration (a state).

4	.	.	.	.	.	.	.
3	9	.	.	.	.	.	.
2	5	8	.	.	.	.	.
1	2	4	7	.	.	.	.
0	0	1	3	6	.	.	.
	0	1	2	3	4	5	6

Figure 16.4: Encoding of pair of natural numbers with natural numbers.

§16.7 A lot of extensions can be designed such as considering several heads, several tapes, tapes infinite and only one side, etc. But most interesting variants have been shown to be equivalent, which sounds very surprising. It shows how *robust* the class of computable functions is. This emphasises the fact that Turing machines capture the notion of effective procedure and is the right choice to define the class of computable functions. Surprisingly, Turing machines are able to simulate Church's λ-calculus and the converse holds as well!

§16.8 **Automata theory.** Turing machines can be weakened to produce what we call *automata* [Sip06, Part One]. The weakest form of commonly studied automata are *finite automata* which linearly consume their input and have no memory. Finite automata recognise a subset of recursive languages called *regular languages*. They can be extended to *pushdown automata* having a stack as memory (a sequence of stacked elements on which we can either extract the top element or add new elements over it) and recognising *context-free languages*. Considering several types of machines allows to consider a whole hierarchy of classes of languages called *Chomsky hierarchy*.

A bit of hacking with Turing machines

§16.9 **Coding.** For someone not used to Turing machines (or even programming), it looks very cumbersome to solve problems. How can Turing machines solve problems about logical formulas or mathematical expressions? The idea is to *encode* inputs in a representation easy for a Turing machine to work with. Actually, even a binary language only consisting of the symbols 0 or 1 is sufficient. More generally, most objects of interest can be turned into natural numbers. There are known bijections between the set of natural numbers $\mathbf{N}$ and the set of pairs of natural numbers $\mathbf{N} \times \mathbf{N} = \mathbf{N}^2$. This allows to represent a pair (n, m) as a single natural number $\#(n, m) \in \mathbf{N}$. The idea is that we can construct an array with $\mathbf{N}$ as vertical and horizontal axis (*cf.* Figure 16.4). By enumerating natural numbers on the diagonal, it is possible to associate a *unique* natural numbers with any pair $(n, m) \in \mathbf{N}^2$. By convention, the first component corresponds to the horizontal axis and the second one corresponds to the vertical one. If we focus on $\#(x, 0)$, we remark that we must enumerate all natural numbers from 0 to x. At the first diagonal, we have enumerated 1 number (only 0), then $1 + 2$ at the second diagonal and so on. Hence

$$\begin{array}{llllllll} 0 & . & 1 & 2 & 9 & 2 & 7 & ... \\ 0 & . & 9 & 5 & 1 & 2 & 7 & ... \\ 0 & . & 8 & 2 & 2 & 2 & 9 & ... \\ 0 & . & 1 & 8 & 0 & 8 & 1 & ... \\ 0 & . & 1 & 0 & 2 & 5 & 3 & ... \\ ... & & & & & & & \end{array}$$

Figure 16.5: Attempt at enumerating all real numbers in the interval $[0,1] \subseteq \mathbf{R}$.

$\#(x,0) = 1+2+...+x = \frac{x(x+1)}{2}$. Now, what happens when we go y steps up in the vertical axis? What we do is basically doing y steps on the right in order to select the right diagonal to traverse, then increase by y to represent the upward traversal of the diagonal. We obtain $\#(x,y) = \#(x+y,0)+y = \frac{(x+y)(x+y+1)}{2} + y$. The idea can be generalised to any tuples with the recursive definition $\#(n_1,...,n_k) := \#(...\#(\#(n_1,n_2),n_3),...,n_k)$ by iterating the encoding of pairs. Since these are bijections, a machine can *decode* inputs in a unique way.

§16.10 **Universality and simulation.** If most objects of interest can be encoded as numbers, so can be Turing machine themselves. Sets can be encoded as tuples and since a machine M is a tuple, it can be encoded itself as a number $\#M$. It is then perfectly possible to consider machines taking other machines as input or even outputting a machine as result. In particular, it is possible to design what we call a *universal Turing machine* which is a Turing machine able to read the encoding of a machine as input (or its *code* in modern terms[15]) and simulate it by reproducing exactly the steps it would have done. More formally, if we have a universal machine U, then $U(\#M, w) = M(w)$.

§16.11 Even more than objects and machines, it is possible to turn problems into set of numbers containing their solution. For instance, if $\#w$ is the encoding of a word into a natural number, then the problem of telling if two words are equal becomes the set $E := \{\#(\#w_1, \#w_2) \mid w_1 = w_n\} \subseteq \mathbf{N}$. Decidable problems become *recursive sets*, *i.e.* sets A for which the (total) characteristic function χ_A such that $\chi_A(x) = 1$ when $x \in A$ and $\chi_A(x) = 0$ otherwise can be implemented by a terminating Turing machine. For instance, it is not difficult to design a Turing machine recognising the language E. It would be a machine accepting two equal input words. A weaker class of sets are *recursively enumerable sets* corresponding to sets which can be enumerated by a Turing machine. A way to formalise this is that recursively enumerable sets are sets E for which there exists a partial function f of domain E which can be implemented by a Turing machine.

§16.12 **Cantor's diagonalisation method.** By using simulation and encoding, it is possible to do some little hacks on Turing machines. Using Cantor's diagonalisation method it is possible to infer the inexistence of some machines. Originally, Cantor's diagonalisation

[15] This is not so impressive in programming: it is what compilers and interpreters do.

is a way to prove that $\mathbf{R}$ is uncountable, *i.e.* that there is no bijection between $\mathbf{N}$ and $\mathbf{R}$. It is sufficient to consider the interval $[0,1] \subseteq \mathbf{R}$. The idea is to try to enumerate all these real numbers as in Figure 16.5, then show that it is always possible to construct a real number not in this enumeration (although it is initially thought to be complete). This problematic natural number x is constructed as follows: makes the first decimal different from the first decimal of the sequence (for instance 2 is different from 1, do the same for the second decimal (for instance 2 is different from 9. At this point we have 0.22. We can do the same for every decimal: the n-th decimal of x should be different from the n-th decimal of the n-th number of the sequence. We indeed obtain a real number theoretically not in the enumeration.

§16.13 The set of Turing machines is countable (hence bijective to $\mathbf{N}$) because they can be encoded as natural numbers and the set of all languages is uncountable, because in particular, the set of all infinite sequences of digits in uncountable (by Cantor's previous proof that $\mathbf{R}$ is not bijective to $\mathbf{N}$). Since the set of all languages cannot be put into correspondence with corresponding Turing machines recognising them, there must be some languages which cannot be recognised by Turing machines. This is a first interesting result which can be interpreted as the fact that some problems cannot be *computationally solved* (you can find more details in Sipser's introduction to the theory of computation [Sip06]). Now, it is interesting to exhibit such problems.

Undecidable problems

§16.14 **Halting problem.** Now, I introduce the so-called halting problem[16], showing that some problems are impossible to solve with a Turing machine. The halting problem asserts that "there is a machine M such that for any machine N, $M(\#N) = 1$ when N halts and $M(\#N) = 0$ when it does not". We can define a weird machine D (for *diagonal*, in reference to Cantor's diagonalisation method) taking the encoding of a machine M as input:

$$D(\#M) = \begin{cases} \infty & \text{when } M(\#M) = 1 \\ 1 & \text{otherwise} \end{cases}$$

where we implicitly make use of a machine checking if a machine loops on an input. This machines D loops for every machine accepting itself. The input of D ranges over the set of *all* machines. In particular, D itself is a machine of this set (it is similar to what happens in Russell's paradox). What happens when we would like to know the forbidden answer of $D(\#D)$?

$$D(\#D) = \begin{cases} \infty & \text{when } D(\#D) = 1 \\ 1 & \text{otherwise} \end{cases}$$

Assume that $D(\#D) = 1$ then by the definition of D... $D(\#D)$ loops. Now, if $D(\#D)$ loops, by the definition of D, $D(\#D) = 1$. This is contradictory. Hence, it is impossible

[16] Term apparently introduced by Davis [Dav58].

to tell if a machine will loop on an input (which was the only assumption). A variant of this problem called *acceptation problem* has exactly the same structure except that we replace halting is replaced by rejection of the input. You can find more details in Sipser's book [Sip06, Section 4.2] about how diagonalisation plays a role in this construction.

§16.15 **Post's correspondence problem.** Undecidability can also occur in very natural problems independent of Turing machines such as in Emil Post's correspondence problem (PCP) [Pos46]. Dominos are pair words $\frac{w_1}{w_2}$. Given a list (ordered sequences) of dominos $L = \frac{v_1}{w_1}, ..., \frac{v_n}{w_n}$, the top words and bottom words form new words $v_1...v_n$ and $w_1...w_n$ by concatenation. If $v_1...v_n = w_1...w_n$ then we say that L is a *match*. Post's problem asks whether a given set of dominos can produce a list (with possible repetitions of dominos coming from the set) which is a match. For instance, Sipser's [Sip06, Section 5.2] example of match is $\{\frac{b}{ca}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c}\}$ which can produce the list $\frac{a}{ab}, \frac{b}{ca}, \frac{ca}{a}, \frac{a}{ab}, \frac{abc}{c}$. However, $\{\frac{abc}{ab}, \frac{ca}{a}, \frac{acc}{ba}\}$ cannot produce a match. This problem is known to be undecidable.

§16.16 **Busy Beaver.** A funny undecidable problems introduced by Radò [Rad62] is formulated as a game. Given a number of state n and the alphabet $\{0, 1\}$, you have to design a terminating Turing machine which outputs as many 1 symbols as possible. The winning machine with the largest input is called BB-n. The problem of verifying if an input machine M is a BB-n for some n is undecidable.

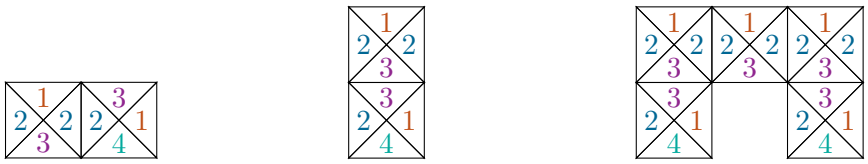
Towards programming

§16.17 **Post machines.** I recommend Liesbeth De Mol's works for more details about Post's contributions. Although it is the name of Turing that is often mentioned as the "inventor of computer science", in the same year as Turing (1936), Emil Post actually designed a Turing-complete model of computation independently [Pos36]. His model of computation works with a two-way infinite sequence of spaces and boxes which be either marked or unmarked. A "worker" operates on one box at a time accordingly to a given ordered finite sequence of instructions[17]. The instructions which can be used are the following:

- marking the current box;
- erasing the mark in the current box;
- moving the current box to the right;
- moving the current box to the left.

Since the notion of recursiveness (which characterises the computable) already existed, Emil Post already conjectured that his model of computation was equivalent to it, which would make his model Turing-complete. It is also possible to think of several extensions such as using several distinct ways to mark boxes or extending the primitive instructions.

[17] He almost invented Amazon's logistics before computer science even existed.



(a) Horizontal connexion. (b) Vertical connexion. (c) More complex tiling.

Figure 17.1: Tiling with two wang tiles.

§16.18 **Wang machines.** In 1954, Hao Wang, a philosopher and mathematician introduced[18] B-machines which are equivalent to Turing machines and close to Post machines [Wan57]. His paper also contained extensions such as the W-machine allowing an erasure instruction. His B-machines have a binary infinite tape and the following set of instructions:

- $\rightarrow$: move right;
- $\leftarrow$: move left;
- $*$: print $*$ (a mark) on the current square
- C_n: if the current square is marked then jump to the instruction n;
- E: erase the mark of the current square (only for W-machines).

This is often considered as the first presentation of Turing machines which is closer to today's programming. A *program* is an ordered sequence of pairs (k, I) where $k \in \mathbf{N}$ is a line number and I is one of instruction described above. For instance, $\{(1, *), (2, \rightarrow), (3, C_2), (4, \rightarrow), (5, \leftarrow)\}$ is a program.

§16.19 Today, various sort of programming languages exist to automate operations on a computer. A computer has several type of memories such as volatile memory (which is used for temporary operations) and non-volatile memory (mostly used for permanent storage). Programming languages allow to write programs executing tasks which manipulate the memory or other features of computers. These languages can be *imperative* like the sequence instructions of Post and Wang machines and they can be *functional* be extending the λ-calculus by the realisation of a mathematical function. They can also be both or coming from completely different traditions of computation. Programming opened a whole new fascinating field of interesting way to automate operations in very different ways.

17 Computation as tiling construction

§17.1 After his Wang machines, Wang introduced a new model of computation: Wang tiles. It is a puzzle-like model which is very different from the previous models presented. It works by placing some pieces next to each other following some conditions in order to obtain a more complex construction. It is what happens with dominos. Dominos are objects having two sides which both contain a number between 1 and 6 (included). Dominos are put on a surface with two sides next to each other only when they hold exactly the same number. Wang tiles generalise this idea with a square of four sides (north, south, east, west) containing a colour (or equivalently a natural number). These tiles are then put on the plane $\mathbf{Z}^2$ (or equivalently a grid graph, *cf.* Appendix C). From a given set of tiles T, we can construct *tilings* with occurrences of tiles in T. An illustration in Figure 17.1 presents different connexions of Wang tiles.

§17.2 **Wang tiles.** More formally, a *tile type* can be formalised by a tuple $t_i = (s^i_{\mathtt{w}}, s^i_{\mathtt{e}}, s^i_{\mathtt{s}}, s^i_{\mathtt{n}})$ for i in a finite set of indexes I. The components correspond to the four sides of the tile. A function $\mathtt{col} \in \mathbf{N}$ associates a natural number of a side. For instance, the two tile types used in Figure 17.1 are $(s^1_{\mathtt{w}}, s^1_{\mathtt{e}}, s^1_{\mathtt{s}}, s^1_{\mathtt{n}})$ with $\mathtt{col}(s^1_{\mathtt{w}}) = 2, \mathtt{col}(s^1_{\mathtt{e}}) = 2, \mathtt{col}(s^1_{\mathtt{n}}) = 1, \mathtt{col}(s^1_{\mathtt{s}}) = 3$ and $(s^2_{\mathtt{w}}, s^2_{\mathtt{e}}, s^2_{\mathtt{s}}, s^2_{\mathtt{n}})$ with $\mathtt{col}(s^2_{\mathtt{w}}) = 2, \mathtt{col}(s^2_{\mathtt{e}}) = 1, \mathtt{col}(s^2_{\mathtt{n}}) = 3, \mathtt{col}(s^2_{\mathtt{s}}) = 4$. Instead of the plane $\mathbf{Z}^2$, we consider subgraph of *square grid graphs* with corresponds to connected subgraphs of the cartesian product $G \times G'$ of two linear graphs (*cf.* Appendix C). Vertices are associated with a coordinate $(x, y) \in \mathbf{Z}^2$. Given a set of tile types T, a *tiling* is a subgraph of square grid graph $G_\alpha = (V, E, \mathtt{end})$ together with a (possibly partial) function $\alpha : V \to T$ associating a tile type $\alpha(v)$ to vertices of G_α. For each tiling, we require that two vertices v and v' respectively of coordinates (x, y) and (x', y') and of tiles $\alpha(v) = (s^i_{\mathtt{w}}, s^i_{\mathtt{e}}, s^i_{\mathtt{s}}, s^i_{\mathtt{n}})$ and $\alpha(v') = (s^j_{\mathtt{w}}, s^j_{\mathtt{e}}, s^j_{\mathtt{s}}, s^j_{\mathtt{n}})$ for $i \neq j$ are adjacent when:

- if v is on the left of v' ($x = x' - 1$ and $y = y'$), then $\mathtt{col}(s^i_{\mathtt{e}}) = \mathtt{col}(s^j_{\mathtt{w}})$;
- if v is on the right of v' ($x = x' + 1$ and $y = y'$), then $\mathtt{col}(s^i_{\mathtt{w}}) = \mathtt{col}(s^j_{\mathtt{e}})$;
- if v is above v' ($x = x'$ and $y = y' + 1$), then $\mathtt{col}(s^i_{\mathtt{s}}) = \mathtt{col}(s^j_{\mathtt{n}})$;
- if v is below v' ($x = x'$ and $y = y' + 1$), then $\mathtt{col}(s^i_{\mathtt{s}}) = \mathtt{col}(s^j_{\mathtt{n}})$.

§17.3 **Domino problem.** Wang's domino problem asks for a way to determine if a given set of tile types is able to produce infinite tilings able to fill all the plane $\mathbf{Z}^2$. In 1966, Robert Berger, one of Wang's student, gave a *negative* answer [Ber66]. He defined a translation from Turing machines to Wang tiles so that if the machine loops then the corresponding set of tile types tiles the plane $\mathbf{Z}^2$. Since the Halting problem is known to be undecidable (*cf.* Paragraph 16.14), it must be impossible to tell if a tile set is able to tile the plane.

[18] The idea has been presented to the ACM in 1954 although the corresponding paper has been published in 1957.

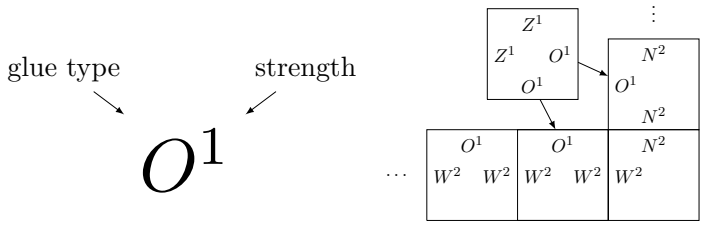


Figure 17.2: Illustration of an assembly in an aTAM. Assume we are at temperature $\tau = 2$. We can connect a new tile to an assembly because the glue types match and the sum of strengths involved is $1 + 1 \geq \tau$.

§17.4 Another notion of interest for people working on Wang tiles is *aperiodicity*. A periodic tiling with a repeated pattern. Soon after Wang introduced his tiles, he conjectured that if a finite set of tiles can tile the plane, then there must be a periodic tiling as well. This is what led to his Domino problem and to Berger's negative answer. But another observation of Berger shows that there must be a finite set of tiles which can tile the plane *aperiodically* (hence without repeated pattern), which sounds very surprising. A quest of some researchers was to found such a finite set of tiles and actually several have been found (some being smaller than others) [CI96, JR15]. It has been shown in 2015 that a set of 11 tiles was the smallest possible.

§17.5 There are known links between tile systems, logic and automata [Tho91]. One link often presented is that set of Wang tiles induce *transducers* [CI96, JR15]. A transducers is a sequential state machine consuming an input word and writing symbols in an output. From the transducer corresponding to a set of Wang tiles it is then possible to reason about properties of (a)periodicity of tilings.

§17.6 Wang tiles are indeed a rather unusual way to compute and yet Berger implicitly showed that it was Turing-complete. Actually, such tile-based computation, although not so natural, found applications in biology where asynchronous communication between independent biological agents is a common thing. It seems that crystal formation, for instance, follows mechanisms of tilings for complex structures. In the remaining parts of this section, we will focus on *DNA computing* [See82]. In DNA computing, DNA strands can be used to compute in different ways with models we will present. DNA computing, although almost inexistent in my field of study, seems underestimated. Biological operations in general can be very long to compute (we have to wait for the end of a biological/chemical reaction but it has the advantage of being massively parallel) and rather cheap.

§17.7 **Abstract Tile Assembly Model.** A first model of DNA computing is the abstract tile assembly model (aTAM) [Win98, Pat14] which is an extension of Wang tiles. We refer to Lathrop et al. [LLS09] for more details[19]. Tiles $t_i = (s^i_{\mathtt{w}}, s^i_{\mathtt{e}}, s^i_{\mathtt{s}}, s^i_{\mathtt{n}})$ are defined in

[19]However, we use a variant without seed assembly σ because it is more natural in our case.

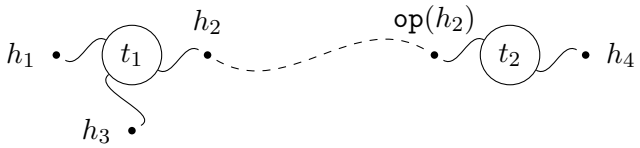


Figure 17.3: Tiling of flexible tiles connected by two complementary flexible arms.

the same way as for Wang tiles. Colours are called *glue types* and are usually represented by constants instead of natural numbers. Given a set of glue types G, we have a function $\mathtt{str} : G \to \mathbf{N}^+$ providing the *strength* of a glue type.

Tiles live in an environment having a *temperature* which is a natural number $\tau \in \mathbf{N}^*$. The idea is that the temperature is a threshold of *cooperation*: a connexion involving several tiles happen only if the sum of strengths of the sides involved in the connexion is at least τ.

A tile assembly system (TAS) is a pair $\mathcal{T} = (T, \tau)$ where T is a set of tile types and $\tau \in \mathbf{N}^*$ is the temperature of $\mathcal{T}$.

Given a set of tile types T, a T-configuration is a partial function $\alpha : \mathbf{Z}^2 \to T$ pasting tiles to the plane. It is associated with a connected square grid graph $G_\alpha = (\mathtt{dom}(\alpha), E)$ with an edge between two vertices representing tiles t_i, t_j with $i \neq j$ when $g_d^i = g_{d'}^j$ for $d = \mathtt{op}(d')$ where $\mathtt{op}$ is the involution defined by $\mathtt{op}(\mathtt{e}) = \mathtt{w}$ and $\mathtt{op}(\mathtt{n}) = \mathtt{s}$.

We say that α is τ-stable if it is impossible to cut E^{G_α} into two parts such that it breaks bonds of total strength at least τ. In other words, it means that a new tile can be added to a T-configuration only if the total strength value of its bonding is at least τ.

A T-assembly for τ is a T-configuration which is τ-stable. Given a TAS $\mathcal{T} = (T, \tau)$, we write $\mathcal{A}_\square[\mathcal{T}]$ for the set of all T-assemblies for τ which are connected and maximal (impossible to extend with more tiles of a given set of tile types). An example of tiling is given in Figure 17.2.

In particular, the case of $\tau = 1$ is called *non-cooperative* since any compatible tiles can be connected independently of others without regards to any global constraints.

§17.8 **Flexible tiles.** Another (less known) model of computation of interest is the model of *flexible tiles* introduced by Nataša Jonoska [JM05] which is able to encode "planar" models (also known as *rigid* tile systems[20]) such as Wang tiles [JM06]. This model is used for what we call DNA computing with *branched junction molecules* [EMJP19, Section 2]. The idea is to consider tiles with *flexible arms* with no constraint with planarity, similarly to cables which can be plugged with each other when complementary. An example of flexible tiling is illustrated in Figure 17.3.

[20] I once asked Jonoska why are we even doing DNA computation with rigid tiles since it intuitively seems more natural to use flexible tiles. Her answer was that it was experimentally easier to have control over rigid tile systems than over flexible tile systems.

Again, the terminology changes. Glue types are called sticky-ends and sticky-end types identify sticky-ends (which correspond to sort of detached part of a DNA strand) having an identical sequence of nucleotides.

We define a Watson-Crick complementarity as an involution[21] $\mathtt{op} : H \to H$. We assume that $\mathtt{op}(h) \neq h \neq \mathtt{op}(\mathtt{op}(h))$. Two sticky-end types h and h' can be connected when $\mathtt{op}(h) = h'$.

Given a set of sticky-end types H and a Watson-Crick complementarity for H, we define a *port bonding system* (PBS) by a pair $(H, \mathtt{op})$.

§17.9 Jonoska, McColm and Staninska's choice [JMS11, Section 2.2] is to model flexible tiles with star-like graphs but I suggest an alternative (and more limited) definition that I personally prefer. A *tile type* is a multiset of sticky-end types and a *pot type* is a multiset of tile types.

For a given PBS $(H, \mathtt{op})$, the *dependency multigraph* $\mathfrak{D}[P]$ of a pot type P defined with H is a multigraph (V, E, ℓ) where the set of vertices is $V := P$ and there is an edge e between two tile types for every pair (p, p') of complementary sticky-end types inside them *w.r.t.* the involution $\mathtt{op}$ so that $\ell(e) = (p, p')$ is an edge-labelling function.

A *complex* (tiling) is a label-preserving graph homomorphism $\alpha : G_\alpha \to \mathfrak{D}[P]$ from a non-empty finite connected multigraph G_α. The vertices v of G_α are *(flexible) tiles* of type $\alpha(v)$.

§17.10 The properties of interest for a given pot type P are the following:

- a complex α is *stable* if G_α has no complementary pair of free (unconnected) sticky-ends in P (which correspond to occurrences of sticky-end types of G_α not involved in any edge label);
- a complex α is *complete* if G_α has no free sticky-ends at all.

A use of flexible tiles is to provide sort of generators of graphs. It is then interesting to study which class of graphs can be generated by which set of flexible tiles.

18 Computation as flow of information

§18.1 It is possible to compute with what we call *circuits* which are directed acyclic graphs (DAG) in which information flows from (usually binary) some inputs to a unique input. In 1937, Claude Shannon have shown in his master thesis [Sha38] that Boolean algebra (which is a more mathematical formulation of Boole's original logic presented in Paragraph 2.2) could be used to simplify electronic machines of his time. This led to the construction of *digital circuits*. Since the History of circuits is a bit unclear and mixed with engineering and technologic developments, I will focus on technical and modern

[21] A function f such that $f(f(x)) = x$.

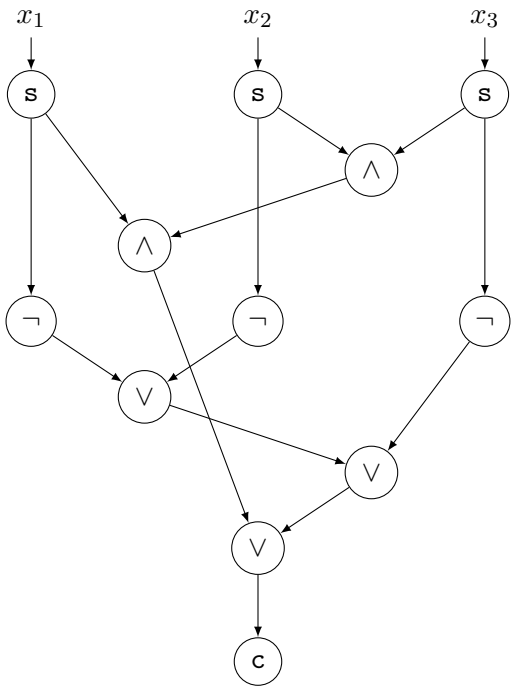


Figure 18.1: Boolean circuit computing $(x_1 \wedge x_2) \wedge x_3 \Rightarrow x_1 \wedge (x_2 \wedge x_3) \equiv \neg((x_1 \wedge x_2) \wedge x_3) \vee (x_1 \wedge (x_2 \wedge x_3)) \equiv ((\neg x_1 \vee \neg x_2) \vee \neg x_3) \vee (x_1 \wedge (x_2 \wedge x_3))$.

mathematical definitions of circuits. Formal definitions can be found in general purpose books mentioning complexity theory[22] such as Sipser's book [Sip06][Section 9.3] or Arora-Barak's book [AB09, Chapter 6]. I suggest a variant of these definitions here.

§18.2 We start with *boolean circuits*. The general idea is that boolean circuits are acyclic directed graphs showing how to produce a single output from a certain number of inputs (either constant or variables) by using logical operations called *gates* (disjunction, conjunction, negation or even more depending on the definitions). We provide a definition using the notion of *sharing* which allows to share the value of an input.

§18.3 A *boolean circuit* is a tuple $C = (V, E, \mathtt{in}, \mathtt{out}, \ell)$ where $(V, E, \mathtt{in}, \mathtt{out})$ is a directed acyclic graph and the function $\ell : V \to \{x_i, 0, 1, \wedge, \vee, \neg, \mathtt{s}, \mathtt{c}\}$ is a labelling function where x_i for $i \in \mathbf{N}$ is a variable, $\mathtt{s}$ corresponds to a *sharing* operation and $\mathtt{c}$ to a conclusion. The *value* $\mathtt{val}_\Omega(v)$ associated with a vertex $v \in V$ is defined inductively as follows by using the usual definitions of boolean functions and an interpretation of variables $\Omega : \{x_i\}_{i\in\mathbf{N}} \to \{0, 1\}$:

- ▷ if $\ell(v) \in \{x_i\}_{i\in\mathbf{N}}$, $\mathtt{in}(v) = \emptyset$ and $|\mathtt{out}(v)| = 1$, we have $\mathtt{val}_\Omega(v) := \Omega(v)$;
- ▷ if $\ell(v) \in \{0, 1\}$, $\mathtt{in}(v) = \emptyset$ and $|\mathtt{out}(v)| = 1$, we have $\mathtt{val}_\Omega(v) := \ell(v)$;

[22] Circuits are subject of a great interest in complexity theory.

- ▷ if $\ell(v) = \mathtt{s}$, $\mathtt{in}(v) = \{v'\}$ and $|\mathtt{out}(v)| = 2$, we have $\mathtt{val}_\Omega(v) := \mathtt{val}_\Omega(v')$;
- ▷ if $\ell(v) = \neg$, $\mathtt{in}(v) = \{v'\}$ and $|\mathtt{out}(v)| = 1$, we have $\mathtt{val}_\Omega(v) := |\mathtt{val}_\Omega(v') - 1|$;
- ▷ if $\ell(v) = \wedge$, $\mathtt{in}(v) = \{v_1, v_2\}$ and $|\mathtt{out}(v)| = 1$,
  we have $\mathtt{val}_\Omega(v) := \mathtt{min}(\mathtt{val}_\Omega(v_1), \mathtt{val}_\Omega(v_2))$;
- ▷ if $\ell(v) = \vee$, $\mathtt{in}(v) = \{v_1, v_2\}$ and $|\mathtt{out}(v)| = 1$,
  we have $\mathtt{val}_\Omega(v) := \mathtt{max}(\mathtt{val}_\Omega(v_1), \mathtt{val}_\Omega(v_2))$;
- ▷ if $\ell(v) = \mathtt{c}$, $\mathtt{in}(v) = \{v'\}$ and $|\mathtt{out}(v)| = 0$, we have $\mathtt{val}_\Omega(v) := \mathtt{val}_\Omega(v')$

with the global condition that there is a unique $v \in V$ such that $\ell(v) = \mathtt{c}$. If a vertex does not follow the above conditions, then the circuit is not well-formed. The *evaluation* of a circuit is defined by $\mathtt{val}_\Omega(C) := \mathtt{val}_\Omega(v)$ such that $v \in V$ and $\ell(v) = \mathtt{c}$. Such a vertex v represents the unique conclusion of the circuit.

§18.4 Figure 18.1 illustrates an example of circuit which is always true whatever the inputs we choose since the associated formula is a tautology (*cf.* Paragraph 6.11). Notice how the sharing gate allows to duplicate the value of an input. This makes boolean circuits different from the truth interpretation of formulas (*cf.* Definition 6.5) since the values of all occurrences of a variable has to be computed. Moreover, sharing is not limited to variables and hence allows for efficient representations of propositional formulas.

§18.5 **Circuit families.** In terms of computability, boolean circuits implement *total* boolean functions $\{0,1\}^n \to \{0,1\}$ for n inputs. However, Turing machines can compute for *any* input size. The only way to compute any input size with circuits is to consider *circuit families* which are set of circuits $(C_i)_{i \in \mathbf{N}}$ for C_k a circuit of k inputs. Such families are actually problematic: they are theoretically able to decide any language including the language associated with the halting problem (Turing machines implement *partial* functions). It looks miraculous but there is actually no way to give a concrete construction of such sets in the first place, hence this is an inaccessible miracle. The usual compromise is to consider *uniform* circuit families which are circuit families which can effectively and efficiently (several definitions can be considered) be constructed by a Turing machine.

§18.6 **Arithmetic circuits.** Boolean circuits can be generalised to *arithmetic circuits* by considering any field K (instead of $\{0,1\}$) and operations on it such as addition or multiplication. This allows to compute finite *polynomials* $\sum_{k=0}^{n} a_k x^k$ for some constants $a_k \in K$ (instead of boolean expressions). Circuits of any kind always have a shape of graph and only the mechanisms underlying gate changes. In particular, nothing prevents us to have more sophisticated functions or mechanisms for gates.

19 Discussion: a single materialisation of computation?

§19.1 Models of computation discussed here are "standard" ones. We are interested in classical, deterministic and finitely described models of computation. We thus choose to exclude models implementing quantum, probabilistic or analogue computation.

§19.2 Let us give a look at all these models from a more synthetic point of view. Some questions of interest to me are the following:

- what do these models have in common?
- cannot we have a single canonical representation of computation which would subsume all the models presented?

Plurality of representations is obviously of great importance since they materialise different practices coming from different computational cultures. However, I believe that these models can still be reunited and that there is something fundamental in how a lot of models of computation work.

§19.3 It seems to me that the models presented in this chapter are all about propagation of information propagated in a structure, thus directly generalising circuits:

- Turing machines but also automata can be represented as state graphs. A run on an input can be seen as propagating a configuration (state of memory and current character read) which is altered at each transition;
- tile systems can be replaced by dependency graphs D and the flow traversing it is a graph-shaped exploration of D representing a tiling construction (Paragraph 17.8). The data propagated and altered corresponds to information about the location of tiles in a particular geometry of space. For instance, in the case of planar tiling in $\mathbf{Z}^2$, tiles are associated with 2D coordinates;
- it is more subtle for functional computation such as λ-calculus but works on the geometry of interaction [Gir89a, DR95] show that λ-terms can be represented by proof-structures [Dan90, Chapter 11] (which are graphs) and reduction of terms can be represented by an exploration of proof-structures [AL95, DR99].

This is an idea shared by Thomas Seiller who represents models of computation by a *space of configurations* and primitive operations as monoid actions on that space [Sei20b, Section A]. It is then possible to define a universal space of computation on which features of models of computation (which define their identity) can be described as constraints over that space. For instance, if automata are seen as tile sets of transitions, choosing the right shape of tile can enforce sequentiality of runs (seen as linear tiling constructions). This shows how sequential computation can be understood as a constraint over a parallel and asynchronous model (tile systems).

§19.4 In order to properly define what would be a "universal" language of computation, fundamental questions about computation have to be answered first:

- what it is that we compute?
- what is a program?
- what is computation?
- what is an algorithm?

In modern terminology, a *program* is a computational entity which can be executed to produce a result. This process of execution is known as *computation* and an *algorithm* is simply a method which can be implemented by a program. However, it makes sense to search for more general and precise definitions.

§19.5 My (probably vague for some people) answer is that what is computed are *answers to questions*. When we compute we usually have a goal in mind (why would we compute otherwise?) and we are looking for an adequacy between the expression of that goal and the answer provided. A *program* corresponds to a construction in the space of answers (without context when it is not put against some goal). This answer can be constructed in an awkward manner, in such a way that it is not clear to us. But it can be explicited by a procedure[23]. This happens when we design a method to construct the answer. If the question is "$4435241325 \times 975746544 =?$" (which is the kind of question a child would ask), the answer 4327671394674730800 does not come to mind instinctively. We usually construct a way to answer the question, which is indeed a sort of program (the method for multiplication we learn at school). Then we use the method to show that it leads to the answer: this is the *execution* (or *computation*) of our program. But the answer was already located in the composite object made of a program plugged to some inputs. It was just implicit. If execution is aborted without completion then it means that I was not able to explain how to get the answer. Lack of sound execution means lack of explanation. If the question is the expression of a goal then we can say that it is an *algorithm*. Algorithms are way to express what we would like to compute, they are sort of *specifications*. They are deeply related to logic which can be understood as a tool to formulate questions or in other words, constructions in a space of questions.

§19.6 These ideas are consistent with Seiller's more formal proposition (at least for computation and programs at the time of this thesis) that:

- computation correspond to dynamical systems (physical world);
- programs correspond to actions on a state space (computational world);
- algorithms correspond to specifications (mathematical/logical world).

[23] It reminds me of the beginning of Saint-Exupéry's "The Little Prince". The narrator tells a story about a child drawing of an elephant inside a boa but it looks like a hat in the point of view of adults. It is then possible to make the drawing more obvious by adding colours, adding eyes, making the boa transparent so that we can actually see the elephant.

§19.7 In this thesis, I will present a finitist candidate for "universal" computation[24]. The idea is to define a chaotic space of computation in which computational features such as synchronicity, sequentiality and direction of computation (which are less "chaotic" form of computation) are *constraints* on it. Moreover, we would like to minimise as much as possible "external intervention" in computation. Such a language would be *autonomous* or *autarkic*. For instance, in the case of boolean circuits, the evaluation function would be implemented at the same level as circuits themselves and not as an external procedure. In the case of tile systems, geometric constraints would be expressed in the same language as tiles themselves.

[24] It has to be noted that we are not looking for *the* universal model of computation but only a candidate or an attempt. It is basically the same but it sounds more modest.

Chapter 3

Linking logic and computation

Logic and Computation have been presented independently in the two previous chapters. In this chapter, several links between them are introduced.

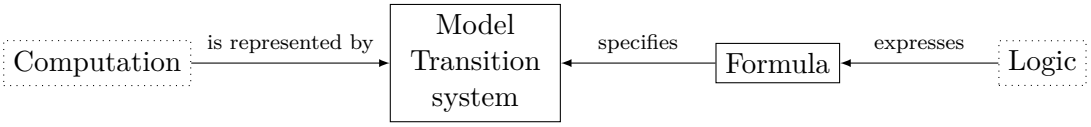
The link between logic and computation (and more precisely the Curry-Howard correspondence) is actually what made me study logic and led me to my current researches. This link is extremely fascinating and yet not so surprising after diving into details. What is fascinating is that the abstract and philosophical (I dare to say mystical) logic is related to down-to-earth and concrete computation, but also the possibility to *explain* and *justify* logic in concrete terms.

20 The different traditions of logic and computation

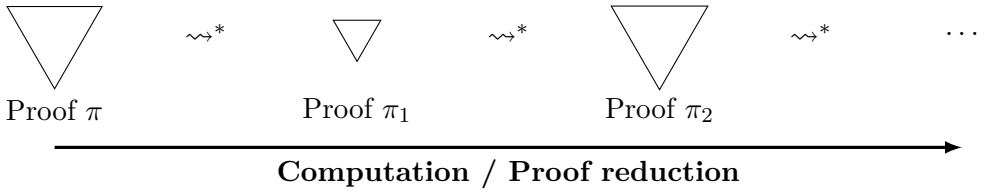
§20.1 According to Dale Miller's lecture notes[1] *"Proof theory, proof search, and logic programming"*, two approaches can be distinguished in order to link logic and computation.

- *Computation-as-model* represents computation with structures of predicate calculus (*cf.* Definition 7.8) and logical statements are used as tools to express properties over computation. Hence, computation and logic are distinct but related notions. This is what happens in model checking [BK08]: a system is represented by an automaton or more generally by a transition system represented some structure M and logic can express facts about the (un)reachability of some states. Given a formula A, it is satisfied by M when $M \models A$ (*cf.* Paragraph 7.13);
- *Computation-as-deduction* makes syntactic elements of logic coincide with computation. Hence computation and logic are partially identified. It is itself divided into two approaches depending on what we consider as computation. There are two approaches:
  - *proof normalisation* in which computation coincides with proof reduction (*cf.* Section 9) or cut-elimination (*cf.* Paragraph 10.5) for a given proof. We have a proof-program which is already there and we only have to reduce it;

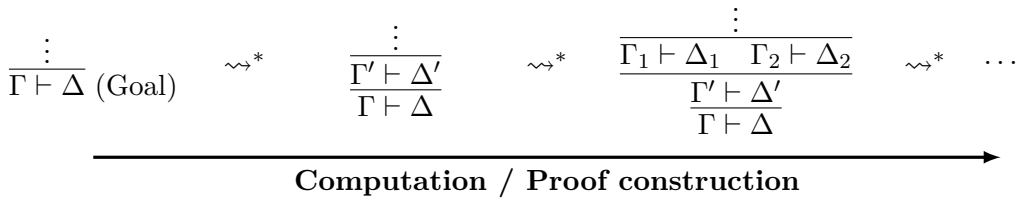
[1] http://www.lix.polytechnique.fr/Labo/Dale.Miller/mpri/ln2021-v3.pdf



(a) Computation-as-model: computation and logic are distinguished notions. Logic is used to state properties about the representation of a computational system.



(b) Computation-as-deduction (proof normalisation): we start from a whole given proof and reduce it with cut-elimination or proof reduction. Sequents are part of programs.



(c) Computation-as-deduction (proof search): we start from a sequent and try to prove it with strategies. Sequents represent states during a computational process.

Figure 20.1: The different traditions of linking logic and computation.

– *proof search* in which computation coincides with proof search (the process of automatically searching the proof of a given sequent). The proof itself is the result of the computation and it has to be constructed. The expression "computation as proof search" has been explicitly mentioned in Jean-Marc Andreoli's works in 1992 [And92].

§20.2 The different links between logic and computation are illustrated in Figure 20.1. In this chapter, only computation as deduction is explained in details (because I am more familiar with it, and otherwise this manuscript would be even longer than it already is).

21 Curry-Howard-Lambek correspondence

Natural deduction and Lambda-calculi

§21.1 **Brouwer-Heyting-Kolmogorov.** The Brouwer-Heyting-Kolmogorov (BHK) interpretation is an attempt at giving an informal definition of what a (constructive) proof should be. It has been introduced by Brouwer and his student Heyting but also independently by Kolmogorov. Unlike the truth interpretation of classical logic which is empty of information because of its reference to an external semantics of truth (are you always satisfied by a yes or no to your questions?), the BHK interpretation explain how to actually *construct* a proof (thus providing the *why* of provable statements).

- A proof of $A \wedge B$ is a pair (a, b) where a is a proof of A and b a proof of B;
- A proof of $A \vee B$ is either a pair $(0, a)$ where a is a proof of A or $(1, b)$ with b is a proof of B (the value 0 and 1 indicates whether it is a proof of A or B);
- A proof of $A \Rightarrow B$ is a *process/function/construction* turning a proof of A into a proof of B;
- There is no proof of $\bot$;
- The special element () is the unique proof of $\top$;
- A proof of $\forall x \in U.A$ is a function from $u \in U$ to a proof of A;
- A proof of $\exists x \in U.A$ is a pair (t, a) where t is a term (witness) and a a proof of A (certificate validating the witness).

As for atomic proofs, they are assumed to be given. The important point is that proofs of implication are functions. This is because proving that A leads to B can be seen as having a method to reach B from A. But what sort of function is it? This ambiguity is actually precious because it can be any kind of function (or process). In particular, Kleene had in mind computable functions.

$$\frac{}{\Gamma, x : A \vdash x : A}\ \text{var} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ \text{abs} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ \text{app}$$

$$\frac{}{\Gamma, A \vdash A}\ \text{ax} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow\text{i} \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \Rightarrow\text{e}$$

Figure 21.1: Correspondence between typing rule (simply typed λ-calculus) and logical rules (intuitionistic natural deduction).

§21.2 **Lambda-terms as proofs.** A possible choice of functional and computable objects to instanciate the BHK interpretation are simply typed λ-terms (*cf.* Section 15). A proof of $A \Rightarrow B$ becomes a term $\lambda x.M$ where x is a proof of A and M is a proof of B. This requirement exactly corresponds to the typing rule of function type in simply typed λ-calculus (*cf.* Figure 15.2). If we erase terms, we exactly obtain the natural deduction rule for implication (in sequent presentation). It shows that proving a statement A is the same as constructing a term of type A. Although a correspondence has already been remarked by Curry [Cur34] between combinatory logic and Hilbert (Frege) proof system, it is Howard [How80] who published the formal presentation of a proof-program correspondence for the λ-calculus. We give again the corresponding rules in Figure 21.1 in order to make the link even more obvious.

§21.3 The correspondence matches so well that modus ponens naturally corresponds to the application of function (but not their execution) and proof reduction corresponds to the reduction of terms. The correspondence is in both objects and their dynamics. Recall that proof reduction for $\Rightarrow$ corresponds to having an introduction of implication followed by its elimination, then reducing the initial proof to a new proof of B where the occurrences of A assumed in the proof of $A \Rightarrow B$ are replaced by the proof of A which has been fed to the implication by the elimination rule $\Rightarrow$e (*cf.* Figure 9.7). In terms of λ-calculus, this exactly corresponds the β-reduction $(\lambda x.M)N \rightsquigarrow_\beta M[x := N]$ which the requirement that everything is well-typed *w.r.t.* simple types.

§21.4 This formal correspondence is called *Curry-Howard-Lambek* (CHL) correspondence (or simply Curry-Howard). As Philip Wadler said[2], this correspondence is not limited to those systems and it shows that some ideas have been discovered twice independently: once by a logician (usually the first) and once by a computer scientist. We have that:

- Church's simply typed λ-calculus corresponds to natural deduction for minimal logic, which is intuitionistic logic restricted to $\Rightarrow$ as only connective (no negation as well);
- λ-calculus can be extended with more types and constructions illustrated in Figure 21.2:

[2] In his "Proposition as type" talk (https://www.youtube.com/watch?v=IOiZatlZtGU)

$$\frac{\Gamma \vdash a : A \quad \Delta \vdash b : B}{\Gamma, \Delta \vdash (a, b) : A \times B} \times \qquad \frac{\Gamma \vdash (a, b) : A \times B}{\Gamma \vdash a : A} \pi_1 \qquad \frac{\Gamma \vdash A \times B}{\Gamma \vdash b : B} \pi_2$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash (0, a) : A + B} \texttt{in}_1 \qquad \frac{\Gamma \vdash b : B}{\Gamma \vdash (1, b) : A + B} \texttt{in}_2 \qquad \frac{\Gamma \vdash x : \bot}{\Gamma \vdash x : A} \bot \qquad \frac{}{\vdash () : \top} \top$$

$$\frac{\Gamma \vdash e : A + B \quad \Delta_1, f_a : A \vdash C \quad \Delta_2, f_b : B \vdash C}{\Gamma, \Delta_1, \Delta_2 \vdash \texttt{match}(e, f_a, f_b) : C} \vee e$$

Figure 21.2: Extended rules for typed λ-calculus. The proof reduction for $\times$ corresponds to an application of projection and the proof reduction for $+$ corresponds to the application of pattern matching (which can be seen as the implication of a condition).

Natural deduction	**Typed λ-calculus**
Formula	Type
Implication $A \Rightarrow B$	Function type $A \to B$
Sequent $\Gamma \vdash A$	Typing assertion $\Gamma \vdash t : A$
Logical rule	Term constructor
Proof	Typed λ-term
Proof reduction	Term reduction
Irreducible proof	Result
Proving	Programming (constructing a program)
Conjunction $A \wedge B$	Product type $A \times B$
Conjunction introduction	Pair construction
Conjunction elimination	Par projection $\pi^1(x_1, x_2) = x_i$
Disjunction $A \vee B$	Sum type $A + B$
Disjunction introduction	Sum type construction
Disjunction elimination	Pattern matching
Top $\top$	Unit type
Bottom $\bot$	Empty type
Second-order universal quantification	Polymorphic type

Figure 21.3: Curry-Howard-Lambek correspondence between natural deduction and typed λ-calculus.

$$\frac{\Gamma, x : A \to \bot \vdash M : \bot}{\Gamma \vdash \mathbf{S}x.M : A}\ \mathbf{S} \qquad \frac{\Gamma, A \to \bot \vdash \bot}{\Gamma \vdash A}\ \text{dne}$$

  - an empty type $\bot$ and a unit type $\top$ only having () as term;
  - pairs (a, b) of type $A \times B$ are *projections* allowing to extract a component (their typing rules exactly correspond to the rules of conjunction);
  - sum (i, x) of type $A + B$ with either $i = 0$ and $x : A$ or $i = 1$ and $x : B$ (its typing rules correspond to the introduction of disjunction). This dives two types into the same abstract type. The proof-term (i, x) is often written $\iota_i(x)$ and is called *injection*. A typical example is the type $\texttt{Maybe}(A) := A + \top$ which may be of type A or nothing. This allows to represent partial functions;
  - pattern matching which analyses a sum type and provide a result for each case. This corresponds to disjunction elimination.

- Polymorphic λ-calculus (where terms can have a generic type, *e.g.* $\lambda x.x : \forall X.X \Rightarrow X$) corresponds to System F which is second-order logic (*cf.* Section 8) without first-order quantification (it can also be called *pure* second-order logic);
- System T which is a computational system featuring built-in constants for natural numbers and booleans, together with primitive recursion (always terminating), corresponds to first-order Peano arithmetic (PA).

We obtain a big correspondence illustrated in Figure 21.3. The CHL correspondence is still an active subject of research at the time of this thesis. We refer to Sørensen and Urzyczyn's lectures for more details [SU06].

§21.5 Perhaps the most important consequence is the reunion of two communities: proof theorists and type theorists. Ideas can be shared and a common language emerges. Ideas appearing in one field can be implemented in the other. In particular, this cultural fusion can be made concrete with *category theory* (usually attributed to Joachim Lambek).

The functional interpretation of classical logic

§21.6 The CHL correspondence was initially about *constructive* proofs because they are the ones which can materialise a (computational) process. For that reason, since classical is not constructive, it has been assumed that no computational correspondence existed for it.

§21.7 If we look at a classical proof in NK (for instance the one in Figure 9.6), we clearly see that something happens. The proof still *shows* something. By using the rule of double-negation elimination, what we do is making a formula A storable (in a bottom-up reading): no operation can be done with A but $\neg\neg A$ can introduce the hypothesis

$\neg A$. We can then extend the CHL correspondence with a new term $\mathbf{S}x.M$ with the rule of Section 21 which looks like the abstraction (abs) typing rule for λ-calculus. We then need more reduction rules not presented here but which can be found in the literature [SU06, Section 8.4].

§21.8 **Continuation-passing style programming.** The first hint of a CHL correspondence for classical logic has been accidentally[3] discovered by Timothy G. Griffin [Gri89] (who actually worked on another subject) by remarking a relation between Peirce's law $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$ (which is a classical axiom) and an operation in the programming language Scheme known as `call/cc` (call with current continuation). This operator allows to save the current context and load it later.

§21.9 Since there exists several ways to express classical logic (depending on what axiom your choose or by allowing several conclusions), we can obtain several different computational systems for classical logic. In particular, Parigot's $\lambda\mu$-calculus [Par92] uses a multi-conclusion sequent calculus and Krivine's machine uses Peirce's law.

The functional interpretation of quantification

§21.10 The BHK interpretation (*cf.* Paragraph 21.1) features rather a satisfying interpretation of quantifiers. Remark that quantifiers are functions and pairs *depending* over some alogical entity: individuals. Because quantifiers hold a computational behaviour, it is not so surprising that first-order quantifiers also enjoy a CHL correspondence.

§21.11 **Dependent type.** A type system which illustrates this dependency is the theory of *dependent types* [SU06, Chapter 10] which has been introduced independently of Howard's paper. It has originally been implemented in languages such as De Bruijn's Automath[4] for the verification of formal proofs using computers. It is common in mathematics to work with restricted objects depending on a parameter. For instance, we could have a type $\mathtt{Str}(n)$ of strings of $n \in \mathbf{N}$ characters and a type $\mathtt{Vect}(n)$ of vectors of size $n \in \mathbf{N}$.

§21.12 The theory of dependent types can be expressed in a calculus called λP in which only universal quantification is implemented. There are several categories of objects:

- *λ-terms* $M, N ::= x \mid \lambda x^A.M \mid MN$;
- *dependent types* $A, B ::= \alpha \mid \forall x^A.B \mid AM$ which can be of a certain *kind* κ and α is an atomic type (implication $A \to B$ is defined by $\forall x^A.B$ where x does not appear free in A);
- *kinds* $\kappa ::= * \mid \Pi x^A.\kappa$ where $*$ represents the class of all types.

[3] According to what I have heard.
[4] One of the first direct application of the Curry-Howard correspondence.

Remark that we have a functional behaviour at the level of terms, types and kinds. Kinds are sort of types over types: types A are of a certain kind κ, which is written $A : \kappa$. We could define $\texttt{Str} := \Pi n^{\texttt{int}}.*$ and construct $\texttt{Str}(n) : *$ with the rules of [SU06, Section 10.2]. It is then possible to construct a term $\texttt{fillZero} : \forall n^{\texttt{int}}.\texttt{Str}(n)$ which constructs a string of n characters 0. Of course, we also need more reduction rules to apply arguments to types and not only the β-reduction which only applies to terms. As for existential types $\exists x.A$, they correspond to a pair (x, A) of a term and a type, expressing a dependency between x and A.

§21.13 Another precious idea appearing in Martin-Löf's intuitionistic type theory [MLS84] directly and explicitly implements the CHL correspondence. Terms are seen as proofs and we write $M \in A$ to express the fact that M is of type A, or that M is a *proof* of the formula A (an ambiguity allowed by the CHL correspondence). This allows to mix functional programming and proving; an essential feature of *proof assistants* such as Coq. If we would like to prove some goal $\vdash A \Rightarrow B$, then it is sufficient to construct a term (program) taking an input of type A and producing any input of type B. For instance, if we had a type `Nat` of natural numbers then the representation of the function $\texttt{succ}(n) = n + 1$ in λ-calculus would be a proof of $\texttt{Nat} \Rightarrow \texttt{Nat}$. More than proving with programs, it is also possible to construct programs from a proof, thus producing *certified programs* which are ensured to have the behaviour we wish for.

§21.14 How these results show how rich the CHL correspondence is. It can be applied for real-world situations and is still an active field of research with several branches and subcultures. It is not limited to natural deduction: we even have several term assignments to sequent calculus [SU06, Section 7.4] or construction of inductive types such as **N** which is defined by self-reference.

22 Realisability

My personal reference for this section is Etienne Miquey's PhD thesis [Miq17]. Other useful references on the subject are Lionel Rieg's PhD thesis [Rie14] and Alexandre Miquel's HdR thesis [Miq09].

Kleene realisability

§22.1 BHK interpretation (*cf.* Paragraph 21.1) is very generic and can be instantiated by other objects, not only computable functions or λ-terms. In realisability, instead of proofs, more general objects called *realisers* are considered.

§22.2 **Realisability interpretation.** We write $t \Vdash A$ when t is a realiser of A (or that A is realised by t). One of Kleene's idea was to interpret intuitionistic formulas by encoding (*cf.* Paragraph 16.9) partial recursive functions (*cf.* Paragraph 13.8) by natural numbers

(he uses Gödel's encoding in particular). We write $\varphi(n, m)$ for the application of the function represented by n to the argument represented by m. We write $\#(n, m)$ for the encoding of pairs (*cf.* Paragraph 16.9). We obtain the following variant of BHK:

- nothing realises $\bot$;
- $0 \Vdash \top$;
- $\#(n, m) \Vdash A \wedge B$ when $n \Vdash A$ and $m \Vdash B$;
- $\#(i, n) \Vdash A \vee B$ when either $i = 0$ and $n \Vdash A$ or $i = 1$ and $m \Vdash B$;
- $n \Vdash A \Rightarrow B$ when for all $m \Vdash A$, we have $\varphi(n, m) \Vdash B$.

§22.3 **Type theory and Realisability.** A question that quickly comes to mind (and it is what I thought too in the beginning) is how it differs from typing as in typed λ-calculus.

- type theory is about using syntactic rules to verify the type of a term. It is about verifying the *shape* or *structure* of a term in order to ensure that it will behave well;
- realisability is about *associating* computational entities to formulas or giving computational representatives for formulas. Realisability is more abstract and computational as we can see in the definition of $n \Vdash A \Rightarrow B$ which tests a function against all possible arguments. The realisation of implication is more about the *behaviour* of a function than its *structure*.

Actually, realisability generalises typing. This is represented by a property called *adequacy* formalised by the implication $t : A \Longrightarrow t \Vdash A$ where $t : A$ is the fact that t has type A. This can be understood as having the computational behaviour stated by A when having the structure of A. Interestingly, because this is a computation association and not an inference about structure, nothing forbids contradictory, unprovable and undecidable formulas to be realised. It happens that some programs have a computational behaviour which is sound without fitting typing rules. Because of that it is undecidable in general to tell whether we have $t \Vdash A$ for a given realiser t and a formula A. According to Alexandre Miquel[5], in type theory, we are interested in correctness *w.r.t.* rules/typing and in realisability to correctness *w.r.t.* execution.

§22.4 The interpretation can be extended to first-order intuitionistic logic, called Heyting Arithmetic (HA). By the CHL correspondence, it corresponds to a computational system for primitive recursive functions called System T (attributed to Gödel), which can be presented as a variant of λ-calculus. Realisability gives tools to reason about the relationship between logic and computation. In particular, it is even possible to attribute a computational content to the axioms of HA. The choice of a realiser can be rather

[5] https://www.fing.edu.uy/~amiquel/realiz21/kleene.pdf

arbitrary providing that it is computationally consistent. For instance, we can have the following realisation of axiom of HA:

$$\lambda x.\lambda y.\lambda z.z \Vdash \forall x.\forall y.(s(x) = s(y) \Rightarrow x = y).$$

Some common properties which can be proven with the help of the realisability interpretation are the following:

◇ **Soundness** if $\vdash_{HA} A$ then there exists t such that $t \Vdash A$;

◇ **Consistency** $\nvdash_{HA} \bot$ (proven by the fact that $\bot$ cannot be realised).

But, what about classical logic?

Krivine realisability

§22.5 An attempt of extending the interpretation to classical logic is given by Jean-Louis Krivine [KCHM09]. Krivine realisability is a complete reformulation of the theory of realisability [Miq17, Section 3.1.2] and has the ambition of giving a computational interpretation of important axiomatic systems such as ZFC axioms for set theory (*cf.* Paragraph 3.5). The extension to classical logic is related to the classical CHL correspondence using the `call/cc` operator (*cf.* Paragraph 21.8).

§22.6 **Lambda-calculus with continuations.** I will give an alternative definitions. A formula or type A can be seen as a set of realisers $|A|$. Krivine's realisers are λ-terms extended with continuations in a system called λ_c-calculus. Terms are executed *w.r.t.* a context or environment taking the form of a stack.

$$M, N ::= x \mid \lambda x.M \mid MN \mid \mathbf{k}_\pi \mid \mathtt{cc} \qquad \text{(Terms)}$$

$$\pi ::= \alpha \mid M \cdot \pi \qquad \text{(Stacks)}$$

where α is a stack constant coming from a set of stack constant. It is also possible to consider more instructions beside `cc` for terms. Notice that the constant $\mathbf{k}_\pi$ is associated with a whole stack. This will be used to store a context to load it later during the computation.

§22.7 **Krivine machine.** What is executed is not a term alone but a term in the context of a stack. We write $M \star \pi$ for a connexion between a term and a stack. Such connexions are called *processes* and are written with variables p and q. The execution of a process is done by a *Krivine Abstract Machine* (KAM) which has 4 transition rules:

Push	$MN \star \pi$	$\rightsquigarrow$	$M \star N \cdot \pi$	(argument is postponed)
Grab	$\lambda x.M \star N \cdot \pi$	$\rightsquigarrow$	$\{x := N\}M$	(function application)
Save	$\mathtt{cc} \cdot M \cdot \pi$	$\rightsquigarrow$	$M \star \mathbf{k}_\pi \cdot \pi$	(the context is saved)
Restore	$\mathbf{k}_\pi \star M \cdot \pi'$	$\rightsquigarrow$	$M \star \pi$	(the context is loaded)

We write $\rightsquigarrow^*$ for the reflexitive-transitive closure of $\rightsquigarrow$ (corresponding to multiple steps of reduction).

§22.8 **Example.** $(\lambda xy.x)z \star \varepsilon \overset{push}{\rightsquigarrow} \lambda xy.x \star z \overset{grab}{\rightsquigarrow} \lambda y.x\{x := z\} \star \varepsilon = \lambda y.z \star \varepsilon$.

§22.9 We now would like to study computational interpretations of classical logic by realisability. We write Λ for the set of all λ_c-terms and Π for the set of all stacks. The idea is to oppose terms and stacks such that stacks are sort of opponents for terms. The interpretation is then parametrised by a way to assert whether a term and a stack *interact correctly*.

§22.10 **Definition** (Pole)**.** A *pole* is a chosen set $\perp\!\!\!\perp \subseteq \Lambda \times \Pi$ which is closed under anti-evaluation, *i.e.* both $p \rightsquigarrow^* p'$ and $p' \in \perp\!\!\!\perp$ implies $p \in \perp\!\!\!\perp$ for all processes $p, p' \in \Lambda \times \Pi$.

§22.11 The requirement of anti-evaluation ensures that membership to the pole is sound *w.r.t.* execution by the steps of the KAM.

§22.12 **Test interpretation.** Given a pole $\perp\!\!\!\perp$, a term M and a stack π, the fact that $M \star \pi \in \perp\!\!\!\perp$ formalises the fact that M and π interact correctly. It is then possible to see stacks as sort of *tests* that terms have to pass in order to be part of a some formula. For instance, consider that we have a set of stack $\|A\|$ associated with a formula A. Then the set of all M such that $M \star \pi$ for all $\pi \in \|A\|$ defines the realisers of the formula A.

§22.13 **Orthogonality.** Given a pole $\perp\!\!\!\perp$, it is possible to define an *orthogonality relation* between terms and stacks: we have $M \perp \pi$ if and only if $M \star \pi \in \perp\!\!\!\perp$. It is then possible to define the orthogonal of sets of terms and stacks:

- ◇ **For set of terms** $S^\perp := \{\pi \in \Pi \mid \forall M \in S. M \perp \pi\}$;
- ◇ **For set of stacks** $S^\perp := \{M \in \Lambda \mid \forall \pi \in S. M \perp \pi\}$.

Such orthogonality relations are usually symmetric and is an alternative definition of "good interaction" or "passing a test".

§22.14 **Behaviour.** Once an orthogonality relation is fixed, what will usually correspond to formula are *behaviours* which are set of terms S such that $S = S^{\perp\perp}$. Although this definition seems abstract, it can be understood as a "closure by interaction". Another equivalent definition is that S is a behaviour when there is some set $\texttt{Tests}(S)$ such that $S = \texttt{Tests}(S)^\perp$, which can be understood as S being *testable* or characterised by a set of tests.

§22.15 Using these definitions, it is then possible to give a computational interpretation of classical second-order Peano arithmetic (which was Krivine's initial goal) [Miq17, Definition 3.8]. It shows how realisability interpretation have the power to *reconstruct* logical systems from choosing a computational system and a right definition of pole.

Reconstruction of simple types

§22.16 By using similar ideas, it is possible to consider simpler reconstructions. An interpretation of Colin Riba's *"Strong Normalization as Safe Interaction"* paper [Rib07] is that typed λ-calculus (a logical system) is reconstructed from untyped λ-calculus (a purely computational system without types).

§22.17 We consider the usual untyped λ-calculus with another notion of context (instead of stacks):

$$\begin{array}{rlll} M, N ::= & x \mid \lambda x.M \mid MN & \text{set } \Lambda & \text{(Terms)} \\ E[] ::= & [] \mid E[]M \mid ME[] & \text{set } \mathcal{E} & \text{(Contexts)} \end{array}$$

with $E[M]$ defined by $E[]$ where $[]$ is replaced by M. A context is a λ-term with holes and we expect it to be filled by another term. It can also be seen as an incomplete term.

§22.18 We can define a pole $\perp\!\!\!\perp$ defined by the set of all strongly normalising terms (all paths of reduction terminate). It is then possible to reconstruct simple types from the orthogonality relation associated with $\perp\!\!\!\perp$ by an interpretation $[\![\cdot]\!]$:

- $[\![\alpha]\!] := \perp\!\!\!\perp$ (which is shown to be a behaviour by Riba [Rib07, Proposition 4.6]);
- $[\![A \Rightarrow B]\!] := ([\![A]\!] \cdot [\![B]\!]^{\perp})^{\perp}$

where for $A \subseteq \Lambda$ and $B \subseteq \mathcal{E}$, $A \cdot B$ is defined by $A \cdot B := \{E\big[]M\big] \mid M \in A, E[] \in B\}$.

§22.19 This result shows that type systems are not necessarily primitive notions and can be in some sense *justified* by computation (although Riba does not state it like that). The reconstruction with strongly normalising terms as basis is not the only solution. Riba also shows that there exist alternative definitions [Rib07, Section 6] and that both conjunction and disjunction can be defined as well.

23 Logic programming

Reasoning with programs

§23.1 Predicate calculus is very convenient to express statements of natural language. For instance, the fact that my brother and I have the same mother can be represented by the formula $\texttt{mother}(i) = \texttt{mother}(\texttt{brother}(i))$ where i is a constant (or nullary function symbol) representing me. We can also express the fact that some brothers have a different father by the formula $\exists p.\exists p'.(\texttt{brother}(p) = p') \wedge \neg(\texttt{father}(p) = \texttt{father}(p'))$.

§23.2 Kowalski's paper *"Predicate logic as programming language"* [Kow74] shows that predicate calculus can be seen as a programming language for symbolic reasoning. Not in the CHL sense that proofs correspond to programs but in the sense that:

```
add(0, Y, Y).
add(s(X), Y, s(Z)) :- add(X, Y, Z).
?add(s(s(0)), s(s(0)), R).
```

Figure 23.1: Example of logic program computing unary addition. The last line is the query.

- a set of assumed formulas can be seen as a database of facts or a knowledge base;
- formulas of predicate calculus can be seen as *goals*, *questions* or *queries*;
- formulas $(A_1 \wedge ... \wedge A_n) \Rightarrow B$ can be seen as *inference rules*, that is, ways to infer a fact B from facts $A_1, ..., A_n$.

In particular, formulas representing questions can contain free variables and answers will not be truth values but terms replacing variables which are consistent with the facts.

§23.3 An example of logic program computing unary addition is given in Figure 23.1 where `B :- A` stands for $A \Rightarrow B$ (if you wish to conclude B then you have to justify A). A standard representation of natural numbers is to use unary numbers: a natural number is either 0 or the successor or a natural number, which can be represented by the application of a symbol `s` over another natural number. A natural number is then a sequence of application of `s` over 0. The expression $\texttt{add}(X, Y, Z)$ is a ternary predicate expressing the fact that $X + Y = Z$. The first expression says that $0 + Y = Y$ for any Y and the second expression that if $X + Y = Z$ then $(X + 1) + Y = Z + 1$. The query asks for the result of the sum $2 + 2$. The next sections are dedicated to formalising this symbolic reasoning and providing ways to answer queries.

§23.4 Terms are used to represent statements. In particular, we are interested in solving equations $t \stackrel{?}{=} u$ between terms by searching for terms replacing variables such that it would make t and u equals. Equations between terms is usually attributed to Robinson [R+65] but Herbrand [Her30] already studied term unification before in the context of his investigations on mathematical equations and proof theory.

§23.5 We recall elementary definitions of term unification [Her30] in Appendix B. We refer the reader to the article of Lassez et al. [LMM88] for more details which are often omitted in the literature or Baader et al. [BN98] for a broader view. We use uppercase letters such as X, Y, Z for variables and lowercase letters a, b, c, f, g and h for function symbols.

Normal forms

§23.6 Now that we have ways to construct sentences, we could simply use predicate calculus but it is not convenient enough to use it as it is. It has a lot of of logical rules whereas what we wish for is simply an interaction between queries and clauses (either facts or

$((A \Rightarrow B) \Rightarrow A) \Rightarrow A$	translation of implication
$\rightsquigarrow \neg(\neg(\neg A \vee B) \vee A) \vee A$	use De Morgan laws
$\rightsquigarrow (\neg\neg(\neg A \vee B) \wedge \neg A) \vee A$	$\neg\neg$-elimination
$\rightsquigarrow ((\neg A \vee B) \wedge \neg A) \vee A$	distribute $\vee$
$\rightsquigarrow ((\neg A \vee B) \vee A) \wedge (\neg A \vee A)$	remove superflous parentheses
$\rightsquigarrow (\neg A \vee B \vee A) \wedge (\neg A \vee A)$	

Figure 23.2: Transformation of Peirce law to its conjunctive normal form (CNF). Notice that two occurrences of excluded middle appear in conjunction (the B is not essential).

inference rules producing new goals). This section introduces ways to transform formulas in other more convenient forms.

§23.7 **Conjunctive normal form.** A simple transformation which can be done in propositional calculus is to transform formulas in conjunctions of disjunctions [HR04, Section 1.5.2]. This can be done with the following operation over formulas:

- translate all symbols (such as $\Rightarrow$) into formulas of the functionally complete set $\{\neg, \wedge, \vee\}$ for instance with:
  - $A \Leftrightarrow B \rightsquigarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$;
  - $A \Rightarrow B \rightsquigarrow \neg A \vee B$;
- first push all negations to atomic formulas with De Morgan laws and double-negation elimination:
  - $\neg\neg A \rightsquigarrow A$;
  - $\neg(A \wedge B) \rightsquigarrow \neg A \vee \neg B$;
  - $\neg(A \vee B) \rightsquigarrow \neg A \wedge \neg B$;
- distribute disjunctions everywhere with:
  - $A \vee (B \wedge C) \rightsquigarrow (A \vee B) \wedge (A \vee C)$;
  - $(B \wedge C) \vee A \rightsquigarrow (B \vee A) \wedge (C \vee A)$.
- remove superfluous parentheses if you want, so that it is easier to read.

§23.8 Formulas in conjunctive normal form (CNF) have the uniform shape $(A^1_1 \vee ... \vee A^1_{n_1}) \wedge ... \wedge (A^m_1 \vee ... \vee A^m_{n_m})$ where the disjunctions are called *(disjunctive) clauses* and conjunctions can be seen as *collection of clauses*. This can be presented in a more friendly way: as multisets (*cf.* Appendix A.2) of clauses. We write clauses $[A_1, ..., A_n]$ instead of $A_1 \vee ... \vee A_n$ and multiset of clauses with a sum notation $C_1 + ... + C_n$ instead of $C_1 \wedge ... \wedge C_n$. An example of transformation of a formula into its CNF is given in Figure 23.2. The

alternative notation for that example would be $[\neg A, B, A] + [\neg A, A]$. It is also possible to transform formulas into disjunctive normal forms (DNF) corresponding to disjunctions of conjunctions.

§23.9 **Prenex normal form.** Now that we are able to normalise formulas of propositional calculus, we have to treat quantifiers to extend the normalisation to predicate calculus. What is usually done is to push all quantifiers to the front of the formula. We then obtain what we call a *prenex normal form* (PNF) [Hed04, Section 3.2.1]. This can be done with the following transformations:

- first translate the formula into an equivalent formula of $\{\forall, \exists, \neg, \wedge, \vee\}$;
- apply $(\forall x.A) \wedge B \rightsquigarrow (\forall x.A \wedge B)$ and $(\forall x.A) \vee B \rightsquigarrow (\forall x.A \vee B)$ everywhere;
- apply $(\exists x.A) \wedge B \rightsquigarrow (\exists x.A \wedge B)$ and $(\exists x.A) \vee B \rightsquigarrow (\exists x.A \vee B)$ everywhere;
- apply $\neg\exists x.A \rightsquigarrow \forall x.\neg A$ and $\neg\forall x.A \rightsquigarrow \exists x.\neg A$ everywhere.

The first two rules only hold when B is independent from A, *i.e.* the variable x is not free in B. If it happens to be the case, the variable bound x in A can be renamed so that it is not in conflict with the free variable x in B. If we combine PNF with CNF then we can obtain a *prenex conjunctive normal form* (PCNF) $Q_1x_1...Q_nx_n.C_1 + ... + C_n$ where C_i is a disjunctive clause and $Q_i \in \{\forall, \exists\}$.

§23.10 **Skolem normal form.** It is also possible to do even more by eliminating existential quantification. This transformation is known as *skolemisation* (or *Herbrandisation* if we remove universal quantification). A formula which is skolemised is said to be in *skolem normal form* (SNF) [Hed04, Section 3.2.2]. The idea is that whenever we have a universal quantification preceding an existential one as in $\forall x.\exists y.A$, it is the same as saying that there is a function f turning x into the "right" y which satisfies the formula. Existential quantification can then be removed by explicitly introducing a new function symbol f corresponding to that function. For instance, consider the formula $\forall n.\exists m.n < m$ stating that for all natural number n, there is a greater natural number m. We know that the function successor s takes a natural number n and produces its successor $n < s(n)$. Hence the formula can be rewritten $\forall n.n < s(n)$. Without expliciting what the function is, giving a universe or a model of the formula, the two formulas are logically equivalent and this is what matters anyway.

§23.11 Skolemisation is applied over a formula $Q_1x_1...Q_nx_n.C_1 + ... + C_n$ in PCNF by successively selecting an existential quantifier from left to right until the last one.

- if the selected $\exists$ has no universal quantifier preceding it, then

$$\exists x.A \rightsquigarrow \{x := \mathsf{c}\}A$$

where c is a fresh constant not appearing in A;

$$\frac{\Gamma \cup \{P(t_1, ..., t_n)\} \qquad \Delta \cup \{\neg P(u_1, ..., u_n)\}}{\theta(\Gamma \cup \Delta)} \text{ Res}$$

Figure 23.3: Robinson's resolution rule. The substitution θ is defined as the solution of the equation underlying the two interacting atoms, that is $\theta := \texttt{solution}\mathcal{P}(\bigcup_{k=1}^{n}\{t_i \stackrel{?}{=} \alpha u_i\})$. We require that the two clauses are independent: there is a renaming α making all variables of $\Delta \cup \{\neg P(u_1, ..., u_n)\}$ disjoint from variables of $\Gamma \cup \{P(t_1, ..., t_n)\}$ (hence θ is an α-unifier).

- if the selected $\exists$ has n consecutive universal quantifiers preceding it, then

$$\forall x_1 ... \forall x_n . \exists x . A \rightsquigarrow \forall x_1 ... \forall x_n . \{x := f(x_1, ..., x_n)\} A$$

  where f is a new function symbol not appearing in A, which depends upon $x_1, ..., x_n$.

§23.12 By combining all these transformations we can obtain a normal form for formula of predicates calculus. All formulas can be rewritten as a formula $\forall x_1 ... x_n . A$ where A has no quantifiers and is in CNF. Universal quantifiers can even be hidden and the variables $x_1, ..., x_n$ occurring in A will be simply considered generic.

First-order resolution

§23.13 With the rise of computers, methods of automated reasoning began to appear. In particular, the DP algorithm (which has been extended to the DPLL algorithm[6]) developed by Davis and Putnam [DP60] provided a way to check if a propositional formula in CNF was satisfiable.

§23.14 The idea of the DP(LL) algorithm is to construct a partial truth valuation and extend it while keeping consistency. It is more efficient than naive brute force search which would try all possibilities. It appears that all satisfiability algorithms have the same complexity: in the worst-case performance, their number of steps is exponential in the size of their input[7]. What is meant by "keeping consistency" is that, for instance, if we had a clause $[A]$ then we can remove all occurrences of $\neg A$ in other clauses since both A and $\neg A$ cannot hold at the same time. More generally, it is possible to deduce a general rule producing $C \vee D$ from $A \vee C$ and $\neg A \vee D$. This is known as the *resolution rule*.

[6] The DPLL algorithm was one of the first things I studied on my own. For some reasons, I was fascinated by it and the fact that computers could solve logical problems. I believe that implementing satisfiability algorithms is a good exercise. If you are interested in implementing automated reasoning algorithms then I think Harrison's *"Handbook of Practical Logic and Automated Reasoning"* [Har09] is a good reference.

[7] There a lot of optimisation giving a number of steps lesser than 2^n for an input of size n but it is still very close. For instance, $2^{n/2} < 2^n$.

$$\dfrac{\dfrac{[A,B] \qquad \dfrac{[\neg A, C] \qquad [\neg C]}{[\neg A]}\,\text{Res}}{[B]}\,\text{Res} \qquad \dfrac{[\neg B, D] \qquad [\neg D]}{[\neg B]}\,\text{Res}}{\bot}\,\text{Res}$$

Figure 23.4: Resolution tree for the refutation of $[A, B] + [\neg A, C] + [\neg B, D] + [\neg C] + [\neg D]$ with the resolution rule. Example taken from Hedman's book [Hed04, Example 3.29].

§23.15 **Resolution rule.** Robinson suggested an algorithm focussing on resolution [R+65] as a way to refute statements of predicate calculus. Resolution can be seen as a logical rule over formulas in normal form (*cf.* Figure 23.3). In this context, predicate calculus has been so normalised to the extreme that the resolution rule is self-sufficient and no other logical rules are needed[8]. The empty set of clause is written $\bot$ and corresponds to a contradiction. The goal is to construct a *resolution tree* to infer $\bot$ with a given set of clauses, by reusing as many occurrences of clauses as we want. The meaning of the resolution rule is that it produces a *consensus* between two clauses by *resolving* contradictions.

§23.16 **Independence of clauses.** Notice that in Figure 23.3, there is an additional requirement of independence between the two clauses being connected. This is due to the fact that $\forall x.A \wedge B$ is equivalent to $(\forall x.A) \wedge (\forall x.B)$. Hence each clause can have its own local bound variables. Without this requirement, we would not be able to infer *bot* from $[P(a, x)] + [\neg P(x, b)]$ since they share a variable which cannot be instantiated to both a and b. However, these two clauses are contradictory since they represent the formula $\forall x.P(a, x) \wedge \neg P(x, b) \equiv (\forall x.P(a, x)) \wedge (\forall x.\neg P(x, b))$ where $\forall x.P(a, x)$ can be instantiated to $P(a, b)$ and $\forall x.\neg P(x, b)$ to $\neg P(a, b)$ which is contradictory with $P(a, b)$.

§23.17 An example of resolution tree is given in Figure 23.4. If a set of clauses is inconsistent then there must be a resolution tree inferring $\bot$ from its clauses. Similarly to reasoning by contradiction, if we would like to show that a formula A is consequence of a set of clauses $C_1 + ... + C_n$, then it is sufficient to provide a refutation of $C_1 + ... + C_n + [\neg A]$ (assuming that A is false leads to a contradiction).

§23.18 Actually, if we are interested in the logical meaning of resolution, then the resolution rule alone is not exactly sufficient. If we consider the set of clauses $[A, B] + [\neg A, \neg B]$, although inconsistent, it will never reach the empty clause $\neg$. Additional rules such as "factoring" or other logical simplifications have to be considered. Since this is a minor technical detail that will not be used in this thesis, we choose to ignore it.

§23.19 Resolution trees are constructed from top to bottom. Their construction is non-trivial. It is as if we were only using the cut rule in sequent calculus. For that reason, it is not

[8] Notice that it is actually an instance of monolateral cut rule (*cf.* Figure 11.1).

clear at all how we should apply the resolution rule: it is possible to get stuck or to loop although there was a good series of choices leading to $\bot$ (when it is possible to infer it). This is related to the fact that the satisfiability of formulas for predicate calculus is undecidable. Hence refutation is subject to the same problem since refuting $\neg A$ is equivalent to showing satisfiability of A.

§23.20 Logic programming is based on the resolution principle. However, strategies and constraints have to be designed in order to have a practical use of it. For instance, it is usually not necessary to consider multiple conclusions when reasoning. It is sufficient to consider implications $(A_1 \wedge ... \wedge A_n) \Rightarrow B$ as shown in Section 23.

Reasoning with Horn clauses

§23.21 Horn clauses [Hor51, Tär77] are type of clauses which are easier to use for practical applications. They correspond to clauses with at most one positive literal (without negation). For instance $[\neg A, B, C]$ is not a Horn clause but $[\neg A, \neg B, C]$ is. The intuition is that such clauses correspond to implications "if ... then ..." of the shape $(A_1 \wedge ... \wedge A_n) \Rightarrow B$ because

$$(A_1 \wedge ... \wedge A_n) \Rightarrow B \equiv \neg(A_1 \wedge ... \wedge A_n) \vee B \equiv \neg A_1 \vee ... \vee \neg A_n \vee B.$$

We will sometimes write the more convenient sequent notation $A_1, ..., A_n \vdash B$ where $A_1, ..., A_n, B$ are atoms instead[9].

§23.22 It is possible to distinguish three classes of Horn clauses. Clauses

- $\vdash A$ with one positive atom and no negative ones, called *facts* (they are assumed);
- $A_1, ..., A_n \vdash A$ with exactly one positive atom, called *inference rules*;
- $A_1, ..., A_n \vdash$ with no positive atom, called *queries*.

These classes correspond to the three kind of elements we needed for logic programming (*cf.* Section 23). The idea is that a query $P(t) \vdash$ corresponds to a clause $[\neg P(t)]$ which can interact either directly with a unifiable fact $\vdash P(u)$ or a unifiable conclusion $P(u)$ of an inference rule $A_1, ..., A_n \vdash P(u)$. In the latter case, we need to answer the new query $A_1, ..., A_n \vdash$.

§23.23 **SLD-resolution.** In order to reason with a computer, several resolution strategies have been designed [KK71]. The SLD-resolution is a strategy which selects a clause (usually in a query) and makes it successively interact with other clauses linearly in order to produce a new resulting clause (called *resolvent*). SLD stands for selective-linear-definite. It is also possible to enrich set of clauses with an order so that a query will connect to the first matching clause, as in the logic programming language Prolog [CR96].

[9] Remark that Horn clauses in sequent notation correspond to intuitionistic sequents.

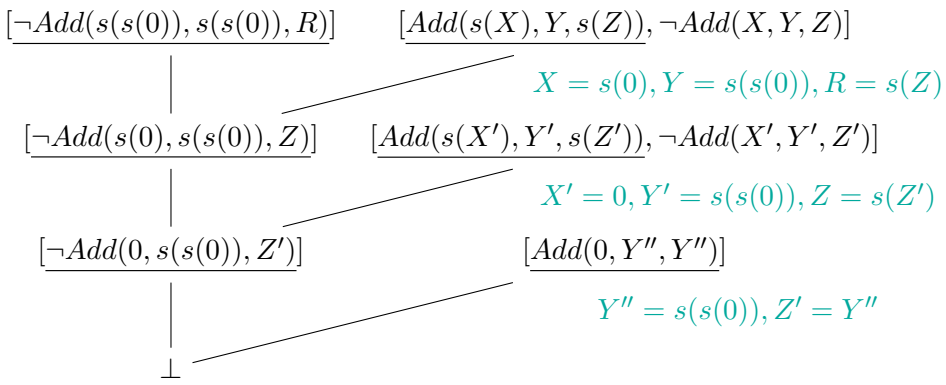


Figure 23.5: Answer of the query $2 + 2 \vdash$ for unary addition with Horn clauses and SLD-resolution. When reaching $\bot$, we can read the result located in R by unifying the values of variables. We have $Z' = Y'' = s(s(0))$, hence $Z = s(s(s(0)))$. And since $R = s(Z)$, we finally have the result $R = s(s(s(s(0))))$, meaning that $2 + 2 = 4$.

§23.24 An example of application of SLD-resolution is given in Figure 23.5. Answers are given by finding a consistent term replacing free variables of the query. There can be several answers for all the free variables of the query or no answer at all. The answer can just be "Yes" if there is no free variable and we are just trying to justify a query seen as a goal.

§23.25 Logic programming had a huge impact during the rise of computer science as several various resolution systems have been built [Lei12, Sic76, Kow75]. It also led to variants of logic programming such as answer set programming (ASP) [Gel08, EIK09] or disjunctive logic programming [LRM91, Min94]. After declining for a long time, it is coming back again with recent works such as Andreoli's [And92] work or Miller's works [Mil21] relating proof theory (with applications to linear logic) and logic programming.

24 Discussion: the limits of the proof-program correspondence

Although it is not related to what I present in this section, some notes of Laurent Regnier (which I find very interesting) discuss the limits of the CHL correspondence[10].

§24.1 **The strict meaning of CHL.** Remark that, stricly speaking, not all logical and computational systems are concerned by CHL.

- Only logical systems enjoying a cut-elimination theorem are considered. It is not the case of all logical systems. A dynamics of proofs is needed because otherwise

[10] https://www.i2m.univ-amu.fr/perso/laurent.regnier/articles/ch.pdf

we would not be able to execute the corresponding programs[11].

- Usually, typed functional systems are considered. Untyped λ-calculus can produce contradictions (*cf.* Paragraph 15.9) and is rejected from the correspondence for that reason. It is not "logical".

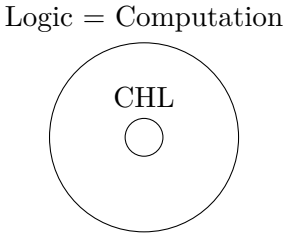
It is a very specific correspondence which cannot be (yet?) extended to all logic and computation. I would like to stress this point: the CHL correspondence is a *historical coincidence*. Actually we do not learn so much apart that two communities were doing more or less the same thing with different vocabulary and methods. More pessimistic: it may even mean that we were *not able* to distinguish the notion of computation from the notion of logic. They are densely mixed together and we are more and more confused with the meaning of the objects we work with. What logic and computation are is still unclear at this point.

§24.2 It seems to me that there are currently three main understanding of the link between logic and computation. I illustrate these points of view in Figure 24.1.

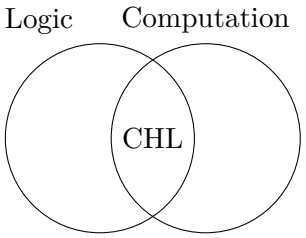
- The first understanding in Figure 24.1a corresponds to the famous (and exaggerated) slogan "proving is programming" which is full of hopes, as if computer science colonised logic and tried to civilise it. However, we were not able to completely merge two cultures, thus obtaining a sort of fake multiculturalism.
  - The C language, Java, Turing machines, Wang tiles, Babbage's engines are computational but are they logical?
  - When I discuss with people and use logical arguments, when I am thinking, when I am doing mathematics, I am probably logical but do I always compute? With the classical meaning of computation?

  Even if the answers to the previous questions were affirmative, it does not necessarily mean that logic and computation are the same but only that they are difficult to distinguish from our point of view. They could be two distinct (but difficult to separate) dimensions of reality.
- The point of view in Figure 24.1b is what I thought to be true at the beginning of my PhD thesis. In this point of view, the CHL correspondence is only an intersection between the notion of logic and computation. A perfect point where they match. This might be because logic and computation are about a lot of different things but they both have a common part speaking about the *process of construction* to reach a syntactic goal and an evaluation from the implicit to the explicit. But, under which basis can we judge that Turing machines are not

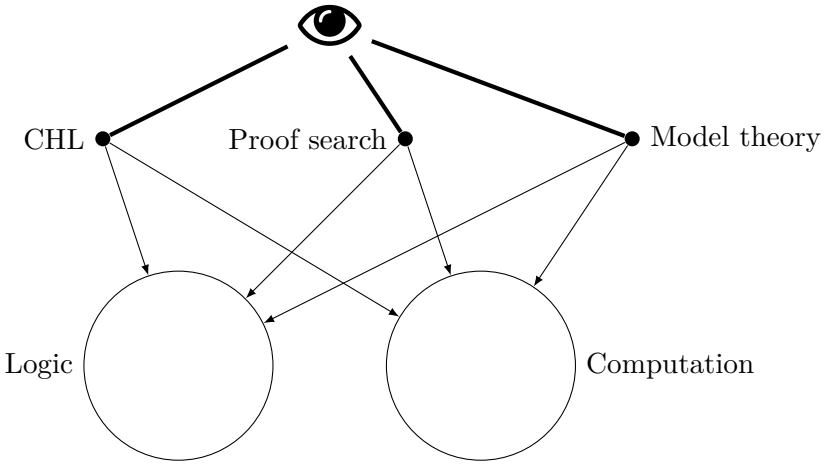
[11] This is probably more profound than we think. I once had a discussion with Paul Séjourné and we reached the conclusion that if cut-elimination (or proof reduction) is a procedure from the implicit to the explicit, it is a sort of *explanation*. What cannot be executed cannot be explained. Logical systems without execution are hence *transcendent* in some sense. This is related to my reflection in Section 19.



(a) CHL is the tip of an iceberg hiding a uniform logico-computational space. Proving is programming.



(b) CHL is the intersection between logic and computation. Some logics are not computational and some models of computation are not logical (Turing machines).



(c) It is all about point of view, representations and models. We can identify some parts of one to some parts of the other. Logic and computation are distinct entities which can be related.

Figure 24.1: Three nuances of Curry-Howard-Lambek (CHL). There are, of course, other points of view such as logic subsumed by computation or the converse. Since these point of view does not seem to have serious foundations, I decided to exclude them.

logical? If we consider an intersection, it looks like we already assumed a sharp distinction between logic and computation in the first place.

- While the two first points of view are like atheism and theism, the third point of view illustrated in Figure 24.1c is more nuanced but still unclear about what logic and computation are. We know that there are two notions that we constructed and, depending on the point of view, we have different ways to relate them (corresponding to the three points of view given in the introduction of this chapter).

§24.3 I personally believe that suggesting a sharp and refined understanding of both logic and computation is *essential* in order to progress in the right direction for two reasons:

- in the computational world, there exists some barriers, especially in complexity theory where there are great difficulties at separating complexity classes. Those difficulties may be related to a lack of understanding of the nature of logic and computation. In particular, logic already has a role as a constraint (implicit computational complexity) or descriptor (descriptive complexity) of complexity classes;
- in the logical world, there are philosophical motivations in the understanding of the nature of reasoning but also of the relationship between thought, language and reality. In particular, computation (which can be thought as more concrete and down-to-earth) seems to be able to reach an explanation and a justification for reasoning.

§24.4 In this thesis, I defend another understanding of the relation between logic and computation which is closer to the point of view in Figure 24.1c while being consistent with the two previous points of view as well. According to me:

- computation may be about time/dynamics and logic about space/shape (logic regulates the space in which computation flows);
- models of computation are "logical" (probably a very controversial statement);
- logic and computation are completely distinct entities but which are strongly related and so densely mixed that we naturally work with logico-computational hybrid objects most of the time;
- CHL and proof search are two instances of such intertwined mix between logic and computation. In particular, they may both live in the same space;
- the computation-as-model subsumes the two kinds of computation-as-deduction;
- it is possible to separate the logical part from the computational part of a hybrid logico-computational entity.

§24.5 This point of view is based on Jean-Yves Girard's transcendental syntax programme. In order to explain what it is, we need to first start from linear logic and its developments. My point of view on logic and computation changed many times during my PhD thesis and this is what I would like to share in this dissertation. However, I do not have Girard's

confidence nor his pretension so I would like to make clear that I am fully aware that it is only a suggestion that I find interesting and not the *final solution*. There are probably still a lot of blind spots in my understanding of logic. What I suggest are mostly potential directions and inspirations.

Chapter 4

Linear logic

Linear logic has been introduced by Jean-Yves Girard [Gir87a]. I will not present the history of linear logic in details as I do not know it very well and I do not think it is very important for this thesis. Hence, I will simply present a short summary and introduce linear logic directly from the sequent calculus. Historical notes can be found in Girard's seminal paper [Gir87a, Section V].

25 The emergence of linear logic

§25.1 During the rise of the Curry-Howard-Lambek correspondence, there was interests in the denotational semantics of λ-calculus where λ-terms enjoy a mathematical interpretation defining their meaning. If Λ is the set of all λ-terms then we need a space D of denotations (mathematical meaning) and an interpretation function $[\![\cdot]\!] : \Lambda \to D$. Now, if λ-terms are interpreted by functions $D \to D$, we have the usual paradox of set theory since it would lead to $D \simeq D \to D$. We must then restrict the space of functions. Dana Scott's [Sco82] solution was to use domains and continuous function.

§25.2 Girard (who was working on functional interpretations and cut-elimination for second-order arithmetic) independently worked on alternative semantics by choosing category theory to interpret λ-terms by *normal functors* [Gir88]. This is in this context that a decomposition of the intuitionistic disjunction $A \vee B := !A \oplus !B$ was discovered. It is only later that this decomposition has been translated into simplified interpretations such as Girard's *coherence semantics* [Gir87a, Section 3] and a decomposition of intuitionistic implication $A \Rightarrow B := !A \multimap B$ has been introduced. At that time, two types of semantics for the λ-calculus were studied:

- *qualitative* semantics studying the relationship between input and output (domains, Scott semantics, coherence spaces);
- *quantitative* semantics such as Girard's normal functors which takes into account occurrences and how much a portion of data of the output is used in the output. It can eventually feature coefficients which are useful for probabilistic considerations.

§25.3 Although linear logic and all its main developments are usually attributed to Girard, several prefigurations or independent (accidental) definitions of (parts of) linear logic deserve to be mentioned:

- the modal logic S4 reasoning on possibility and necessity (introduced by Clarence Irving Lewis) [FY19];
- Lambek calculus studying the syntax of natural languages (introduced by Joachim Lambek) [Pen92];
- $*$-autonomous (star autonomous) categories (introduced by Michael Barr) [Bar91].

26 Seizing the means of production

§26.1 The removal of logical principles can teach a lot and sometimes even adds more than the design of new rules. In intuitionistic logic, the removal of classical principles (either classical rule or constraining the space of conclusions) adds a straightforward constructivity and a direct connexion with typed λ-calculus. Structural rules (contraction and weakening) have already been mentioned in Paragraph 9.10. They make provability and truth *eternal* because we can do these operations as much as we want as if we had an infinite supply of formulas. But what happens if we forbid these rules and stop this machine of unlimited truth?

§26.2 **Substructural logics.** The removal of structural rules yields several interesting logical systems called *substructural logics* [Res02]:

- (strict or strictly) *linear logic* which forbids all structural rules and formulas are seen as limited resources. In particular, the implication $\Rightarrow$ becomes the linear implication $\multimap$ which uses its argument exactly once. It is *linear* for several reasons but a simple explanation is that linear implication represent functions *linearly* consuming their input (n inputs induce n consumptions);
- *affine logic* where the affine implication $\multimap$ uses its argument at most once, hence weakening is allowed on the hypothesis. It means that it is fine to erase formulas but we cannot have more than what we already have;
- *relevance logic* which requires a *real causality* between the hypothesis and the conclusion: weakening is not possible on the hypothesis but contraction is. We cannot have $A \wedge B \multimap B$ because B is not a direct consequence of A.

§26.3 If we apply this to the typing of λ-calculus, then strict linear logic is a type system for λ-terms with abstractions $\lambda a.t$ where a occurs only once in t. The β-reduction $(\lambda a.t)u \rightsquigarrow \{a := u\}t$ only replaces a unique occurrence of a by u. Among λ-abstractions which are already linear, we can find $\lambda a.a$ and, if we allow pairs: $\lambda a.\lambda b.(a, b)$ and $\lambda a.\lambda b.(b, a)$. As for affine logic, we have the two terms $\lambda a.\lambda b.a$ and $\lambda a.\lambda b.b$ because they erase one

argument. As far as I know, relevance logic is not as studied as linear and affine logic in the context of the λ-calculus.

§26.4 **Additive and multiplicative operations.** This chapter will be focussed on linear logic only. Something remarkable happens when we forbid all logical rules. As stated in Paragraph 10.2, multiplicative and additive rules[1] of linear logic are equivalent but it is only thanks to structural rules. If we forbid structural rules then additive and multiplicative rules are *different operations* giving rise to different connectives. In particular, in Figure 10.3, in a bottom-up reading, $\wedge^m$ splits its context and $\wedge^a$ shares it. These two rules are two different ways of handling formulas seen as resources and they both deserve their own connective.

- For conjunction, $\wedge^a$ yields & (with) and $\wedge^m$ yields $\otimes$ (tensor);
- For disjunction, $\vee^a$ yields $\oplus$ (plus) and $\vee^m$ yields $\parr$ (par).

This decomposition also applies to neutral elements.

- For the truth constant, $\top^a$ yields $\top$ (top) which is neutral for & and $\top^m$ yields 1 (one) which is neutral for $\otimes$;
- For the falsity constant, $\bot^m$ yields $\bot$ (bottom) which is neutral for $\parr$ and we have a new constant 0 (zero) which is neutral for $\oplus$ corresponding to an additive $\bot$ (note that it has no rule).

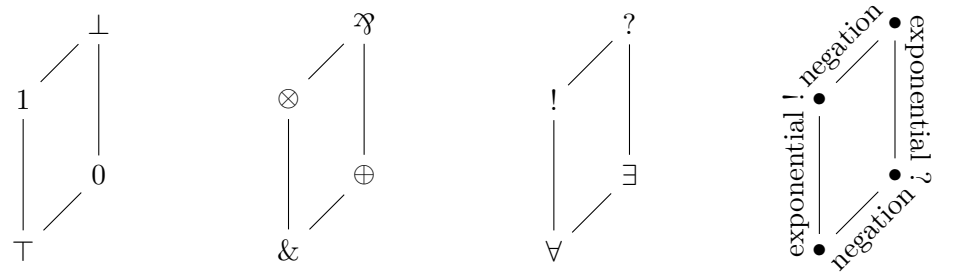
The removal of structural rules *revealed* new connectives underlying the usual logical connectives, as if we opened the clock of logic to look at its mechanisms.

§26.5 **Linear negation.** The linear negation (or orthogonal, or dual) of a formula A is written $A^\bot$ which have several pronunciations depending on your preferences: A orthogonal, A dual, A perp[2]. We have just defined strict linear logic by just taking classical monolateral sequent calculus (*cf.* Section 11) and removing structural rules. Negation is not affected by this removal. Linear negation is still involutive, *i.e.* we have $A \equiv A^{\bot\bot}$ and De Morgan laws still hold for linear logic. In a more general setting (for instance bilateral sequent calculus), linear implication can be defined with $A \multimap B := A^\bot \parr B$. Linear equivalence $A \equiv B$ can be defined with $(A \multimap B) \otimes (B \multimap A)$.

§26.6 **From prohibition to regulation.** Forbidding structural rules is too severe and the logical system we obtain is too weak. We would like to have the power of classical and intuitionistic logical back while keeping the distinct operators of additive and multiplicative rules. In linear logic, it is possible to decompose the intuitionistic implication $A \Rightarrow B$ into $!A \multimap B$ where:

[1] The terminology "additive" and "multiplicative" comes from interpretation of linear logic where formulas are associated with sets: $|A \otimes B| = |A| \times |B|$ and $|A \& B| = |A| + |B|$ (which makes the term "exponential" even more relevant). Such interpretations can be found in introduction to coherent spaces [Gir87a, Section 3] (note that this is not the original presentation of linear logic).

[2] If you say this one, you have very bad taste.



(a) Nullary case. (b) Binary case. (c) Infinitary case. (d) How to read.

Figure 26.1: Relations between linear logic connectives.

- $\multimap$ is a linear implication using its argument exactly once,
- $!A$ (called "of course") corresponds to a potentially infinite supply of A in the world of hypotheses, and
- $?A$ (called "why not") is the negation of ! and corresponds to a potentially infinite demand of A in the world of conclusions.

Although it is not possible to prove $A \multimap (A \otimes A)$ because A is used twice and $\multimap$ is linear, it is possible to prove $!A \multimap (A \otimes A)$[3]. Note that because $!A$ is on the left hand-side of $\multimap$, it becomes $?A^{\perp}$ in monolateral sequent calculus. The connectives ! and ? are called *exponential modalities* or simply *exponentials*. They will be explained later after formally introducing a sequent calculus for linear logic.

§26.7 These exponential modalities are one way among others to reintroduce the infinite potential of provability. However, it is also possible to consider other ways to restrict duplication and erasure to have a finer control over formulas seen as resources. These variant of regulations are also related to computational complexity [Gir98, Laf04].

§26.8 Exponentials ! and ? correspond to infinitary cases for $\otimes$ and $⅋$ respectively. This is illustrated by the two equivalences $!A \equiv !A \otimes !A$ and $?A \equiv ?A ⅋ ?B$. Quantifiers $\forall$ and $\exists$ correspond to infinitary cases for & and $\oplus$ respectively. This is because they are both known to generalise conjunction and disjunction but which one, now that we have several ones? Universal quantification does not feature a "context splitting" so it must be &. As for existential quantification, in a bottom-up reading, its rule chooses a unique candidate for existential witness, which corresponds to the destructive intuitionistic choice of $\oplus$ ($\vee^a$). Does multiplicative quantification make sense? I do not know.

[3] People learning linear logic (including me) tend to think that this was how linear logic was introduced but this is not exact. It seems that the decomposition of intuitionistic disjunction $A \vee B = !A \oplus !B$ has been clearly stated first [Gir87a, Section V.1] and the decomposition of implication has been introduced and propagated for pedagogical purposes [Gir87a, Section IV].

$$A, B = X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \qquad i \in \mathbf{N} \qquad (\mathcal{F}_{\text{MLL}})$$
$$A, B = X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \,\&\, B \mid A \oplus B \qquad i \in \mathbf{N} \qquad (\mathcal{F}_{\text{MALL}})$$
$$A, B = X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid !A \mid ?A \qquad i \in \mathbf{N} \qquad (\mathcal{F}_{\text{MELL}})$$
$$A, B = X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \,\&\, B \mid A \oplus B \mid !A \mid ?A \qquad i \in \mathbf{N} \qquad (\mathcal{F}_{\text{LL}})$$

Figure 27.1: Formulas of linear logic.

§26.9 We obtain the relations between connectives illustrated in Figure 26.1 (taken from Olivier Laurent's course notes[4]). Linear negation relates connectives with:

$$(A \otimes B)^\perp = A^\perp \parr B^\perp \qquad (A \parr B)^\perp = A^\perp \otimes B^\perp \qquad 1^\perp = \bot \qquad \bot^\perp = 1$$
$$(A \,\&\, B)^\perp = A^\perp \oplus B^\perp \qquad (A \oplus B)^\perp = A^\perp \,\&\, B^\perp \qquad \top^\perp = 0 \qquad 0^\perp = \top$$
$$(!A)^\perp = ?A^\perp \qquad (?A)^\perp = !A^\perp \qquad (\forall x.A)^\perp = \exists x.A^\perp \qquad (\exists x.A)^\perp = \forall x.A^\perp$$

and exponentials relate connectives with:

$$!(A \,\&\, B) \equiv !A \otimes !B \qquad ?(A \oplus B) \equiv ?A \parr ?B \qquad !\top \equiv 1 \qquad ?0 \equiv \bot$$

More relations can be found in Lafont's *"Linear Logic Pages"* [Laf99]. The reason exponentials are called exponentials is because they act like exponentials on numbers. If we interpret $\otimes$ by $\times$ and $\&$ by $+$ then, the equivalence $!(A \,\&\, B) \equiv !A \otimes !B$ corresponds to $e^{a+b} = e^a e^b$.

§26.10 **The forgotten case of exchange.** Technically speaking, the exchange rule is a structural rule as well. However it will not be covered at all in this thesis and the exchange rule will always be considered implicitly (it can be made explicit when needed). Removing the exchange rule leads to *non-commutative linear logic* where the order of arguments of binary connectives is relevant. It is not widely studied in linear logic but it has been studied by Abrusci [Abr91, AR99]. It is also related to linguistics where order matters as in Lambek's works on categorical grammars.

27 Classical linear logic sequent calculus

§27.1 Linear logic is usually introduced with several fragments, each having their own characteristics. Some fragments are also studied on their own. The grammar of formulas for the main fragments of linear logic are presented in Figure 27.1. We exclude first-order and second-order quantification because the rules are the same as in classical logic.

[4] https://perso.ens-lyon.fr/olivier.laurent/thdem11.pdf

$$\frac{}{\vdash A, A^\perp}\ \text{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\ \text{cut} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr$$

(a) Multiplicative Linear Logic.

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\ \& \qquad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \oplus A_2}\ \oplus_i$$

(b) Additive rules.

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ \text{w} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ \text{d} \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ \text{c} \qquad \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\ !$$

(c) Exponential rules.

$$\frac{}{\vdash 1}\ 1 \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ \bot \qquad \frac{}{\vdash \Gamma, \top}\ \top$$

(d) Neutral elements.

Figure 27.2: Full Linear Logic (LL)

- Multiplicative linear logic (MLL) is the simplest fragment and treats the most elementary operations of reasoning such as the linear implication $\multimap$.
- Multiplicative-Additive linear logic (MALL) is an extension of MLL with operations managing choices during the process of proving. Purely additive linear logic (ALL) is sometimes studied by itself as well.
- Multiplicative-Exponential linear logic (MELL) is an extension of MLL which allows the duplication and erasure of formulas seen as resources.

Full linear logic (LL) corresponds to a system allowing all these principles. For a fragment C of linear logic, we will write C_u for the extension of C with units (neutral elements). For instance, MLL with units is MLL_u and its set of formulas is $\mathcal{F}_{\text{MLL}_u}$.

Multiplicative fragment

§27.2 The sequent rules of MLL are presented in Figure 27.2a. MLL reveal a very simple symmetry and interaction in logic but also in computation. The relation with programming is nicely explained by Curien in his article *"Symmetry and interactivity in programming"* [Cur03]. If A is seen as a supply of limited resource then $A^\perp$ can be interpreted as a *demand* of this resource. The axiom rule can then be interpreted as meeting offer and demand: we consume what we have. It is very similar to the supply/demand of classical monolateral sequent calculus (*cf.* Section 11) except that in linear logic, resources are *limited* and *consumed*.

$$\dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \qquad \vdash A, \Gamma}{\vdash A, \Gamma}\,\text{cut} \quad \rightsquigarrow \quad \vdash A, \Gamma$$

$$\dfrac{\dfrac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,\otimes \qquad \dfrac{\vdash \Xi, A^\perp, B^\perp}{\vdash \Xi, A^\perp \parr B^\perp}\,\parr}{\vdash \Gamma, \Delta, \Xi}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\vdash \Gamma, A \qquad \dfrac{\vdash \Delta, B \qquad \vdash \Xi, A^\perp, B^\perp}{\vdash \Delta, \Xi, A^\perp}\,\text{cut}}{\vdash \Gamma, \Delta, \Xi}\,\text{cut}$$

(a) Multiplicative fragment. Just a rewiring.

$$\dfrac{\dfrac{\vdash \Gamma, A_1 \qquad \vdash \Gamma, A_2}{\vdash \Gamma, A_1 \,\&\, A_2}\,\& \qquad \dfrac{\vdash \Delta, A_k^\perp}{\vdash \Delta, A_1^\perp \oplus A_2^\perp}\,\oplus_k}{\vdash \Gamma, \Delta}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\vdash \Gamma, A_k \qquad \vdash \Delta, A_k^\perp}{\vdash \Gamma, \Delta}\,\text{cut}$$

(b) Additive fragment. Allows choices.

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash \Delta}{\vdash \Delta, ?A^\perp}\,\text{w}}{\vdash ?\Gamma, \Delta}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\vdash \Delta}{\vdash ?\Gamma, \Delta}\,\text{w}$$

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash \Delta, A^\perp}{\vdash \Delta, ?A^\perp}\,\text{d}}{\vdash ?\Gamma, \Delta}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\vdash ?\Gamma, A \qquad \vdash \Delta, A^\perp}{\vdash ?\Gamma, \Delta}\,\text{cut}$$

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash \Delta, ?A^\perp, ?A^\perp}{\vdash \Delta, ?A^\perp}\,\text{c}}{\vdash ?\Gamma, \Delta}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \vdash \Delta, ?A^\perp, ?A^\perp}{\vdash ?\Gamma, \Delta, ?A^\perp}\,\text{cut}}{\dfrac{\vdash ?\Gamma, ?\Gamma, \Delta}{\vdash ?\Gamma, \Delta}\,\text{c}}\,\text{cut}$$

$$\dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \dfrac{\vdash ?\Delta, ?A^\perp, B}{\vdash ?\Delta, ?A^\perp, !B}\,!}{\vdash ?\Gamma, ?\Delta, !B}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \vdash ?\Delta, ?A^\perp, B}{\vdash ?\Gamma, ?\Delta, B}\,\text{cut}}{\vdash ?\Gamma, ?\Delta, !B}\,!$$

(c) Exponential fragment. Applies operations (duplication, erasure, linearisation) on a box containing A.

$$\dfrac{\dfrac{}{\vdash 1}\,1 \qquad \dfrac{\vdash \Gamma}{\vdash \Gamma, \bot}\,\bot}{\vdash \Gamma}\,\text{cut} \quad \rightsquigarrow \quad \vdash \Gamma$$

(d) Neutral elements. Please tell me what it means because I do not know.

Figure 27.3: Main cut-elimination cases for linear logic.

$$\dfrac{\dfrac{\dfrac{}{\vdash X_1, X_1^\perp}\,\text{ax} \quad \dfrac{}{\vdash X_2, X_2^\perp}\,\text{ax}}{\vdash X_1 \otimes X_2, X_1^\perp, X_2^\perp}\,\otimes}{\vdash X_1 \otimes X_2, X_1^\perp \parr X_2^\perp}\,\parr \qquad \dfrac{\dfrac{\dfrac{}{\vdash X_1^\perp, X_1}\,\text{ax}}{\vdash X_1^\perp \parr X_1}\,\parr \quad \dfrac{\dfrac{}{\vdash X_1, X_1^\perp}\,\text{ax} \quad \dfrac{}{\vdash X_1, X_1^\perp}\,\text{ax}}{\vdash X_1^\perp, X_1 \otimes X_1^\perp, X_1}\,\otimes}{\vdash X_1^\perp, X_1}\,\text{cut}$$

Figure 27.4: Examples of proofs in multiplicative linear logic.

§27.3 Cut-elimination cases are presented in Figure 27.3. They are the same as for classical sequent calculus and also enjoy the same computational interpretation. The cut-elimination between $\parr$ and $\otimes$ is simply a rewiring of interaction (the left premises and right premises are connected together).

§27.4 MLL alone is sufficient as a linear type systems for λ-calculus as it is able to express the linear implication $A \multimap B = A^\perp \parr B$. In particular, the linear modus ponens is purely multiplicative. But if we do not limit types to implications, then it is also possible to interpret pairs $(a, b) : X_1 \otimes X_2$. Even though the usual encoding of booleans are the affine terms $\lambda a.\lambda b.a$ and $\lambda a.\lambda b.b$, it is possible to define *linear booleans* with $\lambda a.\lambda b.(a, b) : X_1 \multimap X_1 \multimap X_1 \otimes X_1$ and $\lambda a.\lambda b.(b, a) : X_1 \multimap X_1 \multimap X_1 \otimes X_1$. Linear booleans are used to relate boolean circuits and proofs of MLL [Ter04].

§27.5 **Note on booleans.** The difference between the usual affine booleans and linear booleans is that boolean in MLL cannot be erased. Moreover, to keep computation linear, we have to keep some *garbage*, *i.e.* useless parts which will not be erased but ensure that we are in a linear world. The advantage is that linear booleans enjoy a more convenient interpretation of exclusive disjunction (XOR) [MT15, MT03]. Recently, Lê Thành Dũng Nguyễn (aka "Tito") also introduced a non-commutative linear type for booleans [Ngu21, Section 7.2].

§27.6 An example of proof in MLL is given in Figure 27.4. The first proof is a proof of $(X_1 \otimes X_2) \multimap (X_1 \otimes X_2)$ which can be seen as the linear λ-term $\lambda(a, b).(a, b)$. The second proof is more subtle. It corresponds to a type derivation for the typing sequent $b : X_1 \vdash (\lambda a.a)b : X_1$ where b is a free variable of type X_1 (which becomes $X_1^\perp$ by duality of monolateral sequent calculus). The left branch of the cut represents the function $(\lambda a.a) : X_1 \multimap X_1 = X_1^\perp \parr X_1 \equiv X_1 \parr X_1^\perp$. The connective $\otimes$ connects the argument $b : X_1^\perp$ together with an output of type X_1.

§27.7 **Non-idempotent intersection types.** There exists alternative ways to type terms. Another way is given by intersection types which type terms with conjunction of types $A \wedge B$ called *intersection types* [DC18]. The difference is that, instead of the computational interpretation with pairs, the expression $\vdash M : A \wedge B$ says that M has two types or two behaviours. For instance, the term $\lambda x.xx$ is not typable with the usual simple types because x is used both as a function $x : \alpha \to \beta$ but also as an argument

$x : \alpha$ of a function of type $\alpha \to \beta$. However, it cannot have both types since $\alpha \to \beta$ cannot be unified with α. Nonetheless, it can be typed in intersection type systems with $\vdash \lambda x.xx : \alpha \wedge (\alpha \to \alpha) \to \alpha$ taking into account the two behaviours of x. In the non-idempotent case, we have that $A \wedge A$ is not equivalent to A, hence intersections are seen as collections of resources, exactly as in linear logic. Actually, $\wedge$ corresponds to $\otimes$ in this case. The advantage of non-idempotent types systems is that they can capture strong normalisation (termination) of terms but consequently, type checking becomes undecidable (otherwise we would be able to solve the Halting problem described in Paragraph 16.14). Intersection types can also be used for considerations on computational complexity as in Mazza's works [Maz17, Section 3.3.5].

Neutral elements

§27.8 The rules for neutral elements (also called units) are presented in Figure 27.2d. The constants 1 and $\top$ are two sort of trivial statements (one without context and the other possibly within a context). The constant $\bot$ is an "acceptable error" which can be discarded. It represents an unprovable goal but other paths can be taken in Γ. Remark that if $\Gamma = \emptyset$ then we can be stuck with $\bot$. The constant 0 has no rule and represents a "fatal error".

§27.9 Cut-elimination for neutral elements is presented in Figure 27.3d. Because 0 is a fatal error, it cannot interact by cut-elimination. There is a rule for the interaction between 1 and $\bot$ but I do not know how to give an interesting interpretation of it so you can just look at it without comment.

Additive fragment

§27.10 The additive rules are presented in Figure 27.2b. As in monolateral sequent calculus, they express binary non-deterministic choices. The formula $A \,\&\, B$ can be understood as a *superposition* (passive choice) between a proof of A and a proof of B requiring the same context Γ and proofs of $A \oplus B$ are binary selectors (active choice).

§27.11 In MALL, it is possible to have another encoding of booleans which is closer to usual (functional) programming. A boolean is either a constant True or a constant False, which can be represent by the type $\texttt{Bool} := 1 \oplus 1$. The proofs of `Bool` are either a left selection (representing True) or a right selection (representing False). The constants True and False are hence represented by proofs, accordingly to the CHL correspondence. They should not be confused with constants $0, 1, \top, \bot$ which are types/formulas and not proofs. False is an object which can be used (for instance in conditions) whereas $\bot$ and 0 are types/formulas representing two sort of errors.

$$\dfrac{\dfrac{\vdash \Gamma, A, \bot \quad \vdash \Gamma, A, \bot}{\vdash \Gamma, A, \bot \,\&\, \bot}\,\&\quad \dfrac{\vdash 1}{\vdash 1 \oplus 1}\,\oplus_1}{\vdash \Gamma, A}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\vdash \Gamma, A, \bot \quad \vdash 1}{\vdash \Gamma, A}\,\text{cut} \quad \rightsquigarrow \quad \vdash \Gamma, A$$

Figure 27.5: Example of proof using additive connectives. It represents the use of a condition. The formula A represents the output of the condition. The point is that the two branches of the rule & can be two different proofs (terms) of the same sequent.

§27.12 If we look at cut-elimination for additive connectives in Figure 27.3b, we can see that depending on whether the proof of $A_1^\perp \oplus A_2^\perp$ is a left ($\oplus_1$) or right selection ($\oplus_2$), it will only select one side of & and erase the other. Conditions can be represented with the connective &. However, the correspondence is not so exact. Consider a condition `if` b `then` x `else` y applied to `true`. The result should be x. This situation is represented by a cut-elimination between the condition and the boolean value. However, cut-elimination produces a new cut between the boolean value and the selected branch. If we assume that the two branches are of type A_1 and A_2, the result our interaction cannot be of type A_1 because it would disappear after its interaction with $A_1^\perp$. Only the contexts Γ and Δ remain. Therefore, it we would like to encode conditions, the connective & must link two occurrences of $\bot$ artificially added in the encoding of the condition, so that it annihilates the constant 1 of the encoding of booleans. This is illustrated in Figure 27.5. Remark that the & connective now links those two occurrences of $\bot$ instead of the terms of the two branches of the condition. The condition must then have two branches of same type A.

§27.13 **Comment on linear equivalence.** In Paragraph 26.5, we defined the linear equivalence $A \equiv B$ as $(A \multimap B) \otimes (B \multimap A)$. However, there is an alternative definition with $(A \multimap B) \,\&\, (B \multimap A)$. In a lot of cases, these definitions are equivalent because the rule for $\otimes$ and & are equivalent in an empty context. They both prove either $\vdash A \,\&\, B$ or $\vdash A \otimes B$ from $\vdash A$ and $\vdash B$. There is no problem when equivalence is considered *alone* (for formulas $A \equiv B$ for any A and B). However, when equivalence is negated (typically when it appears on the left of a linear implication), then depending on the chosen definition, we obtain either $(A \multimap B)^\perp \oplus (B \multimap A)^\perp$ or $(A \multimap B)^\perp \parr (B \multimap A)^\perp$ which do not have the same behaviour since $\parr$ keep premises in the same space of interaction and $\oplus$ makes an exclusive choice. For instance, if we consider the transitivity of equivalence $(A \equiv B) \otimes (B \equiv C) \multimap (A \equiv C)$, it is provable for $\otimes$ but not for &. This example is illustrated in Figure 27.6.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \quad \dfrac{\dfrac{}{\vdash B, B^\perp}\,\text{ax} \quad \dfrac{}{\vdash C, C^\perp}\,\text{ax}}{\vdash B^\perp, B \otimes C^\perp, C}\,\otimes}{\vdash A \otimes B^\perp, B \otimes C^\perp, A^\perp, C}\,\otimes}{\vdash A \otimes B^\perp, B \otimes C^\perp, A^\perp \parr C}\,\parr \qquad \dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \quad \dfrac{\dfrac{}{\vdash B, B^\perp}\,\text{ax} \quad \dfrac{}{\vdash C, C^\perp}\,\text{ax}}{\vdash B, C \otimes B^\perp, C^\perp}\,\otimes}{\vdash B \otimes A^\perp, C \otimes B^\perp, C^\perp, A}\,\otimes}{\vdash B \otimes A^\perp, C \otimes B^\perp, C^\perp \parr A}\,\parr}{\vdash A \otimes B^\perp, B \otimes A^\perp, B \otimes C^\perp, C \otimes B^\perp, A \equiv C}\,\parr$$

$$\dfrac{\vdash A \not\equiv B, B \not\equiv C, A \equiv C}{\vdash (A \not\equiv B) \parr (B \not\equiv C) \parr (A \equiv B)}\,\parr$$

(a) Linear equivalence $A \equiv B$ defined by $(A \multimap B) \otimes (B \multimap A)$. Nonequivalence $A \not\equiv B$ is $(A \otimes B^\perp) \parr (B \otimes A^\perp)$.

$$\dfrac{\dfrac{?}{\vdash (A \otimes B^\perp) \oplus (B \otimes A^\perp), (B \otimes C^\perp) \oplus (C \otimes B^\perp), A \equiv C}\,?}{\vdash (A \not\equiv B) \parr (B \not\equiv C) \parr (A \equiv B)}\,\parr$$

(b) Linear equivalence $A \equiv B$ defined by $(A \multimap B) \,\&\, (B \multimap A)$. Nonequivalence $A \not\equiv B$ is $(A \otimes B^\perp) \oplus (B \otimes A^\perp)$. We have two choices, either we use $\otimes$ and split with $A \multimap C$ and $C \multimap A$. But no matter how we distribute the context, we will never be able to join A and C. The other choice is to use the rule $\oplus_k$ but by doing so, we lose precious information. The sequent is unprovable. It would have worked if we could duplicate the context so as not to lose information.

Figure 27.6: We give two proofs of the transitivity of linear equivalence: $(A \equiv B) \otimes (B \equiv C) \multimap (A \equiv C)$ with different definitions of equivalence.

Exponential fragment

§27.14 The rules for exponentials are presented in Figure 27.2c. Only connectives prefixed by the connective "?" can be duplicated or erased. The rule "d" is called *dereliction*[5]. From a bottom-up reading, it corresponds to linearising a formula so that duplication and erasure does not apply any more. From a top-down reading, it makes a linear formula non-linear. The most mysterious rule is the rule "!" called *promotion*. This rule *wraps* a linear formula with the connective !, but this is only possible when the context is fully non-linear. This connective is better understood when looking at cut-elimination.

§27.15 The cut-elimination steps for exponentials are presented in Figure 27.3c. I know they look scary but they are actually very intuitive. The rule ! on $!A$ constructs a sort of box encapsulating A in a non-linear context. This box can then interact with a non-linear formula:

- in a $w/!$ interaction, the box is erased;
- in a $d/!$ interaction, the box is opened and its content A interact with the linearised formula $A^{\perp}$;
- in a $c/!$ interaction, the box is duplicated[6];
- in a $!/!$ interaction, what happens is more spectacular. This is a commutation case where two ! are related but do not interact directly. One wishes to interact with something deeper in the proof. For that reason, one box *enters* inside the other one.

§27.16 **Intuitionistic and classical translations.** Intuitionistic logic can be defined from linear logic. There are two translations introduced by Girard.

- $[\![X_i]\!] = X_i$ and $[\![A \Rightarrow B]\!] = ![\![A]\!] \multimap [\![B]\!]$ (the original translation which gave rise to linear logic). Non-linearity applies on the argument;
- $[\![X_i]\!] = !X$ and $!([\![A]\!] \multimap [\![B]\!])$. Non-linearity applies on the whole implication/function.

It is possible to define a translation for classical logic by choosing the right combinations of ! and ? to wrap formulas (which are known as q-translation and t-translation for the systems LKQ and LKT [DJS95]) or using the formalism of *polarised linear logic* [LR03].

[5] According to the dictionary it means "failing to do what you should do". In French, it is a literary word referring to a state of moral abandon or loneliness.

[6] Because of this duplication, we can think that cut-elimination will not terminate but it actually does. It is sufficient to find a measure which decreases during cut-elimination. Methods for proof of termination can be found in the literature [BN98, Chapter 5].

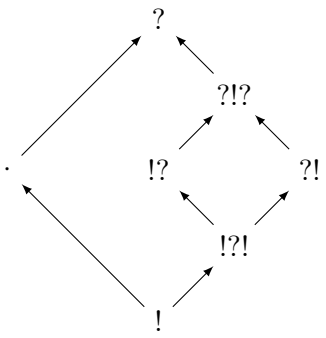


Figure 27.7: The lattice of exponentials modalities. The dot · represents the absence of modality.

§27.17 Exponentials can be used to reconstruct a type system for simply typed λ-calculus (since linear logic decomposes intuitionistic logic). A type assertion $\Gamma \vdash M : A$ becomes $\vdash ?\Gamma^{\perp}, A$. The idea is that boxes represent arguments (the N in $(\lambda x.M)N$). When applying the β-reduction which replaces all x in M by N, several types of operation can be done. The term N can be erased (if x does not appear in M) and it can also be duplicated (if several occurrences of x appear in M). It may also be possible to imagine other logical operations by thinking about what computational operation we would like to apply on an argument. By using λ-calculus, it is also possible to give an interpretation to the two intuitionistic translations. They correspond to two *evaluation strategies* for β-reduction known as "call-by-name" and "call-by-value" evaluation [MOTW99, Wad03]. The first evaluates functions first and the second the arguments first.

§27.18 **Hybrid calculi for linear logic.** The interpretation of simply typed λ-calculus corresponds to a restricted (intuitionistic) subset of MELL where we only consider non-linearity but it is also possible to construct a λ-calculus mixing both linearity and non-linearity [ABCJ98, ACJ97]. It is also possible to find a presentation and several extensions in Mazza's HdR thesis [Maz17, Section 1.1.2].

§27.19 **The lattice of linear modalities.** Not all combinations of exponential connectives such as $!!!A$ or $?!??A$ can be distinguished. If the formula $!A$ is interpreted as a potentially infinite supply of A then it is not so surprising that we have $!!A \equiv !A$ (this is proven in Figure 27.8a). A combination of ! and ? is called a *modality*. In his PhD thesis, Joinet remarks that for any modalities μ, we have $!A \multimap \mu A$ and $\mu A \multimap ?A$ [Joi93, Section 5.2]. This induces an order over modalities. From that, he deduces a lattice of exponential modalities and distinguishes 7 equivalence classes presented in Figure 27.7. As a corollary, we have that all modalities are idempotent: for any modality μ, $\mu A \equiv \mu\mu A$. This has later been presented by Schellinx as well in his PhD thesis [Sch94, Section 1.1]. Similar facts also appeared independently in modal logic (especially S4) [FY19].

§27.20 Examples of proofs in MELL are presented in Figure 27.8. The third example in Fig-

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax}}{\vdash A^\perp, A}\,\text{d}}{\vdash \mathbf{??}\,A^\perp, A}\,!}{\vdash ??A^\perp, \mathbf{!}\,A}\,\parr \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax}}{\vdash A^\perp, A}\,\text{d}}{\vdash \mathbf{?}\,A^\perp, A}\,!}{\vdash ?A^\perp, \mathbf{!!}\,A}\,\parr}{\vdash ??A^\perp \parr !A}\,\otimes}{\vdash !!A \equiv !A}$$

(a) Proof that limitlessness of limitlessness is limitlessness.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax}}{\vdash A^\perp \oplus B^\perp, A}\,\oplus_1}{\vdash \mathbf{?}\,(A^\perp \oplus B^\perp), A}\,\text{d}}{\vdash ?(A^\perp \oplus B^\perp), \mathbf{!}\,A}\,! \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\vdash A^\perp \oplus B^\perp, B}\,\oplus_2}{\vdash \mathbf{?}\,(A^\perp \oplus B^\perp), B}\,\text{d}}{\vdash ?(A^\perp \oplus B^\perp), \mathbf{!}\,B}\,!}{\vdash ?(A^\perp \oplus B^\perp), ?(A^\perp \oplus B^\perp), !A \otimes !B}\,\otimes}{\vdash ?(A^\perp \oplus B^\perp), !A \otimes !B}\,\text{c}}{\vdash \mathbf{?}\,(A^\perp \oplus B^\perp), !A \otimes !B}\,\parr}{\vdash ?(A^\perp \oplus B^\perp) \parr (!A \otimes !B)}\,\parr \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax}}{\vdash \mathbf{?}\,A^\perp, A}\,\text{d}}{\vdash ?A^\perp, \mathbf{?}\,B^\perp, A}\,\text{w} \qquad \dfrac{\dfrac{\dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\vdash \mathbf{?}\,B^\perp, B}\,\text{d}}{\vdash \mathbf{?}\,A^\perp, ?B^\perp, B}\,\text{w}}{\vdash ?A^\perp, ?B^\perp, A \,\&\, B}\,\&}{\vdash ?A^\perp, ?B^\perp, \mathbf{!}\,(A \,\&\, B)}\,!}{\vdash ?A^\perp \parr ?B^\perp, !(A \,\&\, B)}\,\parr}{\vdash (?A^\perp \parr ?B^\perp) \parr !(A \,\&\, B)}\,\parr}{\vdash !(A \,\&\, B) \equiv !A \otimes !B}\,\otimes$$

(b) Proof that exponentials link the additive and multiplicative conjunction.

$$\dfrac{\dfrac{\dfrac{}{\vdash X_1^\perp, X_1}\,\text{ax}}{\vdash X_1^\perp \parr X_1}\,\parr \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash X_1, X_1^\perp}\,\text{ax}}{\vdash X_1, ?X_1^\perp}\,\text{d}}{\vdash !X_1, ?X_1^\perp}\,! \qquad \dfrac{}{\vdash X_1, X_1^\perp}\,\text{ax}}{\vdash ?X_1^\perp, !X_1 \otimes X_1^\perp, X_1}\,\otimes}{\vdash ?X_1^\perp, X_1}\,\text{cut}$$

(c) Type derivation for $(\lambda x.x)y$ in MELL.

Figure 27.8: Examples of proofs in multiplicative-exponential linear logic.

$$\frac{\vdash \Gamma, ??A}{\vdash \Gamma, ?A}\ \text{dig} \qquad \frac{\vdash \Gamma, A}{\vdash ?\Gamma, !A}\ f! \qquad \frac{\dfrac{\vdash ?\Gamma, A}{\vdash ??\Gamma, !A}\ f!}{\vdash ?\Gamma, !A}\ \text{dig}$$

Figure 27.9: Rules for digging and functorial promotion (also called weak promotion). A proof that promotion can be derived from digging and functorial box is given.

$$\frac{\vdash \Gamma, A, \overset{n}{\ldots}, A}{\vdash \Gamma, ?A}\ \text{mux(n)} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ \text{mux(1)} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ \text{mux(0)} \qquad \frac{\dfrac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ??A}\ \text{mux(2)}}{\vdash \Gamma, ?A}\ \text{dig}$$

Figure 27.10: Rules for multiplexing and derivation of other structural rules.

ure 27.8c is the exponential version of the type derivation of the linear term $(\lambda a.a)b$ in Figure 27.4. Remark that the argument is put in a box. If we had a contraction or weakening on the side of the function ($\parr$ rule on the left), then it would trigger a duplication or erasure of the argument inside the box (! rule on the right).

§27.21 **Alternative exponentials.** Several variant of exponential rules exist. For instance, the digging and functorial promotion rules in Figure 27.9 can replace the promotion rule. These rules are useful for categorical interpretations or in implicit computational complexity using linear logic (especially light linear logics [Gir98] such as ELL and SLL). It is also possible to consider a *multiplexing* rule splitting $?A$ by duplicating it into n linearised copies. We obtain the following equivalences of rules:

- promotion $\simeq$ functorial promotion + digging;
- dereliction + weakening $\simeq$ multiplexing.

28 Some applications and intuitive interpretations

§28.1 **Logic of resources.** The most straightforward interpretation is that linear logic is a logic of limited resources and control over resources. It seems that it has generated a quite important hype of "linear typing" in (functional) programming [Wad90] with implementations in Haskell or Rust. It is consistent with functional languages' philosophy of strong typing[7] where we would like to prevent unwanted behaviours of a program.

[7] I remember that during a course of compilers, Yann-Régis Gianas told us that there was a sort of duality between weakly typed languages such as Python who consider that programmers know what they are doing and should be free to do so (for instance condition returning objects of different types in the two branches) and strongly typed languages such as OCaml where programmers are considered dumb so the system limits what they can do so that they will not unintentionally break something

Linear types can ensure that some data will never be duplicated or erased, and they could be used to represent "read-only" operations which can guarantee the safety of data. Typically, money is also a limited resource which can be regulated. It may be possible to use linear types to verify economic operations, as suggested by some vague applications to blockchain technologies [Mer15].

§28.2 Still in this interpretation of a logic of resources, biology, chemistry and physics also work with consumable entities. The chemical reactions mentioned in Paragraph 12.10 are linear. In the chemical equation $2H_2 + O_2 \mapsto 2H_2O$, the relation $\mapsto$ can be rewritten with $\multimap$ to signify that the same amount of input resources is found in the output: nothing is lost, nothing is created. However I am not aware of important works in that direction. As for biology, there exists few works relating biological systems and linear logic systems [dMDF+20, DFLO19, Des16].

§28.3 **Logic and games.** Logic has been related to the idea of game in several ways by several authors such as Gentzen, Lorenzen (in his *"Logik und Agon"*) and Hintikka. Proving a statement can be seen as a dialogue between a prover and someone to convince. Hintikka imagines a language game between us and the "nature" [HS97]. This dialogue becomes even more explicit with linear logic.

- The connective & can be interpreted as a passive choice from "our side". Its rules shows that we can consume all the resources Γ to produce either A or B and it will works in the two cases. The formula $A \parr B$ is a way to express the fact that we wish to keep A and B in the same space of interaction. Both & and $\parr$ are risk-free moves for "us". They have no influence on provability.
- The connective $\otimes$ is an active choice since we have to choose how to split the context. It is the same for $\oplus$ where we can either choose the left or right premise. However, these choices are not risk-free. We can make wrong choices. In some sense, it is the "nature" who chooses where provability is hidden.

We remark that & and $\parr$ are *reversible*. They are called *negative* (passive) connectives. As for $\oplus$ and $\otimes$, they are *irreversible*. They are called *positive* (active) connectives. These two classes of connectives are related by negation which can be seen as a change of point of view: "we" becomes "they" and vice-versa.

§28.4 Cut-elimination can then be interpreted as actually triggering a dialogue or a sort of battle in a game:

- in a $\parr/\otimes$ cut-elimination, a player ($\parr$) suggests two arguments A and B waiting in its space of interaction as a sort of question and the other player ($\otimes$) creates two threads in order to answer A and B;

somewhere. It was told in the tone of a joke but I think that it was actually very profound (and almost political).

- in a $\&/\oplus$ cut-elimination, a player ($\&$) suggests two options by proposing the other player to either play against $\vdash \Gamma, A$ or $\vdash \Gamma, B$. Then, the other player ($\oplus$) makes a choice.

In the context of linear logic, a *game semantics* has been proposed by several authors but we can cite Blass [Bla92] and Abramsky [AJ94] among others. True/provable formulas will correspond to players having a "winning strategy". In the simple case of axioms, a player of $A^\perp \parr A$ is expected to win against $A \otimes A^\perp$. This interpretation with games makes linear logic a *logic of actions* as well as a logic of resources.

§28.5 In Girard's ludics [Gir01], synthetic rules are considered: there is only one positive rule and one negative rule. In this particular case, dialogue can be infinite and a player wins when the other abandon with a special axiomatic rule called "Daimon" which proves anything. Ludics is used as a tool in the analysis of dialogues as explained in Fouquéré et al. *"Mathématique du dialogue"* (French book) and has been recently applied in the analysis of dialogues with schizophrenic individuals [FPQ21].

§28.6 **Processes and sessions.** An early interpretation was also that linear logic was a *logic of interaction* [Abr16, Chapter 5]. In programming, concurrent programming studies the (usually asynchronous) interaction between independent agents which transmit information locally. Some languages such as the π-calculus [iln90] are dedicated to such systems of communication. Another form of communication are *sessions*. It happens that a (web) client communicate with a server by opening a session. During this session, several messages will be transmitted. At the end the session is closed. It appears that linear logic is able to type processes [Abr94] and sessions [Wad12]. This leads to an extension of the CHL correspondence with proofs as processes and formulas as session/process types. By using realisability theory (*cf.* Section 22), Emmanuel Beffara has suggested attempts at reconstructing linear logic from π-calculus. This realisability interpretation allows to use linear logic as a tool to study the behaviour of processes [Bef06].

§28.7 **Logic programming and linear logic.** In the previous interpretations of linear logic with games, multiplicative and additive linear logic are classified into reversible and irreversible connectives. A consequence is that rules of reversible connectives can always been applied before any irreversible rules. From this fact, Jean-Marc Andreoli [And92] developed the idea of *focussing*. It is possible to consider strategies of proof by alternating between reversible and irreversible rules. This gave rise to several applications of linear logic to logic programming. More information can be found in Dale Miller's survey [Mil95].

29 Proof-structures and proof-nets

§29.1 Until now, only sequent calculus for linear logic has been mentioned. But what about natural deduction? The advantage of natural deduction is that it is a quotient on

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax} \quad \dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\vdash A^\perp, A \otimes B^\perp, B}\,\otimes \quad \dfrac{}{\vdash C^\perp, C}\,\text{ax}}{\vdash A^\perp, A \otimes B^\perp, B \otimes C^\perp, C}\,\otimes}{\vdash A \otimes B^\perp, B \otimes C^\perp, A^\perp \parr C}\,\parr}{\vdash (A \otimes B^\perp) \parr (B \otimes C^\perp), A^\perp \parr C}\,\parr}{\vdash ((A \otimes B^\perp) \parr (B \otimes C^\perp)) \parr (A^\perp \parr C)}\,\parr$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax} \quad \dfrac{\dfrac{}{\vdash B^\perp, B}\,\text{ax} \quad \dfrac{}{\vdash C^\perp, C}\,\text{ax}}{\vdash B^\perp, B \otimes C^\perp, C}\,\otimes}{\vdash A^\perp, A \otimes B^\perp, B \otimes C^\perp, C}\,\otimes}{\vdash A \otimes B^\perp, B \otimes C^\perp, A^\perp \parr C}\,\parr}{\vdash (A \otimes B^\perp) \parr (B \otimes C^\perp), A^\perp \parr C}\,\parr}{\vdash ((A \otimes B^\perp) \parr (B \otimes C^\perp)) \parr (A^\perp \parr C)}\,\parr$$

Figure 29.1: Two sequent calculus proofs of linear logic equivalent up to permutation of rules.

$$\dfrac{\begin{matrix}\vdots & \vdots \\ A & B\end{matrix}}{A \otimes B}\,\otimes i \qquad \dfrac{\begin{matrix}\vdots & [A] & [B] \\ A \otimes B & \vdots & \vdots \\ & C & C\end{matrix}}{C}\,\otimes e$$

Figure 29.2: Natural deduction rules for tensor.

sequent calculus proofs: a single natural deduction proof potentially corresponds to several sequent calculus proofs. In Figure 29.1, we have two proofs of a same sequent. They only differ by the order of $\otimes$ rules. One can be applied before the other without risking to lose provability. The same thing occurs if we permute $\parr$ rules.

§29.2 As remarked by Girard [Gir11a, Section 11.1.5], it is possible to define a natural deduction for linear logic. The introduction and elimination rules for tensor are presented in Figure 29.2. The introduction rule is natural and not surprising. The elimination rule, on the other side, is not sufficient at all. It suffers from the same problem as for the rule $\vee e$ and will lead to complications although tensor is a very basic connective with a simple behaviour. In linear logic, negation is involutive and inputs can be seen as negated formula. Hence, because of the morphologic constraint of natural deduction, $[A]$ and $[B]$ are sort of wannabe negated conclusion formulas but an artificial bending through C appears instead. For that reason, we can do better by choosing a structure of proofs which is a not tree-shaped stacking of rules. We have to find another way to capture the *essence* of proofs by providing a syntactic format in which proofs can express everything that have to say[8].

[8] In the same fashion, Japanese square watermelons are grown in square boxes in order to obtain a square shape. This shape is more convenient to cut or to carry. More generally, you can think of the process of dwarfing in plants or animals. Another (sadder) example that I have learnt recently is that goldfishes growing in a fish bowl are subject to a painful dwarfing.

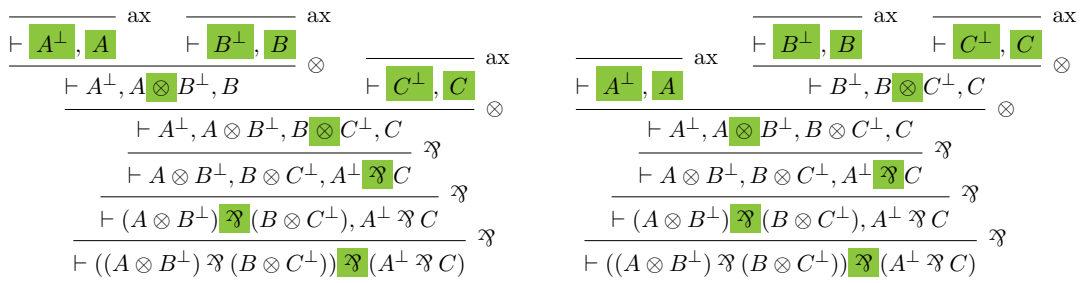


Figure 29.3: Underlying formula graphs of the proofs of Figure 29.1.

$$\dfrac{\dfrac{\vdash \Gamma, A, B, C, D}{\vdash \Gamma, A, B, C \parr D}\parr}{\vdash \Gamma, A \parr B, C \parr D}\parr \quad \leftrightarrow \quad \dfrac{\dfrac{\vdash \Gamma, A, B, C, D}{\vdash \Gamma, A \parr B, C, D}\parr}{\vdash \Gamma, A \parr B, C \parr D}\parr$$

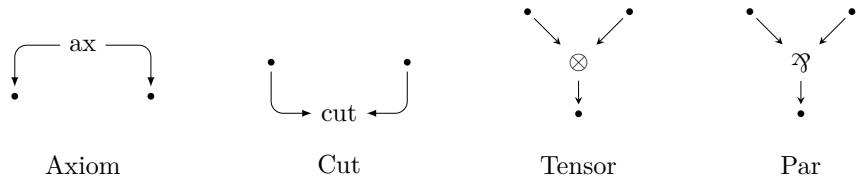
Figure 29.4: Commutation rule for $\parr$. It states the equivalence of two proofs up to permutation of rules.

Multiplicative unit-free proof-nets

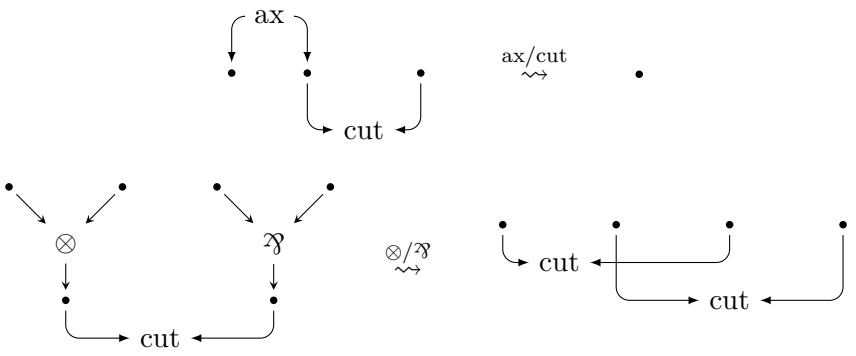
§29.3 **The skeleton of the essence of proofs.** If we look at the two previous sequent calculus proofs of Figure 29.3, we remark that $\parr$ splits two formulas kept in the same space of interaction and $\otimes$ distributes two formulas in disjoint spaces of interaction. By following how rules do these operations, it is possible to construct a graph linking formulas. Each node $A \otimes B$ (respectively $A \parr B$) or the graph are connected to A and B in the top. The two proofs (which are equivalent up to permutation of rules) have the same graph. These graphs are then appropriate candidate for the *essence of proofs* where (almost) all superfluous information is removed. This ability to quotient proofs which are equivalent for wrong reasons is often called *canonicity*. Translation of proofs to such graphs should be invariant up to commutation rules such as the one presented in Figure 29.4. Sequent calculus proofs can then be seen as sequential recipes (algorithms?) to construct those graphs that Girard called a "parallel syntax" for proof theory [Gir96]. Those graphs are called *proof-nets* and they only make explicit how formulas are structurally related in a proof.

§29.4 Before introducing proof-nets, we introduce *proof-structures* which is the format in which proofs will now live[9]. They are purely computational and alogical graphs linking formulas. In the same spirit as Girard's ludics [Gir01], "only location matters" at this point. The idea is that when considering the structure of proofs, formulas are nothing more

[9] If you have read my previous footnote on dwarfing of goldfishes, we now have an aquarium.



(a) Links/constructors of proof-structures as hyperedges.



(b) Cut-elimination reductions. The ax/cut case is a graph contraction and $\otimes/\parr$ is a rewiring.

Figure 29.5: Multiplicative proof-structures.

than decorative labels which can be forgotten. In this syntax, proof-structures are defined with directed hypergraphs constructed with the hyperedges of Figure 29.5a. These hyperedges are the elementary bricks of proofs. Definitions of hypergraphs are formally given in Appendix C.

§29.5 **Definition** (Proof-structure)**.** A *proof-structure* is defined by $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ where $(V, E, \mathtt{in}, \mathtt{out})$ is a ordered directed hypergraph (*cf.* Appendix C) and $\ell_E : E \to \{\otimes, \parr, \mathrm{ax}, \mathrm{cut}\}$ is a labelling map on hyperedges. A proof-structure is subject to these additional constraints:

- hyperedges satisfy the arities and labelling constraints shown in Figure 29.5a;
- each vertex must be the target of exactly one hyperedge, and the source of at most one hyperedge;
- cut hyperedges must connect either:
  - the conclusion of a $\parr$ hyperedge with the conclusion of a $\otimes$ hyperedge, or
  - two atoms.

§29.6 **Notation** (Axioms and cuts)**.** Let $\mathcal{S}$ be a proof-structure. We define its set of axioms hyperedges: $\mathtt{Ax}(\mathcal{S}) := \{e \mid e \in E, \ell_E(e) = \mathrm{ax}\}$, and its set of cut hyperedges: $\mathtt{Cuts}(\mathcal{S}) := \{e \mid e \in E, \ell_E(e) = \mathrm{cut}\}$.

§29.7 **Convention** (Left and right sources)**.** For practical purposes, the sources of hyperedges are ordered, and "left" and "right" sources are considered since there are never more than two; illustrations in Figure 29.5a implicitly represent the left (*resp.* right) source on the left (*resp.* right).

For a proof-structure $\mathcal{S}$ and a hyperedge e of $\mathcal{S}$:

- if $\ell_E(e) = \mathrm{ax}$ and $\mathtt{out}(e) = (u, v)$, we define $\overleftarrow{e} := u$ and $\overrightarrow{e} := v$ for the left and right conclusion of e;
- if $\ell_E(e) \in \{\mathrm{cut}, \otimes, \parr\}$ and $\mathtt{in}(e) = (u, v)$, we define $\overleftarrow{e} := u$ and $\overrightarrow{e} := v$ for the left and right premise of e.

§29.8 **Notation** (Conclusions and atoms)**.** The *conclusions* of $\mathcal{S}$ are defined by the set $\mathtt{Concl}(\mathcal{S}) = \{v \in V \mid \text{there is no } e \in E \text{ such that } v \in \mathtt{in}(e)\}$. Similarly, the *atoms* of $\mathcal{S}$ are defined by the set $\mathtt{Atoms}(\mathcal{S}) = \{v \in V \mid \exists e \in \mathtt{Ax}(\mathcal{S}) \text{ such that } v \in \mathtt{out}(e)\}$. They are conclusions of axiom hyperedges.

§29.9 The cut-elimination procedure (corresponding to program execution) is defined with two graph-rewriting rules in Figure 29.5b. Cut-elimination is made even more natural in proof-structures; the case ax/cut is a graph contraction identifying two atomic formulas and the case $\otimes/\parr$ makes explicit the idea of rewiring. An example of proof-structure and its full cut-elimination is illustrated in Figure 29.6 and a formal definition of cut-elimination is given below.

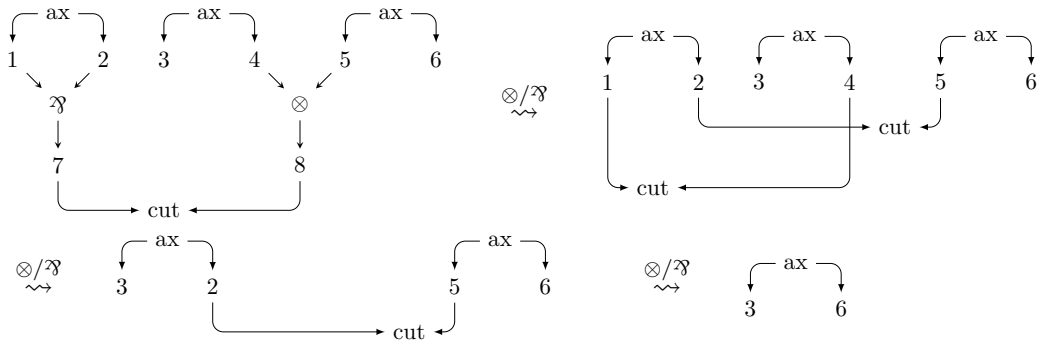


Figure 29.6: Cut-elimination for a proof-structure.

$$\frac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Gamma, \Delta}\ \text{mix} \qquad\qquad \dfrac{\dfrac{\dfrac{}{\vdash X_1^\perp, X_1}\ \text{ax} \qquad \dfrac{}{\vdash X_2^\perp, X_2}\ \text{ax}}{\vdash X_1^\perp, X_2^\perp, X_1, X_2}\ \text{mix}}{\vdash X_1^\perp \parr X_2^\perp, X_1, X_2}\ \parr}{\vdash X_1^\perp \parr X_2^\perp, X_1 \parr X_2}\ \parr$$

Figure 29.7: The MIX rule of MLL+MIX sequent calculus and an example of a sequent calculus proof of $A \otimes B \multimap A \parr B$ which is not provable in MLL.

§29.10 **Definition** (MLL cut-elimination)**.** Let $\mathcal{S} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ be an MLL proof-structure with a cut $e_{\text{cut}} \in E$ such that $\mathtt{in}(e_{\text{cut}}) = (v_1, v_2)$ with both v_1 and v_2 being conclusions of some hyperedges e_1 and e_2.

◇ **Left axiom** If $\ell_E(e_1) = \text{ax}$ for $i \in \{1, 2\}$ and $\mathtt{out}(e_1) = (v_0, v_1)$, then the elimination of e_{cut} is a new proof-structure $\mathcal{S}' := (V', E', \mathtt{in}, \mathtt{out}', \ell_E)$ such that $V' := V - \{v_1, v_2\}$, $E' := E - \{e_{\text{cut}}, e_1\}$ and finally, $\mathtt{out}'(e') = v_0$ for any e' such that $\mathtt{out}(e') = v_2$ and $\mathtt{out}'(x) = \mathtt{out}(x)$ otherwise for any other $x \in E$.

◇ **Right axiom** We have a similar case for $\ell_E(e_2) = \text{ax}$ which correspond to the previous case in which we exchange e_1 (and v_1) for e_2 (and v_2).

◇ **Multiplicative** Assume we have $\ell_E(e_1) = \parr$ and $\ell_E(e_2) = \otimes$ (or the converse) with $\mathtt{in}(e_1) = (u_1, u_2)$ and $\mathtt{in}(e_2) = (w_1, w_2)$. The elimination of e_{cut} is a new proof-structure $\mathcal{S}' := (V', E', \mathtt{in}', \mathtt{out}, \ell_E)$ with $V' := V - \{v_1, v_2\}$, $E' := (E - \{e_{\text{cut}}, e_1, e_2\}) \cup \{e^1_{\text{cut}}, e^2_{\text{cut}}\}$ and finally, $\mathtt{in}'(e^i_{\text{cut}}) = (v_i, w_i)$ for $i \in \{1, 2\}$ and $\mathtt{in}'(x) = \mathtt{in}(x)$ otherwise for any other $x \in E$.

§29.11 There exists a remarkable extension of MLL with a rule called MIX (*cf.* Figure 29.7), initially studied by Fleury and Rétoré [FR94]. This rule corresponds to the axiom scheme $A \otimes B \multimap A \parr B$ and constitutes, together with the other rules of MLL, a new proof system called MLL+MIX. Beside this new rule, MLL+MIX works with the same

formulas as MLL. In particular, all MLL sequent calculus proofs are MLL+MIX sequent calculus proofs as well.

§29.12 We now would like to define the underlying proof-structure of an MLL+MIX sequent calculus proof, exactly as informally done in Figure 29.3. In order to do so, we define a labelling of the vertices of proof-structures (for which labels correspond to formulas) in order to make proof-structures look like actual proofs. By doing so, we already implicitly give a little bit of meaning to the purely computational proof-structures.

§29.13 **Definition** (Labelled proof-structure)**.** A *labelled proof-structure* is a tuple

$$\mathcal{S} = (V, E, \texttt{in}, \texttt{out}, \ell_V, \ell_E)$$

where $(V, E, \texttt{in}, \texttt{out}, \ell_E)$ is a proof-structure and $\ell_V : V \to \mathcal{F}_{\text{MLL}}$ is a function labelling vertices of V by formulas.

We write $\vdash \mathcal{S} : \Gamma$ with a set of formula $\Gamma := \{\ell_V(v) \mid v \in \texttt{Concl}(\mathcal{S})\}$ in order to specify the formulas associated with the conclusions of $\mathcal{S}$.

§29.14 In Figure 29.8, we define a translation $[\![\cdot]\!]$ from MLL+MIX sequent calculus derivations to labelled proof-structures. Notice that this translation is not surjective, and that some proof-structures do not represent sequent calculus proofs. This is tackled by the *correctness criterion*, which characterises those proof-structures that do translate sequent calculus proofs through structural properties and which are considered "correct". but for the time being, we give a preliminary definition of proof-net, the proof-structures coming from sequent calculus proofs. The relationship between proof-structures, proof-nets and sequent calculus proofs is illustrated in Figure 29.9.

§29.15 The MIX rule corresponds to allowing disjoint union of proof-structures as being "correct". Although not "logical" (*i.e.* not coming from MLL sequent calculus which decomposes intuitionistic and classical logic), MLL+MIX proofs keep interesting computational properties which naturally appear in various models of linear logic such as coherence spaces [Gir87a, Chapter 4].

§29.16 **Definition** (MLL and MLL+MIX proof-nets)**.** An MLL (*resp.* MLL+MIX) *proof-net* is a proof-structure $\mathcal{S}$ for which there exists an MLL (*resp.* MLL+MIX) sequent calculus proof π such that $\mathcal{S} = [\![\pi]\!]$.

§29.17 Once we get out of the multiplicative paradise, things are less simple. Even today, the theory of proof-nets suffer from a lot of various problems when trying to extend to other fragments of linear logic. In the next sections, I mention other fragments of linear logic with some being less detailed than others. Units and additives are rather considered problematic but exponential proof-nets are fine although not perfect. When designing extensions of proof-nets we have to think about the following points:

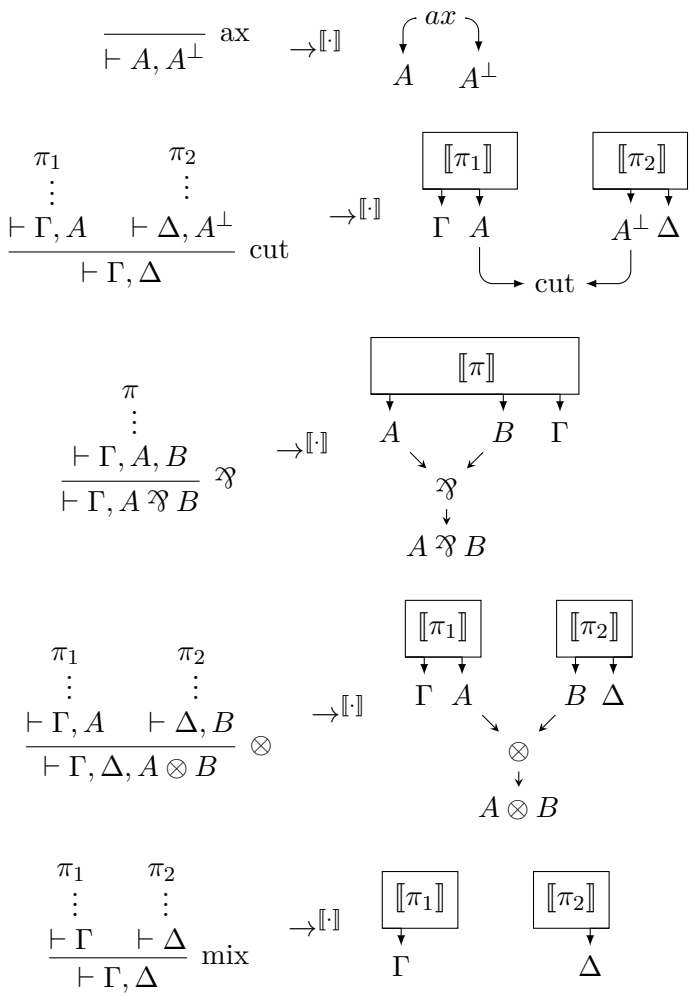


Figure 29.8: Translation of MLL+MIX sequent calculus proofs into labelled proof-structures.

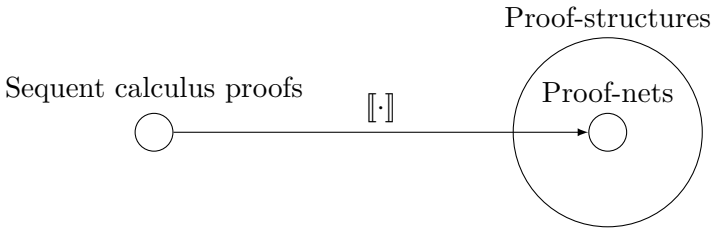


Figure 29.9: Proof-structures are general structures in which proof-nets can be defined as special cases coming from the translation of sequent calculus proofs. Correctness criteria are ways to tell if a proof-structure is correct, *i.e.* corresponds to a proof-net independently of sequent calculus proofs.

◇ **Loss of canonicity** proof-nets are meant to capture the essence of proofs. In particular, it has to enjoy a good canonicity by providing a synthetic syntax for proofs. This is usually handled by considering "commutation rules" transforming sequent calculus proofs into equivalent proofs up to exchange of rules [HH16, Section 1]. Other equivalences of proofs can be considered as well but translation into proof-structures have to be invariant under these equivalences;

◇ **Simulation of cut-elimination** proof-structures must be able to faithfully simulate cut-elimination as it is done in the sequent calculus. This is not always the case because sequent calculus is often subject to some implicit operations which leave some unwanted parts or residuals in the cut-elimination of proof-structures (for instance the cut-elimination $\&/\oplus_k$ hides an erasure of either $\vdash \Gamma, A_1$ or $\vdash \Gamma, A_2$);

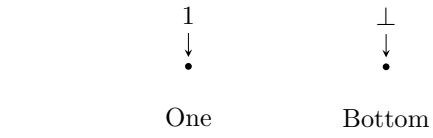
◇ **Effective correctness criterion** we must be able to computationally check if a proof-structure is correct, *i.e.* it is the translation of a sequent calculus proof. This means that we are able to characterise a fragment of linear logic.

◇ **Ad-hoc extensions** With all the previous points, it can happen that some ad-hoc technical and unnatural changes are made to proof-nets. Proof-nets should be able to provide an analysis of sequent calculus proofs for linear logic and if possible, of mathematical logic in general. The design of proof-nets should not get lost in laborious technicalities for the sole sake of making things work[10] (although it can indeed be theoretically interesting to develop further extensions).

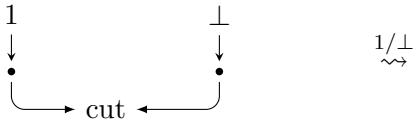
Multiplicative unit proof-nets

§29.18 The links and cut-elimination of multiplicative proof-structures are presented in Figure 29.10. The extension of the translation of sequent calculus proofs to proof-structures is straightforward. The proof of $\vdash 1$ becomes the link for 1 and the proof of $\vdash \Gamma, \bot$ becomes the link for $\bot$ coming attached to a proof-net corresponding to Γ. If this $\bot$ interacts with a link 1, then only the context Γ associated with $\bot$ is left.

§29.19 **Forgotten dependency.** Notice that there is no way to tell what context Γ a $\bot$ link is attached with, although $\bot$ constants are always in the same sequent as a context Γ in sequent calculus. At this point it is not a problem since we only consider cut-elimination but correctness will be problematic as we will see later.



(a) Links/constructors of multiplicative units.



(b) Cut-elimination reductions. The two constants are erased when interacting.

Figure 29.10: Multiplicative unit proof-structures.

Additive proof-nets

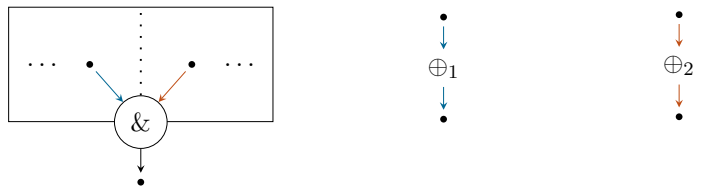
§29.20 There are several different designs of additive proof-nets. The particularity of additive proof-nets is that the formula $A \,\&\, B$ has to share a same context Γ for its to premises A and B (*cf.* Figure 27.2b). As for the cut-elimination (*cf.* Figure 29.11b), a branch of the & rule is selected but other one is erased. Problems regarding the different presentation of MALL proof-nets are presented in one of Marc Bagnol's notes[11].

§29.21 **Additive boxes.** The solution of additive boxes [Gir96, Section 1.1] (I personally like Gimenez's PhD thesis for an accessible explanation of additive boxes [Gim09, Chapter 4]) is presented in Figure 29.11. The connective & is handled with a box having two disjoint parts. Each premise is wired in its own side. In the rule of &, a context Γ is duplicated in two premise sequents $\vdash \Gamma, A$ and $\vdash \Gamma, B$, in the case of proof-structures a proof-structure corresponding to Γ is duplicated and located in each side of the corresponding & box. Its conclusions can go outside the box to interact with other proof-structures. When a & box interacts with a selector $\oplus_k$, it opens one corresponding side of the box (left for $\oplus_1$ and right for $\oplus_2$) and destructs the other.

§29.22 **A return to sequentiality.** Additive boxes work quite well. If we are not too demanding, we can be satisfied and even use them as such as in Gimenez's PhD thesis [Gim09]. There are also reasons to not be satisfied:

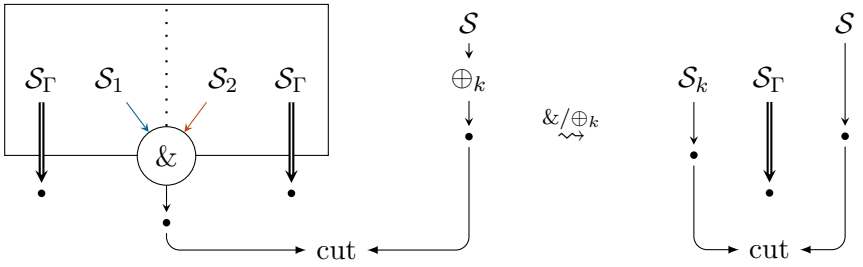
[10] When people learn computer programming for the first time, they tend to think that a working program is sufficient. However, a program's life is not set in stone. People will read the code, want to change it for various reasons, or use it as a tool for another purpose. Therefore, programs should not only be working but also workable.

[11] https://www.normalesup.org/~bagnol/notes/additifs.pdf



With (additive box) Plus left Plus right

(a) Links/constructors of additive proof-structures.



(b) Cut-elimination reductions. The selection triggers a box opening of either the left or right part of the additive box and erase the other.

Figure 29.11: Additive proof-structures with additive boxes.

$$\dfrac{\dfrac{\vdash \Gamma, A, C \quad \vdash \Gamma, A, D}{\vdash \Gamma, A, C \,\&\, D}\,\& \qquad \dfrac{\vdash \Gamma, B, C \quad \vdash \Gamma, B, D}{\vdash \Gamma, B, C \,\&\, D}\,\&}{\vdash \Gamma, A \,\&\, B, C \,\&\, D}\,\& \quad \leftrightarrow \quad \dfrac{\dfrac{\vdash \Gamma, A, C \quad \vdash \Gamma, B, C}{\vdash \Gamma, A \,\&\, B, C}\,\& \qquad \dfrac{\vdash \Gamma, A, D \quad \vdash \Gamma, B, D}{\vdash \Gamma, A \,\&\, B, D}\,\&}{\vdash \Gamma, A \,\&\, B, C \,\&\, D}\,\&$$

Figure 29.12: Example of commutation rule for &. It states the equivalence of two proofs up to permutation of rules.

- the context Γ is duplicated although it could be truly *shared* (and thus exist in uniquely one copy) and
- a *global* erasure occurs during cut-elimination (hence a rather *external* mechanism), as if there was an implicit human intervention. But do we need it to speak about logic? Cannot it be made explicit?;
- Similarly, the two contexts $\mathcal{S}_\Gamma$ in the left and right part have to be the *same*. This is a quite strong requirement for alogical objects. Maybe proof-structures are not so alogical after all;
- using boxes is a return to sequentiality. A part of a proof-structure is frozen and handled as a whole with some implicit and global operations. Actually, sequent calculus can be seen as having proof-nets where all links are boxes [Gir96, Section 1.1]. By using boxes (and thus introducing sequentiality), we can lose canonicity if we translate proofs such as the ones in Figure 29.12. With additive boxes, we would have one additive box duplicated inside the other and two different proof-nets. This can however be handled by adding more external congruence rules.

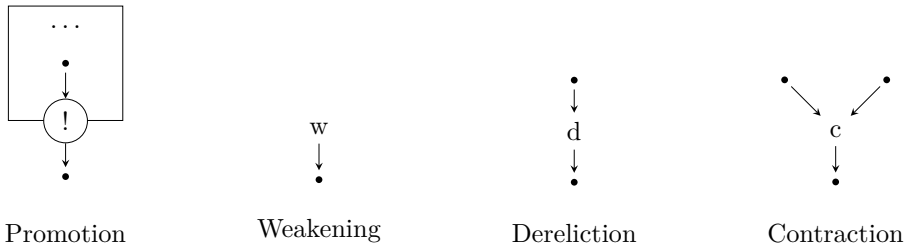
These are not necessarily problems but it is fair to not be satisfied and ask for better. If we have sequent calculus in mind, it may be fine but if we have bigger ambitions for a foundational theory of logic, it is difficult to be satisfied.

§29.23 **Boolean weights.** If we wish to get rid of boxes, another early solution is to use weights [Gir96, Section 1.4] where links are attached to a variable in a boolean algebra. It is then possible to identify the left and right premise/context of a & link and to act on it. This idea has been developed and modified by several authors such as Girard himself with monomial weights [Gir96, Section 1.4] or Laurent and Maieli [LM08].

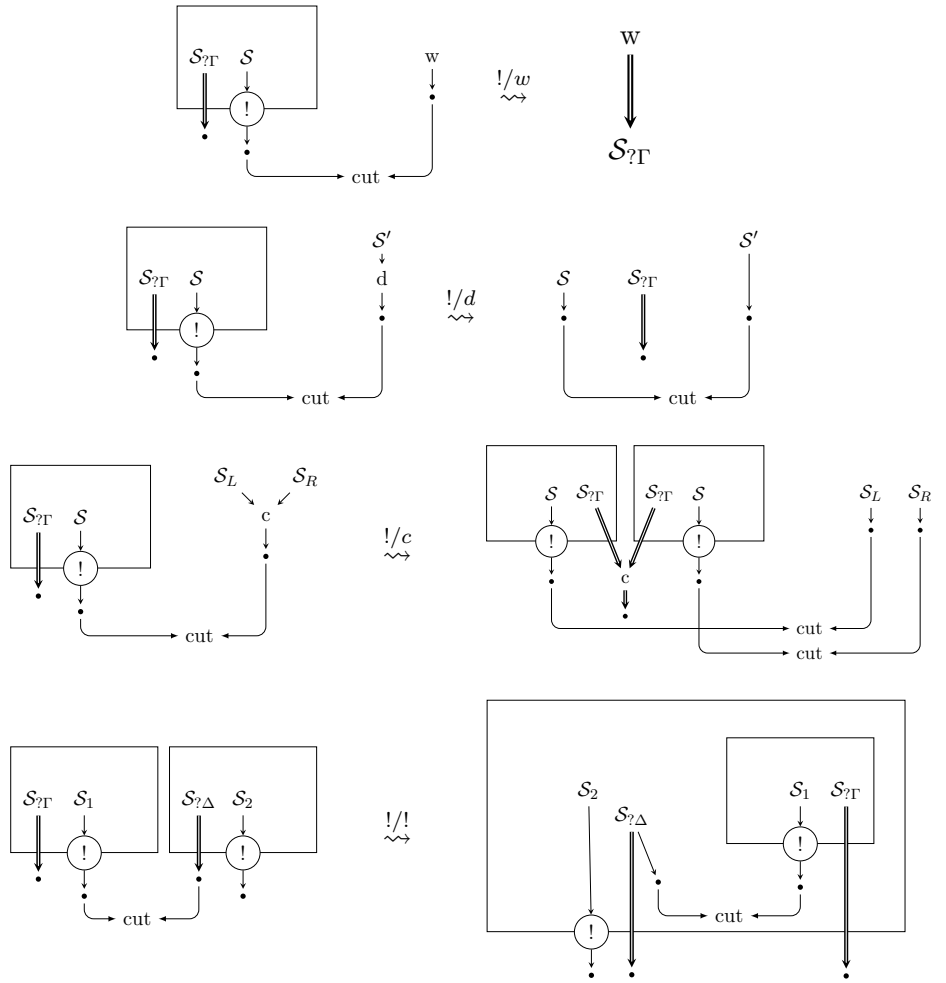
§29.24 **Additive slices.** Another solution is to use *additive slices* [Gir96, Section A.1.6] by considering a proof of $\vdash \Gamma, A\&B$ as a sum of two independent proofs of $\vdash \Gamma, A$ and $\vdash \Gamma, B$. A MALL proof-net is then a sum of all its slices, as if we made explicit all the "additive universes" for each choice with & (think of representing a non-deterministic Turing machine as a sum of deterministic machines for all possible choices). The problem is that we obtain very big proofs: if the proof-structure is cut-free then it can have 2^n slices for n the number of & connective. This idea has been developed by Hugues and Van Glabbeek [HVG03]. Another more recent solution is Heijltjes and Hugues' conflict nets [HH16] which is able to obtain a local canonicity (invariance under "local" commutation rules).

Exponential proof-nets

§29.25 **Exponential boxes.** The most natural translation of MELL proofs uses exponential boxes [Gue04, Section 3] presented in Figure 29.13. The use of boxes is rather faithful because in the sequent calculus, the promotion rule ! could already be seen as a box



(a) Links/constructors of exponential proof-structures.



(b) Cut-elimination reductions. Notice that there are a lot of global operations related to contexts.

Figure 29.13: Exponential proof-structures with exponential boxes.

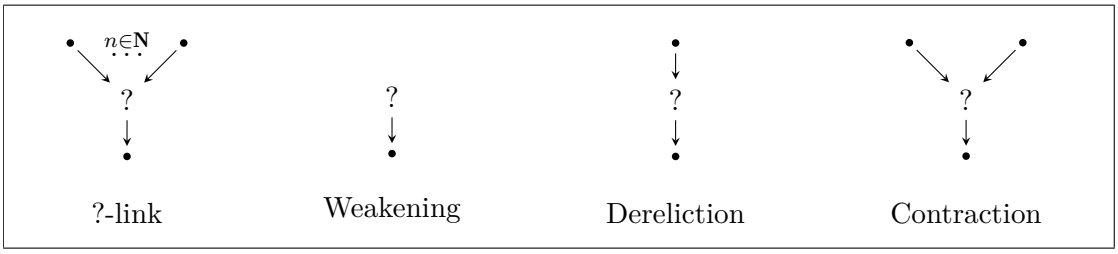


Figure 29.14: Generalised links for structural rules.

on which some operations could be done. The context ?Γ of the promotion rule is represented by links going out of a box without passing by the ! link. The conclusion of such links are called *auxiliary conclusions* passing through *auxiliary doors* of a box. The cut-elimination reductions of Figure 29.13b correspond to the erasure, opening, duplication and insertion of a box (in case a box interacts with an auxiliary conclusion of another box). For the $!/w$ cut-elimination, what remains is a weakening link for each conclusions appearing in the proof-structure $\mathcal{S}_{?\Gamma}$, exactly as in the corresponding sequent calculus cut-elimination.

§29.26 **Generalised structural rules.** A problem with this formulation of structural rules is that it is possible to construct arbitrary binary trees by using contractions. By using two contraction links, it is possible to construct two different trees having the same behaviour. This is a loss of canonicity. A solution is to use reduction rules to merge structural rules as in Di Cosmo and Kesner's works [DCK97, Section 4], to define congruences rules on proof-structures or to use Danos and Regnier's generalised structural rules [Gue04, Section 4.3]. Danos and Regnier's links come from the fact that structural rules can be seen as the instance of a same rule (*cf.* Figure 29.14) similarly to the multiplexing rule of Paragraph 27.21. If only have one n-ary ? link which is always merging with the others connected to it and pushed out of boxes. This recover the canonicity lost by the original structural rules.

§29.27 **Lambda-calculus.** Since MELL is where the typed λ-calculus can be expressed, it is natural that we can also obtain a translation of typed λ-terms into proof-nets [Dan90, Reg92, Gue04]. More surprisingly, it is also possible to define untyped λ-calculus directly with proof-structures (and not proof-nets) since they are alogical (actually not exactly) [Dan90, Section 11]. This simulation is based on the translation of intuitionistic logic into linear logic. Regarding encodings of classical logic, it is possible to find encoding of $\lambda\mu$-calculus in proof-nets in Olivier Laurent's works [Lau03, Lau99]. Proof-nets are also able to provide a canonical presentation of λ-terms. Regnier shows that a subtle equivalence on terms known as σ-equivalence can be simulated by the elimination of some multiplicative cuts [Reg94]. This is also presented in English in Olivier Laurent's works [Lau03, Definition 11]. It is also possible to encode more sophisticated functional languages such as PCF [Gim09, Section 5.4].

§29.28 **Explicit substitutions.** In the context of λ-calculus, exponential boxes actually enjoy a computational interpretation by themselves: they correspond to *explicit substitutions* [ACCL91, Ros96]. In λ-calculus, the β-reduction is defined by $(\lambda x.M)N \rightsquigarrow \{x := N\}M$ which is an external operation. However, it can make sense to have explicit substitutions set in the syntax of terms. Such substitutions can be moved, erased, opened, duplicated freely. This exactly correspond to exponential boxes which allow to manipulate a frozen section of a proof/program. Works in that direction have been led by Delia Kesner and Beniamino Accattoli [Acc18, DCKP03, DCK97].

§29.29 **Box-free approaches and optimal reduction.** In λ-calculus, when we have a redex $(\lambda x.M)N$, it happens that N is duplicated several times (the same thing occurs with exponential boxes). However, this duplication is useless: it is sufficient for the part of M requiring N to access to a unique copy of N, which correspond to a *sharing* of N. Lévy formalised this notion in his PhD thesis [Lév78] and defined a notion of *optimal reduction* for λ-calculus. This is only more than ten years later that Lamping came up with an algorithm for optimal reduction [Lam89] which accidentally corresponded to a box-free approach to proof-nets which features sharing [GAL92b]. Even more surprising, this approach is actually related [GAL92a] to Girard's geometry of interaction [Gir89a] which develops the ideas of linear logic (it is presented in the next chapter). The story and techniques of optimal reduction are told in Guerrini's survey [Gue04, Chapter 5].

30 Correctness criteria

§30.1 Proof-structures are almost free of logical meaning. We have two hyperedges $\otimes$ and $\parr$ but it is only a matter of labels. Nothing truly differentiates them although they correspond to different logical operations in sequent calculus and have to be distinguished. For that reasons, correctness criteria have been developed. They are method asserting whether a proof-structure corresponds to a proof-net. There exists a lot of various correctness criteria for a lot of fragments of linear logic, the most simple and standard being the Danos-Regnier correctness criterion. More information about other correctness criteria can be found in Marc Bagnol, Amina Doumane and Alexis Saurin's *"On the dependencies of logical rules"* [BDS15] and in Lutz Straßburger *"Proof Nets and the Identity of Proofs"* [Str06, Section 2.5]. In this chapter, I first introduce Girard's long trips criterion then Danos-Regnier correctness.

§30.2 A proof-structure is logically correct when it is possible to define a labelling such that it corresponds to a proof-net (which is itself the translation of a sequent calculus proof). In particular, a proof-structure can correspond to several proof-nets depending on the labelling we choose.

§30.3 **Definition** (MLL-certification and MLL+MIX-certification)**.** A proof-structure $\mathcal{S} = (V, E, \texttt{in}, \texttt{out}, \ell_E)$ is *MLL-certifiable* (*resp. MLL+MIX-certifiable*) with $\vdash A_1, ..., A_n$

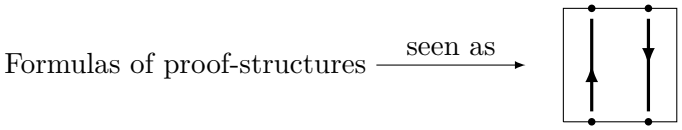
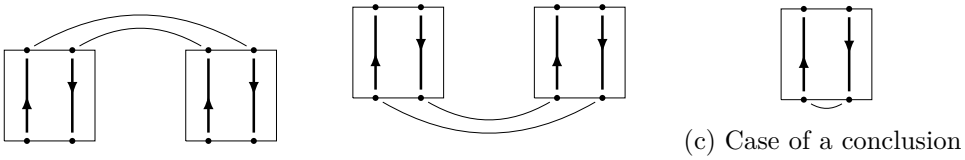


Figure 30.1: Ports and information flow of a binary connective.



(a) Case of an axiom. (b) Case of a cut. (c) Case of a conclusion formula.

Figure 30.2: Flow of information inside an axiom, a cut and a conclusion.

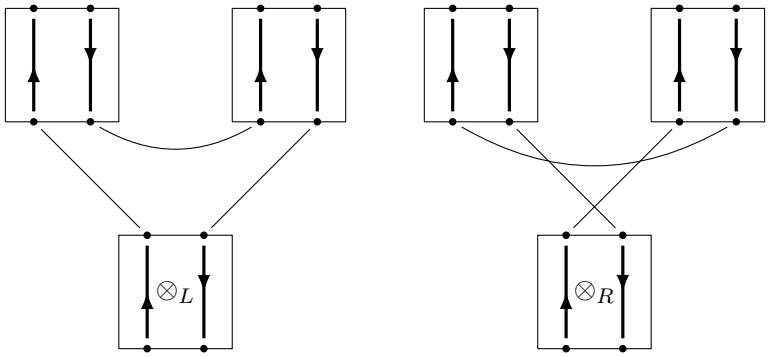
> when there exists a vertex-labelling function ℓ_V such that $(V, E, \texttt{in}, \texttt{out}, \ell_V, \ell_E)$ is an MLL (*resp.* MLL+MIX) proof-net.
>
> When there exists $\vdash A_1, ..., A_n$ such that $\mathcal{S}$ is MLL(+MIX)-certifiable with $\vdash A_1, ..., A_n$ then we simply say that $\mathcal{S}$ is MLL(+MIX)-certifiable.

Girard's long trip criterion

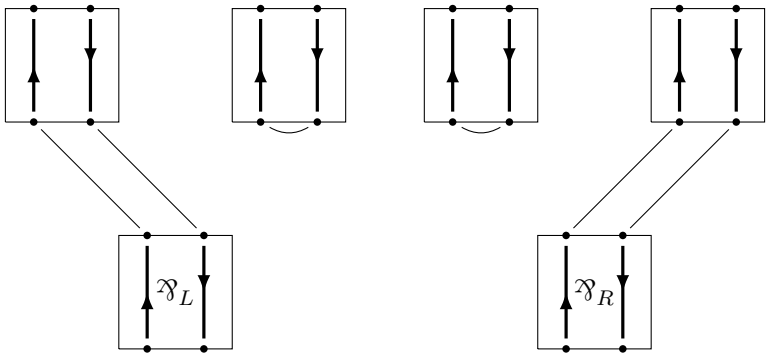
§30.4 **Sequentialisation.** The original and first criterion is Girard's long trips criterion [Gir87a, Section 2.2] for multiplicative proof-structures. The goal is to recover sequentiality back from a proof-structure and produce a sequent calculus proof. This process is known as *sequentialisation theorem*. Girard imagined a sequential, continuous and circular path exploring all formulas of a proof-structures. Those correct paths are called *long trips* and the incorrect ones are called *short trips*. All connectives can be seen as boxes with input and output ports. It is then possible to travel from the bottom to the top or the converse (*cf.* Figure 30.1). If we have a long trip, it would be possible to extract a sequential application of rules and the path corresponds to the *contour* of the sequent calculus proof. In order to define how to traverse a proof-structure, we have to take into account that $\parr$ and $\otimes$ have different behaviour and should not be traversed in the same way.

§30.5 **Case of axioms, cuts and conclusions.** The most basic cases of traversal are presented in Figure 30.2. They are very natural and there is nothing special about it.

§30.6 **Case of tensor.** The translation of sequent calculus proofs into proof-structures in Figure 29.8 shows that a correct tensor connects two disjoint proof-nets. It follows that when exploring from a tensor conclusion $A \otimes B$, we have to fully explore one premise, going back then exploring the other and returning to $A \otimes B$. Figure 30.3a shows that there is only two ways to that. These two choices of paths are called *switchings*. We have



(a) Flow of information in a tensor hyperedge.



(b) Flow of information in a par hyperedge.

Figure 30.3: Flow of information in binary connectives.

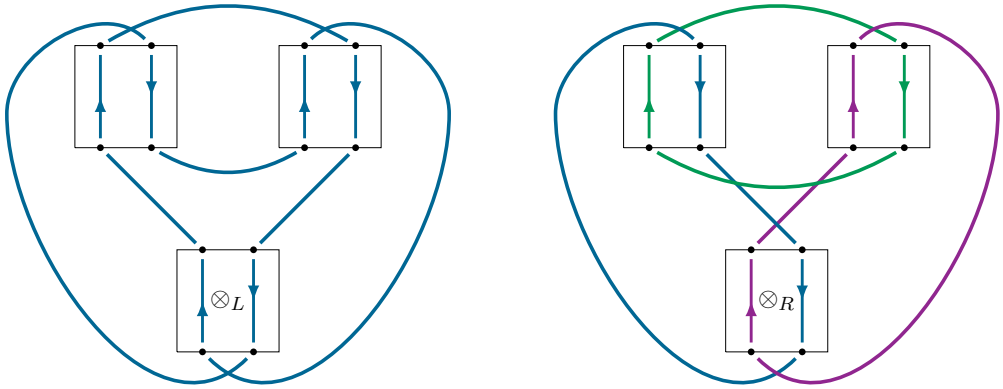


Figure 30.4: Example of situation where $\otimes_L$ succeeds but $\otimes_R$ fails. Based on a proof-structure construction from $\vdash (A \otimes B) \otimes (A^{\perp} \otimes B^{\perp})$. The exact construction with other boxes is left implicit and we only look at how ports are related.

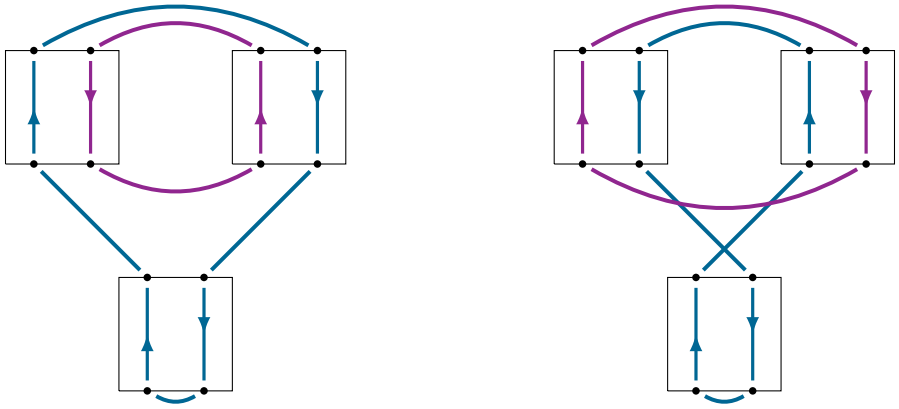


Figure 30.5: Reunion of formulas in an axiom. We obtain two short trips.

two switchings: $\otimes_L$ exploring the left premise first and $\otimes_R$ exploring the right premise first. Because the tensor splits in two disjoint components, switchings have to *reunite* the two branches in order to hope for a full connected traversal (long trip). An example is given in Figure 30.4 where for a same situation, $\otimes_L$ forms a long trip (success) but $\otimes_R$ forms three short trips (failure). This example can be obtained by constructing a proof-structure from the syntax tree of the sequent $\vdash (A \otimes B) \otimes (A^\perp \otimes B^\perp)$.

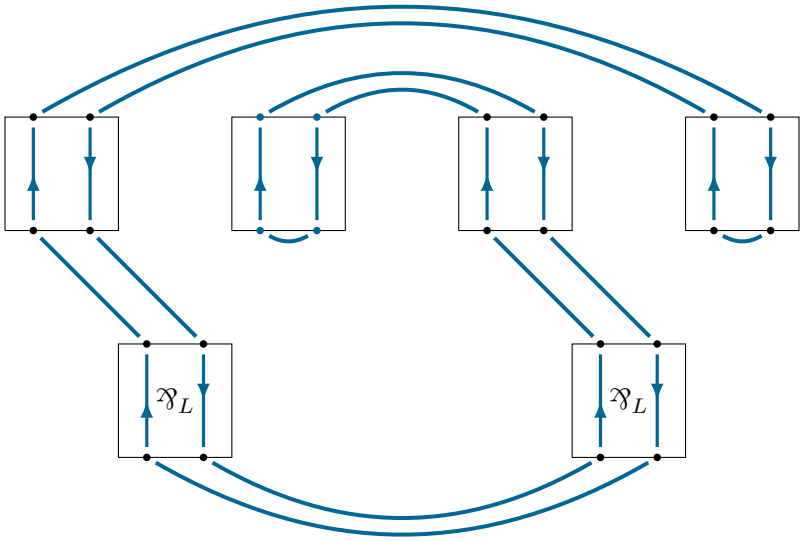
§30.7 **Case of par.** For the $\parr$ rule, this is the opposite situation. In Figure 29.8, we see that a $\parr$ link connects two formulas in a *same* connected component. Switchings must then *splits* the two premises to avoid short trips. We obtain the two ways of traverse the proof-structure in Figure 30.3b. Without splitting, we would reunite two already connected formula and form a short trip, as shown in the reunion of conclusion of axiom in Figure 30.5. From the sequent $\vdash (A \parr B) \otimes (B^\perp \parr A^\perp)$, it is possible to construct a situation where there is a long trip for a switching and not for another. This is illustrated in Figure 30.6.

§30.8 **Definition** (Girard switching)**.** Let $\otimes(\mathcal{S})$ and $\parr(\mathcal{S})$ respectively be the set of all $\otimes$ and $\parr$ hyperedges in a proof-structure $\mathcal{S}$. A (Girard) *switching* is a total function φ defined as the union of two functions:

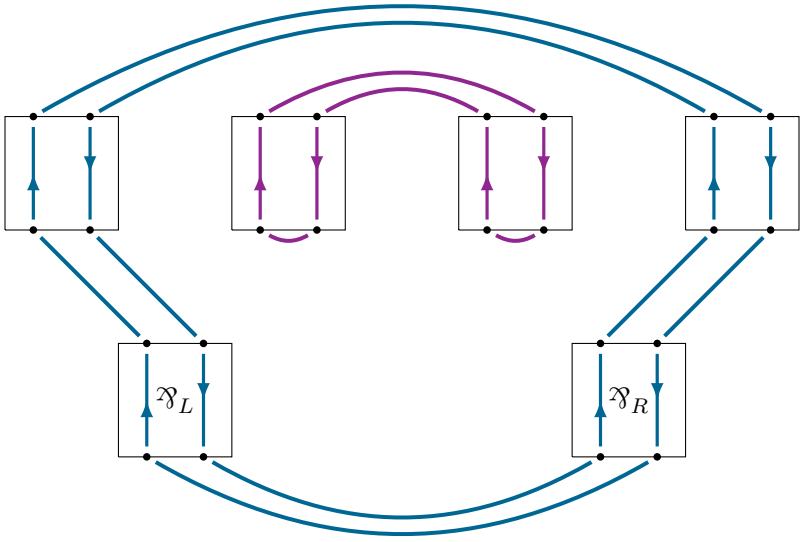
- $\varphi_\otimes : \otimes(\mathcal{S}) \to \{\otimes_L, \otimes_R\}$;
- $\varphi_\parr : \parr(\mathcal{S}) \to \{\parr_L, \parr_R\}$.

A switching of a proof-structure is hence a selection of a choice of traversal for any $\otimes$ and $\parr$ hyperedges.

§30.9 **Definition** (Switching flow graph)**.** Let $\mathcal{S}$ be a proof-structure and φ be a Girard switching. The *switching flow graph* $\mathcal{S}^\varphi$ associated with φ is a directed multigraph $(V, E, \texttt{in}, \texttt{out})$ where $V := V^\mathcal{S}$ and E follows the rules of flow of information given in



(a) Presence of a long trip.



(b) No long trip but two short trips.

Figure 30.6: Situation where a switching fails and the other succeed. Based on a proof-structure construction from $\vdash (A \parr B) \otimes (B^{\perp} \parr A^{\perp})$. A tensor is hidden below so you can just imagine that some construction is linking the two $\parr$ conclusions.

Figure 30.2, Figure 30.3b and Figure 30.3a. We use the notation $u \overset{e}{\mapsto} v$ of Appendix C for directed edges.

- ◇ **Axiom** if we have $e \in \mathtt{Ax}(\mathcal{S})$ such that $\mathtt{out}(e) = (u, v)$, then there are two edges $u \overset{e_1}{\mapsto} v$ and $v \overset{e_2}{\mapsto} u$;
- ◇ **Cut** it is similar for cuts;
- ◇ **Conclusion** for any conclusion v, we have a loop $v \overset{e}{\mapsto} v$;
- ◇ **Tensor** if we have $\ell_E(e) = \otimes$ with $\mathtt{out}(e) = v$ and $\mathtt{in}(e) = (v_l, v_r)$, there are two cases:
  - if $\varphi(e) = \otimes_L$, then we have edges $v \overset{e_1}{\mapsto} v_l$, $v_l \overset{e_2}{\mapsto} v_r$ and $v_r \overset{e_3}{\mapsto} v$;
  - if $\varphi(e) = \otimes_R$, then we have edges $v \overset{e_1}{\mapsto} v_r$, $v_r \overset{e_2}{\mapsto} v_l$ and $v_l \overset{e_3}{\mapsto} v$;
- ◇ **Par** there are two cases for each v conclusion of an $⅋$ hyperedge e with left premise v_l and right premise v_r:
  - if $\varphi(e) = ⅋_L$, then we have edges $v \overset{e_1}{\mapsto} v_l$, $v_l \overset{e_2}{\mapsto} v$ and $v_r \overset{e_3}{\mapsto} v_r$;
  - if $\varphi(e) = ⅋_R$, then we have edges $v \overset{e_1}{\mapsto} v_r$, $v_r \overset{e_2}{\mapsto} v$ and $v_l \overset{e_3}{\mapsto} v_l$.

§30.10 **Definition** (Trips)**.** A *trip* in a switching flow graph $\mathcal{S}^\varphi$ is a directed path in $\mathcal{S}^\varphi$. It is a *long trip* if it is an Eulerian circuit (a path traversing all edges exactly once and returning to its starting point). Otherwise, it is a *short trip*.

§30.11 **Theorem** (Girard correctness)**.** A proof-structure $\mathcal{S}$ is MLL-certifiable if and only if $\mathcal{S}$ has no short trip for any switching of $\mathcal{S}$.

Proof. A proof based on combinatorial arguments can be found in Girard's papers [Gir87a, Section 2.1][Gir11a, Section 11.3.3]. □

§30.12 **Question, answer and communication.** An intuition given by Girard about these trips [Gir11a, Section III.4.1] is that they correspond to the flow of questions and answers. Consider a function application $\lambda x.M : A \to B$ with an argument $N : A$, yielding the β-reduction $(\lambda x.M)N \rightsquigarrow_\beta \{x := N\}M$. The occurrences $x_i : A$ of x can be seen as *questions* asking for values of type A. The occurrences $N_i : A$ of N are then seen as *answers*. This interaction between inputs and outputs, or questions and answers can be understood through a notion of *communication*. Imagine a sequential communication exploring a proof-structure. We explore M and end on the question x_1. We go out and look for the answer N_1, then go back in M to find the next question and so on. The same idea appears for long trips criterion: there is a communication between axioms and the rest of the proof-structure (on which a switching has been applied). This dual communication between axioms and the rest of the switching flow graph will be made explicit in the next chapter about the geometry of interaction.

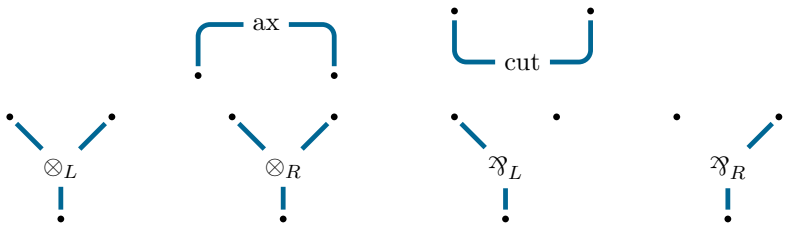
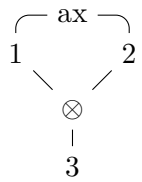
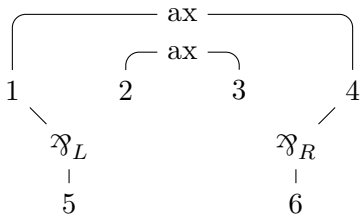


Figure 30.7: Undirected current flowing inside a proof-structure.



(a) Correctness hypergraph corresponding to an axiom connected to a tensor (*cf.* Figure 30.5). We have a cycle.



(b) Correctness hypergraph corresponding to the failure of Figure 30.6 where two short trips appear. We have two connected components. If we had $⅋_L$ on the right, then it would succeed.

Figure 30.8: Two failing correctness hypergraphs.

Danos-Regnier criterion for MLL+MIX

§30.13 The Danos-Regnier correctness criterion is a simplification of Girard's long trips criterion. Consider the long trip criterion. Instead of looking at paths of a flow of information inside a proof-structures, we simply look at the structure of the flow of information. Forget the directions of the current and see the boxes with ports as simple nodes. We obtain the undirected flow graph of Figure 30.7.

- The two switchings for ⊗ correspond to two three-way channels of communication. Since they have the same structure, they are identified and there is no need for a switching of ⊗ any more.

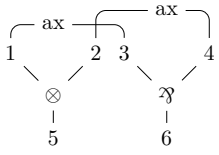
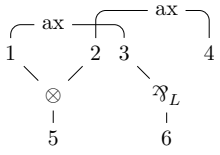
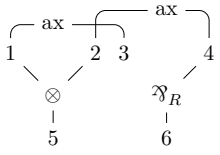
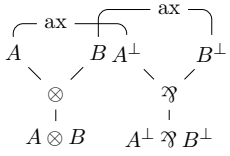
Proof-structure	Switching 1	Switching 2	Proof-net
			

Figure 30.9: A proof-structure, all its switching hypergraphs and a possible labelling corresponding to a proof-net.

- The two switchings for $⅋$ correspond to disconnecting one premise for each $⅋$ link.

If long trips are about directed graphs then Danos-Regnier correctness is about hypergraphs induced by trips. It follows that it is sufficient to consider a disconnexion of one premise for each $⅋$ link directly on a proof-structure and construct a *correctness hypergraph*. In particular, it is possible to show that all correctness hypergraphs for a proof-structure $\mathcal{S}$ is connected and acyclic (a tree) if and only if $\mathcal{S}$ has no short trip for any switching. In particular, all short trips induce either more than one connected component or a cycle. To illustrate this fact, two correctness hypergraphs corresponding to failure of finding a long trip are illustrated in Figure 30.8. An example of correct proof-structure is presented in Figure 30.9 together with its two only correctness hypergraphs and a possible corresponding proof-net.

§30.14 Since the Danos-Regnier correctness criterion is very standard in linear logic, we provide more formal definitions which will be used later in this thesis. We define the *correctness hypergraphs* associated with a proof-structure $\mathcal{S}$ as undirected copies of $\mathcal{S}$ with one source of each $⅋$-labelled hyperedge removed. The Danos-Regnier criterion states that a proof-structure is an MLL proof-net if and only if all its correctness hypergraphs are all connected and acyclic. Since the idea of logical correctness is purely structural as well, our definitions still deal with unlabelled proof-structures.

§30.15 **Notation.** Given a proof-structure $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$, we write $⅋(\mathcal{S})$ the subset $P \subseteq E$ of $⅋$-labelled edges, *i.e.* $⅋(\mathcal{S}) = \{e \in E \mid \ell_E(e) = ⅋\}$.

§30.16 **Definition** (Correctness hypergraph)**.** Let

$$\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$$

be a proof-structure. A (Danos-Regnier) *switching* is a map $\varphi : ⅋(\mathcal{S}) \to \{⅋_L, ⅋_R\}$. Its associated *correctness hypergraph* is the undirected hypergraph with labelled hyperedges $\mathcal{S}^\varphi = (V, E', \mathtt{end}, \ell_{E'})$ induced by the switching φ which is defined with:

- $\mathtt{end}(e) = \{u\}$ where $u = \overleftarrow{e}$ when $e \in ⅋(\mathcal{S})$ and $\varphi(e) = ⅋_L$;
- $\mathtt{end}(e) = \{u\}$ where $u = \overrightarrow{e}$ when $e \in ⅋(\mathcal{S})$ and $\varphi(e) = ⅋_R$;
- $\mathtt{end}(e) = \mathtt{in}(e) \cup \mathtt{out}(e)$ in all other cases.

The labelling $\ell_{E'}$ is defined by $\ell_{E'}(e) = \varphi(e)$ when $e \in ⅋(\mathcal{S})$ and $\ell_{E'}(e) = \ell_E(e)$ otherwise.

§30.17 **Theorem** (Danos-Regnier correctness)**.** A proof-structure $\mathcal{S}$ is MLL-certifiable if and only if it is MLL+MIX-certifiable and $\mathcal{S}^\varphi$ is connected for all switching φ.

Proof. Proven in Danos and Regnier's *"The structure of multiplicatives"* [DR89, Theorem 4]. □

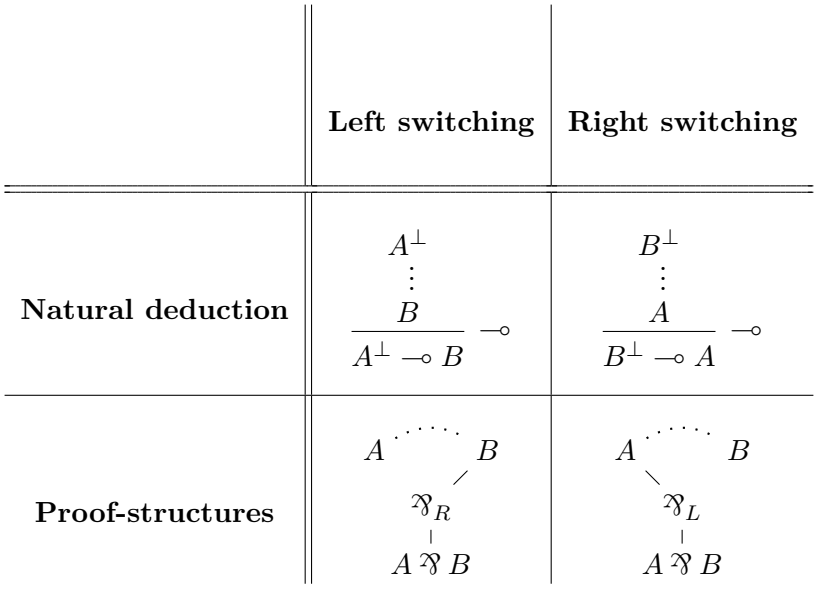


Figure 30.10: Switching for $\parr_L$ seen from the point of view of natural deduction. The premise $A^\perp$ and $B^\perp$ become conclusions A and B in monolateral sequent calculus and proof-nets.

§30.18 Once we defined Danos-Regnier correctness criterion for MLL, the correctness criterion for MLL+MIX is not so far from that. Since MLL+MIX proof-nets allow disjoint union of MLL-certifiable proof-nets, it is sufficient to only consider acyclicity of correctness hypergraphs.

§30.19 **Theorem** (MLL+MIX correctness)**.** A proof-structure $\mathcal{S}$ is MLL+MIX-certifiable if and only if $\mathcal{S}^\varphi$ is acyclic for all switching φ.

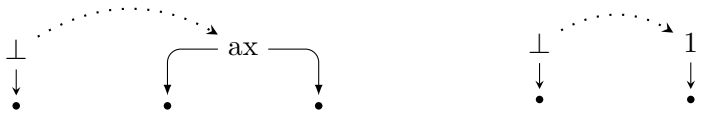
Proof. Proven in Fleury and Retoré's seminal paper [FR94, Theorem 4.7 and 4.8]. □

§30.20 Danos-Regnier can be puzzling when we are introduced to correctness criterion without intuitive explanations. Why do we need to disconnect either the left or right premise for each $\parr$ link? This is mainly because of combinatorial considerations about sequent calculus rules as explained previously. However, in the *"Blind Spot"* [Gir11a, Section 11.3.2], Girard suggests an intuitive explanation to this switching: we would like to extract/distinguish a tree-shaped structure from the proof-structure and obtain a well-defined natural deduction proof for linear logic (as cumbersome it would be). The formula $A \parr B$ is equivalent to both $A^\perp \multimap B$ and $B^\perp \multimap A$ (thanks to the commutativity of $\parr$). The switching then corresponds to a choice of natural deduction writing.

- If $A \parr B$ is written $A^\perp \multimap B$ then we have a premise reaching the conclusion B from the top in order to construct $A^\perp \multimap B$. However, as we saw, premise A are

$$\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax}}{\vdash A^\perp \parr A}\,\parr}{\vdash \perp, A^\perp \parr A}\,\perp \qquad \dfrac{\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash \perp, 1}\,\perp \qquad \dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax}}{\vdash \perp, A^\perp, A}\,\perp}{\vdash \perp, A^\perp, A}\,\text{cut}$$

Figure 30.11: Example of MLL_u proof with a $\perp$ constant.



(a) Jump to axiom. (b) Jump to 1.

Figure 30.12: Jumps in MLL_u for $\perp$. In the two case, we have connected and acyclic graphs.

negated conclusions $A^\perp$ and negated premises $A^\perp$ must be positive conclusions A. Hence it means that a (virtual) conclusion A is linked to B to construct $A^\perp \multimap B$. This is what happens with the switching $\parr_R$.

- The other case of $B^\perp \multimap A$ is similar.

To make it clearer, the two situations are illustrated in Figure 30.10. The structure is correct when we have a tree in the two cases (since natural deduction proofs are trees).

§30.21 In the next sections, extensions of the multiplicative criterion to other fragments of linear logic are explained. Once again, we are leaving the multiplicative paradise. All extensions below are based on Danos-Regnier correctness since it is standard and simpler than Girard's long trips.

Criteria for multiplicative units

§30.22 Multiplicative units introduce two links 1 and $\perp$ (*cf.* Figure 29.10). The unary link 1 corresponds to an axiom (because its rule terminates a proof) and it causes no problem for correctness. The constant 1 is typically introduced alone in its own branch, with a tensor for instance. Since there is no switching for $\otimes$, the correctness of a proof-structure with 1 will depend on the correctness of the rest of the proof-structure, without 1.

§30.23 As for $\perp$, it is where things get problematic. A constant $\perp$ comes together with a context Γ to which it can be attached or independent. Examples are given in Figure 30.11. Such proofs could be equivalently use $\parr$ rules ($\vdash \perp, A^\perp, A$ is equivalent to $\vdash \perp \parr A^\perp \parr A$). If $\perp$ is disconnected or if we consider a switching $\parr_L$ or $\parr_R$ disconnecting it, then we lose connectedness which was essential for multiplicative correctness. But a $\perp$ in a correct proof is always related to a correct context Γ and we would like to recover this. A

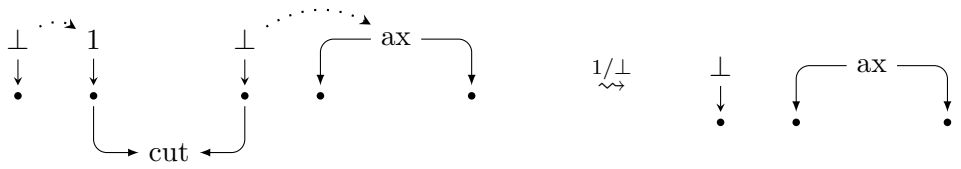


Figure 30.13: Cut-elimination with jumps does not preserve Danos-Regnier correctness. The initial proof-structure is correct but not its cut-elimination.

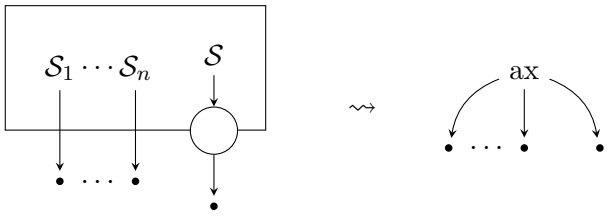


Figure 30.14: Transformation from boxes to generalised axioms.

solution is to use *jumps* [Gir96, Appendix A.2] which are artificial links relating each $\bot$ to a correct context. A correct context must always contain at least one constant 1 or an axiom. It is then sufficient to link occurrences of $\bot$ to any 1 or axiom. Two examples of jumps are given in Figure 30.12. A MLL_u proof with jumps is correct when it is connected and acyclic (the Danos-Regnier correctness criterion is preserved).

§30.24 A problem with jumps is whether they are part of proof-structures themselves or something added when verifying correctness. In the first case, cut-elimination can erase the target of jumps as in Figure 30.13. We have to recreate a jump for the $\bot$ links. This is rather complicated and unnatural. In the two cases, we have a sort of implicit external human intervention.

§30.25 As for MLL+MIX with units, the links 1 and $\bot$ cannot create cycles, hence any correct proof-structures with 1 and $\bot$ added will yields a correct proof-structure with the usual MLL+MIX correctness criterion. If we consider MLL+MIX and not MLL then there is nothing to change for the verification correctness (no jumps).

Criteria for additive proof-structures

§30.26 Assume we use additive boxes (*cf.* Figure 29.11) to represent MALL proofs. Boxes are frozen part of a proof. It is sufficient to hide the content of boxes and see boxes are generalised axioms and check for multiplicative correctness (*cf.* Figure 30.14). Then the content of each box should be correct as well. This generalised axiom is an hyperedge connecting all conclusions of the boxes (the conclusion of the box and the conclusions of the contexts associated with the box).

§30.27 In more recent representations of additive proof-structures, it is possible to find other correctness criteria [HVG03, HH16].

Criteria for exponential proof-structures

§30.28 Correctness for MELL (*cf.* Figure 29.13) is similar to correctness for multiplicative units and additive boxes:

- boxes are seen as generalised axiom and multiplicative correctness is checked in that context then locally for the content of each box;
- the only problematic structural rule is weakening which acts exactly like $\bot$ (*cf.* Section 30). Hence either we must consider MELL+MIX or jumps.

§30.29 We have previously seen that it was possible to encode untyped λ-terms with proof-structures (*cf.* Paragraph 29.27). MELL correctness then corresponds to choosing a type assertion $\Gamma \vdash M : A$ and typechecking a term M, *i.e.* verifying if M can be typed with A in the typing context Γ.

31 Discussion: the structure of normative constraints

§31.1 So, what linear logic and proof-net theory teach us about the nature of logic? Linear logic teaches to beware the appearance. The logical notions which are naturally accessible to our intuition such as disjunction ($\vee$), conjunction ($\wedge$) or implication ($\Rightarrow$) can hide subtle mechanisms. It is possible to provide an analysis of existing logical notions such as the implication $A \Rightarrow B$. It does not mean that if we "opened the black box" of implication, we would find linear logic providing new black boxes waiting to be opened, as if we had reductionist Matryoshka dolls. Linear logic should probably be understood as a tool to analyse our practice of logic. In particular, it may be possible to construct even more refined tools.

§31.2 From linear logic sequent calculus proofs, it is possible to characterise a bit more the *essence* of proofs by considering proof-nets. A problem is: what the essence of proof even is? In what direction should we go to extract this essence of proof? Technical considerations tell us that the essence of proofs corresponds to a notion of canonicity, *i.e.* of invariance under some equivalences on sequent calculus proofs. However, developments in proof-net theory seem to tell us that logic lives beyond sequent calculus and that this relative canonicity is probably not sufficient if we are interested in foundational questions.

§31.3 **Are correctness criteria useless.** When looking back at the history of linear logic and the introduction of proof-nets, a question may come to our mind: what is the purpose of correctness criteria? Why would we even want to check whether a proof-structure,

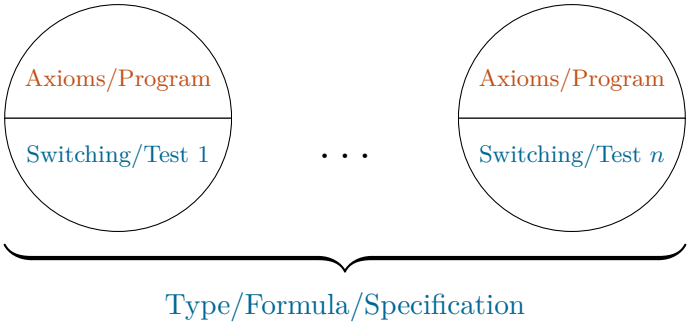


Figure 31.1: Correctness criterion as a way to test axioms (the computational essence of proofs). The interaction between axioms and test can be certified by either Girard's long trip criterion or the Danos-Regnier criterion.

a purely computational entity correspond to a proof-net? Cannot we simply work with proof-nets as primitive objects directly? What is all this nonsense? The translation of sequent calculus proofs into proof-structures even yields an inductive definition of proof-nets. Correctness criterion are not even really used in practice. Moreover even if it had a practical use, it does not even work well for full linear logic.

§31.4 **Correctness criteria as exit doors.** Correctness criteria are more interesting from a conceptual (not to say philosophical) point of view. They allow us to be *conscious of the format* we are in. They make us realise that something concrete and *material* regulate the logical entities we work with and their *use*. They exhibit the *shape* of logical constraints. Starting from proof-structures is actually a first step towards the dangerous computational and alogical world. We are testing the waters and there may be something beyond our format:

- what about starting from an even more general model of computation to go beyond proof-structures?
- if Danos-Regnier correctness is a way among others to test computational entities, what about considering other ways to test?

§31.5 **Correctness as testing.** Figure 31.1 shows that correctness hypergraphs can be seen as axiom links connected to a bottom part entirely definable from a sequent with a switching (since it corresponds to a switching applied to the syntax tree of a sequent).

- Axioms can be seen as a sort of program. They are invariant in the testing with switching hypergraphs. They hold the computational content of proofs.
- Switching hypergraphs can be seen as tests. A set of tests correspond to a specification.

It is similar to how programs are tested against several tests in order to check whether they have an expected behaviour or satisfy some specification. It makes logical correctness closer to the program testing of software engineering.

§31.6 **Making it explicit.** The theory of proof-nets suffers from several technical problems. Although a lot of them have been more or less solved, a question remains. What are we looking for? Are we satisfied? Is this the end of the history of logic?

- Logical correctness remains an *external* operation done on proof-structures. Hence logic is still *explained by us*. If we wish to understand logic more, we have to be able to reach an *internal explanation* without any "human intervention".
- This implicit human intervention is actually very present. In correctness for multiplicative units, jumps can be seen as an external intervention.
- Another philosophically problematic point is that additive, exponentials and multiplicative units hides global operation dependent on a logical system, which makes proof-structures not so alogical. For instance: the two equal contexts of additive boxes can be seen as a hidden regulation.

Girard's following quote at the end of the Blind Spot[12] (2011) summarises his opinion on these problems:

> "*Sequentialisation is not a dogma, it is a tool, which enabled one to find the procedural contents of nets; [...] But, on the whole, one has nothing against the idea of non-sequentialisable nets, as long as one can manipulate them: the ultimate meaning of logic is this ability to manipulate.*"
>
> – *Jean-Yves Girard [Gir87a, Section 11.C.5]*

This explains why Girard will later abandon the developments of original proof-nets to look for alternative spaces where linear logic can be expressed. In particular, he will strive for a more complete explicitation of those hidden assumptions of logical definitions where everything is syntax, including external and global operations.

§31.7 This is nothing but abstract and vague intuitions. This is only starting from Girard's transcendental syntax that these intuitions will be meaningful. But before reaching this point, the next destination is Girard's geometry of interaction.

[12] I recently learned that there are actually at least three different versions of it.

Chapter 5

The geometry of interaction

32 Towards a geometry of interaction

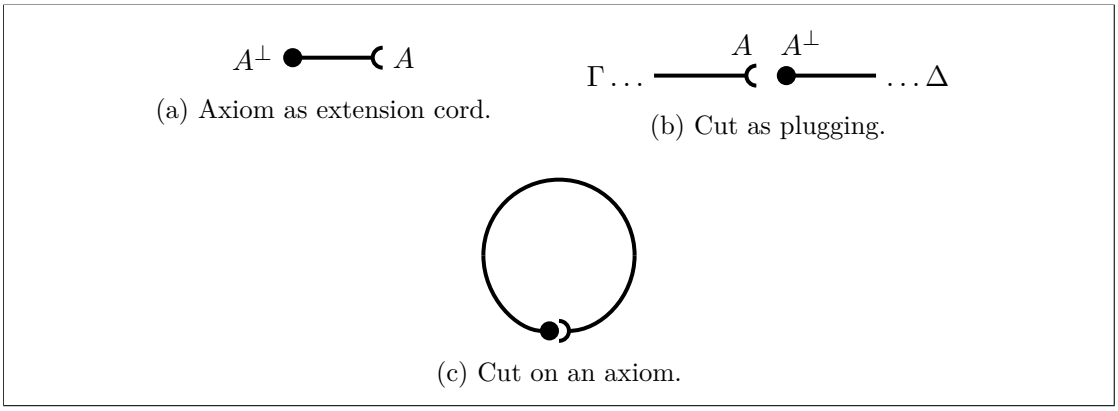
§32.1 Quickly following proof-nets, Girard introduced what he called a *geometrical semantics of computation*[1] by starting from the multiplicative case [Gir87b]. With such a name, anyone would wonder if it is still about logic.

§32.2 In his paper *"Multiplicatives"*, Girard remarks that proof-nets have something of a *geometrical* nature. We can actually forget formulas, connectives, everything and study proof-nets (and hence linear logic) only in terms of paths and shapes. This leads to a study of purely mathematical characterisations of proof-nets. Logic is not primitive but reconstructed from a notion of *interaction* corresponding to the now central cut-elimination procedure. He used simple permutations on natural number as a basis for proof-structures and was able to interpret MLL sequent calculus rules. This paper has later been developed by Danos and Regnier [DR89].

§32.3 It is only couple of years later that his new project has been presented under the name of "Geometry of Interaction" [Gir89b] (GoI) (a very confusing but cool name again). In this paper, he exposes his point of view on several topic of computer science and logic such as logic programming, denotational semantics and linear logic. What is striking is that this paper contains rather strong philosophical and conceptual insights. In particular, a very pragmatic point of view on linear logic is developed:

- MLL rules can be informally pictured with cables (*cf.* Figure 32.1). The connectives $\parr$ and $\otimes$ are two boxes in which the current flows differently;
- types/formulas are program specifications;
- a *dynamic* conception of meaning is defended;
- Girard opposes reductionism (which he associates with static interpretations of logic) and subjectivism (which he associates with bureaucracy, syntactic heaviness), and hence takes a philosophical position;

[1] It seems that he was not yet too allergic to the word "semantics" at that time.



(a) Axiom as extension cord.

(b) Cut as plugging.

(c) Cut on an axiom.

Figure 32.1: MLL identity rules with cables.

- vague links with physics or quantum computation are imagined at the end.

§32.4 What we now call "geometry of interaction" usually differ from Girard's original motivations. It usually refers to:

- abstract machine exploring proof-nets or typed λ-terms (encoded as proof-nets) [DR99, AL95];
- categorical semantics taking dynamics into account [HS06].

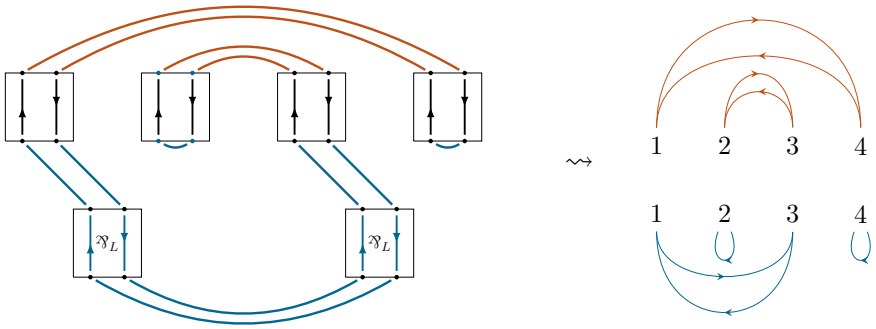
But in this thesis, "geometry of interaction" refers to Girard's original programme. Some works more faithful to Girard's original motivation exist. Thomas Seiller studied Girard's original geometry of interaction in his PhD thesis [Sei12b]. A categorical presentation can also be found in Etienne Duchesne's PhD thesis [Duc09].

§32.5 In Girard's original programme, the theory of proof-structure is seen from the point of view of *paths*. Even more than the order of applications of logical rules, we forget almost everything: formulas, connectives, logical rules... Only paths are remaining. Cut-elimination is a computational procedure on paths and correctness criterion are criterion on paths. This opens the interpretation of linear logic to several models of paths.

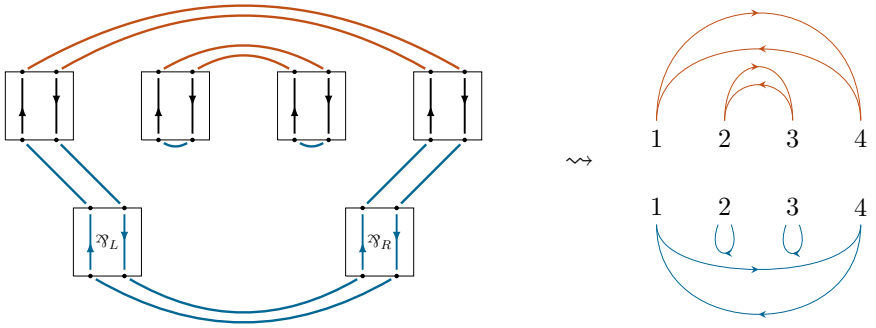
33 Multiplicative proofs with permutations

This section is inspired by Seiller's master thesis [2] and PhD thesis [Sei12b] from which some definitions are taken but explained in a different way.

[2] https://www.seiller.org/documents/pfnhyp.pdf



(a) Correct proof-structure. There is a long trip $2-3-1-4-1-3-2$.



(b) Incorrect proof-structure. There are two short trips $1-4-1$ and $2-3-2$.

Figure 33.1: Permutations associated with the two switchings of Figure 30.6.

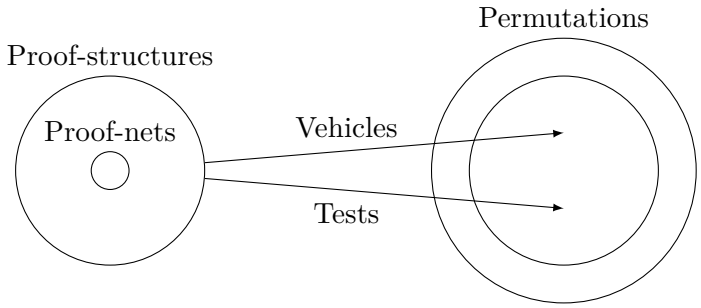


Figure 33.2: Shifting proof-net theory to permutation theory.

§33.1 From the end of the previous section, we have seen that switching flow graphs and correctness hypergraphs can be divided into two disjoint parts: the *vehicle* (top, axioms) and a *test* (bottom, syntax forest of sequent). The rough idea is that if we look at the long trip criterion for a proof-structure for a given switching flow graph $\mathcal{S}^\varphi$, paths do an alternation between vehicle and test. This decomposition is illustrated in Figure 33.2. If we only keep the mathematical content of the long trip criterion, both the vehicle and the test correspond to permutations over atomic formulas (represented by natural numbers standing for addresses of physical locations in the proof-structure). An example of translation of the paths of a proof-structure for two switchings (coming from Figure 30.6) into permutations is given in Figure 33.1.

§33.2 Let $X = \{1, ..., n\} \subseteq \mathbf{N}^+$ be a finite sequence of stricly positive natural numbers representing atomic formulas in a proof. A *permutation* is a bijection $\sigma : X \to X$. It associates with each elements of X a unique other element of X. An association can be written $x \mapsto y$ when we have a permutation σ such that $\sigma(x) = y$ (and $\sigma^{-1}(y) = x$). A permutation σ can then be written $\{x_1 \mapsto y_1, ..., x_n \mapsto y_n\}$ or $\begin{bmatrix} x_1 & ... & x_n \\ y_1 & ... & x_n \end{bmatrix}$ when $\sigma(x_i) = y_i$. The association should be read from top to bottom in its matrix presentation.

§33.3 From the trips of a switching hypergraph $\mathcal{S}^\varphi$ with a given switching φ, it is possible to extract two permutations σ_{ax} and σ_φ corresponding to the axioms of $\mathcal{S}$ and to $\mathcal{S}^\varphi$ respectively. We enumerate each boxes corresponding to the n atomic formulas and obtain a sequence of natural numbers $X = \{1, ..., n\}$.

- The permutation $\sigma_{ax} : X \to X$ is constructed by adding an association $n \mapsto m$ when there is a directed path from n to m from the top.
- The permutation $\sigma_\varphi : X \to X$ is constructed by adding an association $n \mapsto m$ when there is a directed path from n to m from the bottom.

For instance, the two permutations in Figure 33.1a can respectively be represented by $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{bmatrix}$ and the two permutations in Figure 33.1b by $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 \end{bmatrix}$.

Long-trip criterion with cyclic permutations

§33.4 **Cyclic permutations.** The point is now to characterise the long trip criterion only from permutations (which mean nothing in particular). Permutations can be composed like any functions. For instance, if we compose the two previous permutations $\sigma := \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{bmatrix}$ and $\tau := \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{bmatrix}$, it is possible to plug the associations vertically with σ over τ to construct the sequence of associations $1 \mapsto 4 \mapsto 4$, $2 \mapsto 3 \mapsto 1$,

$3 \mapsto 2 \mapsto 2$ and $4 \mapsto 1 \mapsto 3$, written $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 4 & 1 & 2 & 3 \end{bmatrix}$. This induces a new permutation $\tau \circ \sigma = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{bmatrix}$. Composition is often written $\tau\sigma$ instead of $\tau \circ \sigma$.

We say that a permutation $\sigma : X \to X$ is *cyclic* when for $\sigma^0(1) \neq \sigma^1(1) \neq ... \neq \sigma^{|X|-1}(1)$ with $1 \in X$. It means that starting from 1, if we move one step by following σ, we end on a different location (atomic formula), if we move one step further, we end again a location different from the previous and so on until we do $|X| - 1$ moves which would be sufficient to traverse all natural numbers. It means that the permutation when composed with itself forms one big cycle (you should be able to see how it is related to long trips) by alternating between the two components of the composition.

§33.5 If permutations are represented as in Figure 33.1, then to check if $\tau\sigma$ is cyclic, we can merge the two corresponding graphs along identical natural numbers. We start from 1. One application of $\tau\sigma$ correspond to moving one step along the edges. If we are able to always reach different numbers in 3 steps then we must have reached all the locations.

- If $\sigma := \sigma_{ax}$ and $\tau := \sigma_{\varphi}$ are extracted from the correct proof-structure of Figure 33.1a, then it appears that $\tau\sigma$ is cyclic because we have:
  $(\tau\sigma)^0(1) = 1$, $(\tau\sigma)^1(1) = 4$, $(\tau\sigma)^2(1) = 3$ and $(\tau\sigma)^3(1) = 2$.
- As for the two permutations of Figure 33.1b, we can see why their composition is not cyclic. If σ is the composition of the two permutations extracted from the proof-structure, then $\sigma^0(1) = 1$, $\sigma^1(1) = 4$ and $\sigma^2(1) = 4$ which is a short trip.

§33.6 **Decomposition of permutation.** Any permutation can be decomposed into a composition of circular permutations with disjoint domains. Each circular permutation is closed under reachability by association of a permutation. This gives an alternative notation for permutation where $(x_1x_2...x_n)$ corresponds to a circular sequence of associations $x_1 \mapsto x_2 \mapsto ... \mapsto x_n$. For instance (132) is the permutation $\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix}$ (1 gives 3 which gives 2 then 1 back again). The permutation $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{bmatrix}$ can be rewritten (14)(23) and $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{bmatrix}$ can be rewritten (13)(2)(4). We have that $\tau\sigma$ is cyclic when its decomposition is a unique cyclic permutation (corresponding to a long trip). Otherwise, each components of the decomposition correspond to a short trip.

§33.7 **Orthogonality.** We are now able to mathematically express the long trip criterion. We say that two permutations τ and σ are *orthogonal*, written $\tau \perp \sigma$, when $\tau\sigma$ is cyclic. Orthogonality is symmetric since permutations are bijective and $\tau\sigma$ can be rewritten $\sigma^{-1}(\sigma\tau)\sigma$ [Gir87b, Section 2.3].

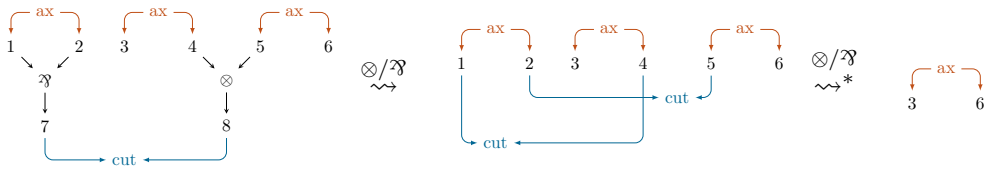


Figure 33.3: Cut-elimination for a proof-structure. We have the two following permutations: $\sigma_{ax} := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{bmatrix} = (12)(34)(56)$ and $\sigma_{cut} := \begin{bmatrix} 1 & 2 & 4 & 5 \\ 4 & 5 & 1 & 2 \end{bmatrix} = (14)(25)$.

§33.8 **Theorem.** A proof-structure $\mathcal{S}$ is MLL-certifiable if and only if for all switchings φ of $\mathcal{S}$, we have $\sigma_\varphi \perp \sigma_{ax}$ where σ_{ax} and σ_φ respectively correspond to the permutation associated with the axioms of $\mathcal{S}$ and the switching hypergraph $\mathcal{S}^\varphi$.

General interaction of permutation and cut-elimination

§33.9 The verification of the cyclicity of a permutation $\tau\sigma$ corresponds to a sort of *interaction* between two permutations. It works as if we connected two permutations to look at all the possible alternating paths induced. But cut-elimination is also a sort of interaction between axioms and cuts since multiplicative cuts are way to reorganise atomic formulas by rewiring or contraction.

§33.10 Cut-elimination can also be studied only from the notion of *paths*. A normal form makes explicit the shortcuts between input/output atomic formulas given by cuts. For instance, in Figure 33.3, the normal form connects 3 and 6. This normal form tells us that if we start from the input point 3, it is possible to reach 6 by following the rules of cut-elimination. Cuts also induce a permutation σ_{cut} on atomic formulas with the rewiring obtained by eliminating all multiplicative cuts $\otimes/⅋$. In Figure 33.3, the permutation corresponding to cuts is $\sigma_{cut} := \begin{bmatrix} 1 & 2 & 4 & 5 \\ 4 & 5 & 1 & 2 \end{bmatrix}$. The difference between verifying the long trip criterion and cut-elimination is that switching are *full* connexions between atomic formulas, represented by total permutations whereas cut-elimination only partially (potentially totally) connects atomic formulas. In the latter case, we must consider permutations over a subset of atoms.

§33.11 **Plugging.** Let $\sigma : X \cup Y \to X \cup Y$ and $\tau : Y \cup Z \to Y \cup Z$ be two permutations with X, Y, Z sequences of natural numbers over $\mathbf{N}^+$ and $X \cap Z = \emptyset$. The two permutations can be decomposed with disjoint partial injections (we keep the notation of permutations) relating subsets of their domain. We write $\sigma[E, F] : E \to F$ for such bijections. We obtain:

- the decomposition $\sigma = \sigma_{[X,X]} + \sigma_{[Y,X]} + \sigma_{[X,Y]} + \sigma_{[Y,Y]}$;

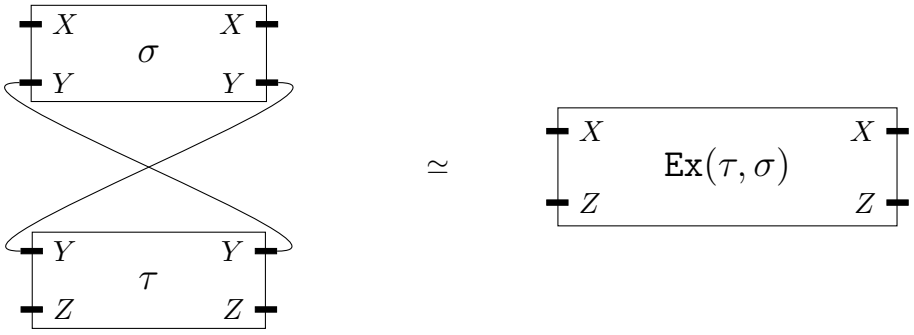


Figure 33.4: Interaction between two permutations $\sigma : X \cup Y \to X \cup Y$ and $\tau : Y \cup Z \to Y \cup Z$ with $X \cap Z = \emptyset$ as the computation of a new permutation $\mathtt{Ex}(\tau, \sigma) : X \cup Z \to X \cup Z$ which is defined only if we can always go out of the loop. The inputs are on the left and the outputs on the right.

- the decomposition $\tau = \tau_{[X,X]} + \tau_{[Y,X]} + \tau_{[X,Y]} + \tau_{[Y,Y]}$

where $\sigma + \sigma' : X \cup X' \to Y \cup Y' = \begin{bmatrix} x_1 & \dots & x_n & y_1 & \dots & y_n \\ x'_1 & \dots & x'_n & y'_1 & \dots & y'_n \end{bmatrix}$ when $\sigma : X \to X' = \begin{bmatrix} x_1 & \dots & x_n \\ x'_1 & \dots & x'_n \end{bmatrix}$ and $\sigma : Y \to Y' = \begin{bmatrix} y_1 & \dots & y_n \\ y'_1 & \dots & y'_n \end{bmatrix}$. To define the plugging (or interaction) $\mathtt{Ex}(\tau, \sigma)$ between σ and τ, we define a (potentially infinite) path starting either from X or Z to either X or Z by traversing the place of interaction Y (potentially several times). We obtain the following definition of plugging:

$$\mathtt{Ex}(\tau, \sigma) := \mathtt{Ex}(\tau, \sigma)_{[X,X]} + \mathtt{Ex}(\tau, \sigma)_{[X,Z]} + \mathtt{Ex}(\tau, \sigma)_{[Z,X]} + \mathtt{Ex}(\tau, \sigma)_{[Z,Z]} \text{ with}$$

$$\mathtt{Ex}(\tau, \sigma)_{[X,X]} := \sigma_{[X,X]} + \sum_{k=0}^{\infty} \sigma_{[X,Y]} \tau_{[Y,Y]} \big(\sigma_{[Y,Y]} \tau_{[Y,Y]}\big)^k \sigma_{[Y,X]};$$

$$\mathtt{Ex}(\tau, \sigma)_{[Z,X]} := \sum_{k=0}^{\infty} \sigma_{[X,Y]} \big(\tau_{[Y,Y]} \sigma_{[Y,Y]}\big)^k \tau_{[Y,Z]};$$

$$\mathtt{Ex}(\tau, \sigma)_{[X,Z]} := \sum_{k=0}^{\infty} \tau_{[Z,Y]} \big(\sigma_{[Y,Y]} \tau_{[Y,Y]}\big)^k \sigma_{[Y,X]};$$

$$\mathtt{Ex}(\tau, \sigma)_{[Z,Z]} := \tau_{[Z,Z]} + \sum_{k=0}^{\infty} \tau_{[Z,Y]} \sigma_{[Y,Y]} \big(\tau_{[Y,Y]} \sigma_{[Y,Y]}\big)^k \tau_{[Y,Z]}.$$

This new permutation $\mathtt{Ex}(\tau, \sigma)$ may not always be defined (it is not when there are infinitely many paths). The process underlying the construction of $\mathtt{Ex}(\tau, \sigma)$ is illustrated in Figure 33.4. The formula corresponding to $\sigma :: \tau$ is also known as the *execution formula*. I comment $\mathtt{Ex}(\tau, \sigma)_{[X,X]}$ (and the other permutations follow the same reasoning). It either goes directly from X to X in one step or goes to Y through $\sigma_{[Y,X]}$ then goes

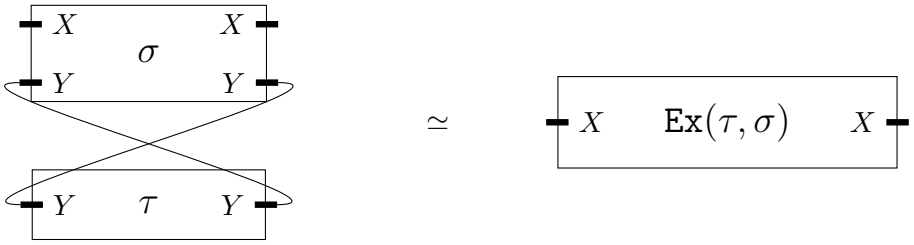


Figure 33.5: Interaction between two permutations $\sigma : X \cup Y \to X \cup Y$ and $\tau : Y \to Y$ as the computation of a new permutation $\mathtt{Ex}(\tau, \sigma) : X \to X$ which is defined only if we can always go out of the loop. The inputs are on the left and the outputs on the right. The box τ is often omitted and replaced by a loop from Y to Y in the box σ.

several times in a (potentially infinite) loop from X to itself to finally exit by X.

§33.12 **Plugging (simplified).** In the above definition, X and Z correspond to the atoms of two disjoint proofs which are not related by cuts. For instance, in Figure 33.3, we would have $X = \emptyset$ and $Z = \{3, 6\}$. The atoms which interact are $Y = \{1, 2, 4, 5\}$. However, our definition is too general. It is sufficient to set $Z := \emptyset$ at put all atoms unrelated to cuts in X. This is because the connexion between two disjoint proofs can be seen as an interaction happening within a single proof (*i.e.* you either see the proof-structure of Figure 33.3 as connecting two proofs by a cut or see it as a unique proof, as a whole). In this case we obtain the simplified version of plugging between $\sigma : X \cup Y \to X \cup Y$ (axioms with some atoms unrelated to cuts in X) and $\tau : Y \to Y$ (cuts):

$$\mathtt{Ex}(\tau, \sigma) = \mathtt{Ex}(\tau, \sigma)_{[X,X]}.$$

This situation of simplified plugging corresponds to the illustration of Figure 33.5 which simplifies Figure 33.4.

§33.13 We illustrate plugging with the proof-structure of Figure 33.3. We have

$$\mathtt{Ex}(\sigma_{cut}, \sigma_{ax}) := \sigma_{ax[X,X]} + \sum_{k=0}^{\infty} \sigma_{ax[X,Y]} \sigma_{cut[Y,Y]} \big(\sigma_{ax[Y,Y]} \sigma_{cut[Y,Y]}\big)^k \sigma_{ax[Y,X]}$$

with $X = \{3, 6\}$ and $Y = \{1, 2, 4, 5\}$. There is direct association from X to X in one step hence $\sigma_{ax[X,X]} = \emptyset$ (paths of length 1). Now we compute the formula for different values of k.

◇ **Case $k = 0$ (paths of length 3 from X to X):** we have

$$\sigma_{ax[X,Y]} \sigma_{cut[Y,Y]} \sigma_{ax[Y,X]}$$

$$= \emptyset \circ \begin{bmatrix} 4 & 5 \\ 1 & 2 \end{bmatrix} \circ \begin{bmatrix} 3 & 6 \\ 4 & 5 \end{bmatrix}$$
$$= \emptyset \text{ (undefined)};$$

$\diamond$ **Case $k = 1$ (paths of length 5 from X to X):** we have

$$\sigma_{ax[X,Y]}\sigma_{cut[Y,Y]}\big(\sigma_{ax[Y,Y]}\sigma_{cut[Y,Y]}\big)\sigma_{ax[Y,X]}$$
$$= \begin{bmatrix} 5 & 4 \\ 6 & 3 \end{bmatrix} \circ \begin{bmatrix} 2 & 1 \\ 5 & 4 \end{bmatrix} \circ \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \circ \begin{bmatrix} 4 & 5 \\ 1 & 2 \end{bmatrix} \circ \begin{bmatrix} 3 & 6 \\ 4 & 5 \end{bmatrix};$$

Remark that since we are alternating between axioms and cuts, only paths of odd length are considered. We see that there is a path of length 5 between 3 and 6, meaning that the proof-structure has a normal form.

§33.14 **Correctness with general plugging.** By using this more general definition of plugging, it is possible to characterise long trip correctness. The execution $\mathtt{Ex}(\tau, \sigma)$ computes *maximal alternating paths* from inputs/outputs. The idea is to turn a long trip into a maximal path. This is what Seiller did with his interaction graphs using graphs instead of permutations [Sei12a]. It is sufficient to distinguish an edge (representing an association in a permutation) and consider a maximal path between the two disconnected vertices. The advantage of expressing orthogonality with execution is that a lot of results are usually free such that the *adjunction property* which ensure a correct interpretation of linear logic and the possibility to define a categorical model ($*$-autonomous category).

§33.15 Proof-structures are now fully characterised by vehicles corresponding to links between atoms. The orthogonality relation $\perp$ formalises the fact that a vehicle (or program) passes a test. A permutation is hence MLL-certifiable (logically correct) when it passes all the tests for a given set of tests corresponding to a sequent $\vdash \Gamma$. It remains to define the set of tests corresponding to a sequent.

Interpretation of types/formulas

§33.16 Remark that in the world of permutations, there is no difference between vehicles and tests, they are of same nature. In particular, set of axioms (which can be seen as a type defined as a set of programs) opposes set of tests (which can be seen as way to assert that a program is of some type). We can construct arbitrary set of tests or vehicles by putting permutations in a set. Let $\mathbf{A}$ be such a set. We define its *orthogonal* $\mathbf{A}^{\perp} := \{\sigma \mid \forall \tau \in \mathbf{A}, \tau \perp \sigma\}$. Given a permutation σ, it is possible to say that it passes all the tests of a set $\mathbf{A}$ by writing $\sigma \in \mathbf{A}^{\perp}$. Not all sets will correspond to MLL formulas/types. The good sets are called *behaviours* and are closed by biorthogonal. The following definitions are very similar to realisability interpretations (*cf.* Section 22). An even more similar and notable work is Beffara's realisability interpretation used to

speak about concurrent computation and linear logic [Bef06]. His approache differs from the fact that he starts from process calculi but the same techniques are used. Beffara's work can be seen as an alternative interpretation of linear logic.

§33.17 **Definition** (Behaviour)**.** A set of permutations $\mathbf{A}$ is a *behaviour* when $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

§33.18 Behaviours corresponds to MLL formulas (in which linear negation is involutive). An intuitive way to understand this requirement is through the alternative equivalent definition.

§33.19 **Proposition.** Let $\mathbf{A}$ be set of permutations. We have that $\mathbf{A}$ is a behaviour if and only if there exists a set of permutations $\mathbf{B}$ such that $\mathbf{A} = \mathbf{B}^{\perp}$.

Proof. The proof can be found in the literature [JS21, Proposition 15]. □

§33.20 This alternative definitions tells us that $\mathbf{A}$ is a behaviour when there is a set of tests $\mathbf{B}$ fully characterising $\mathbf{A}$. In particular, all elements of $\mathbf{A}$ pass the tests of $\mathbf{B}$. In other word, a behaviour is set of *testable* permutations. The idea is now to construct behaviours corresponding to all MLL formulas.

§33.21 As we have seen in the previous chapter, the proof-nets of $A \otimes B$ join two disjoint proof-nets (*cf.* Figure 29.8). It is then natural to define the tensor as a disjoint union of set of permutations. Moreover, the switchings for $\otimes$ (*cf.* Figure 30.3a) correspond to reunion of switching flow graphs. The only way to be orthogonal to them (by forming a long trip) is having two disjoint switching flow graphs (otherwise we would have short trips).

§33.22 **Definition** (Support of set of permutations)**.** Let $\mathbf{A}$ be a set of permutations. If all permutations $\sigma \in \mathbf{A}$ are defined on a same set X, then X is called the *support* of $\mathbf{A}$. The support of $\mathbf{A}$ is written $\mathtt{dom}(\mathbf{A})$.

§33.23 **Definition** (Tensor)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two behaviours with disjoint support, *i.e.* $\mathtt{dom}(\mathbf{A}) \cap \mathtt{dom}(\mathbf{B}) = \emptyset$. We define their tensor as a new behaviour:

$$\mathbf{A} \otimes \mathbf{B} := \{\sigma_A + \sigma_B \mid \sigma_A \in \mathbf{A}, \sigma_B \in \mathbf{B}\}^{\perp\perp}.$$

§33.24 The double orthogonality in the definition is there to ensure that $\mathbf{A} \otimes \mathbf{B}$ is a behaviour. Depending on the chosen orthogonality relation, it is not always the case. If we already have a behaviour, *i.e.* $\mathbf{A} \otimes \mathbf{B} = \{\sigma_A + \sigma_B \mid \sigma_A \in \mathbf{A}, \sigma_B \in \mathbf{B}\}$, then we say that the tensor satisfies the property of *internal completeness*.

§33.25 Finally, $⅋$ is defined by the orthogonal of $\otimes$. The tests correspond to disjoint unions of switching flow graphs waiting to be reunited. The proofs of $A ⅋ B$ then reunites a proof of A with a proof of B. We finally also obtain a definition of linear implication.

§33.26 **Definition** (Par)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two behaviours with disjoint support. We define the new behaviour:

$$\mathbf{A} \parr \mathbf{B} := (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp.$$

§33.27 **Definition** (Linear implication)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two behaviours with disjoint support. We define the new behaviour:

$$\mathbf{A} \multimap \mathbf{B} := \mathbf{A}^\perp \parr \mathbf{B}.$$

§33.28 **Comparison with realisability interpretation.** This interpretation of linear logic matches with realisability interpretations in Section 22 (especially Krivine's classical realisability). The difference is that we are starting from permutations instead of λ-terms, there is no duality between λ-terms (programs) and stacks (tests) since they both are permutations in the GoI, and finally we are reconstructing multiplicative linear logic instead of classical logic. The pole is implicit in our case but could have been made explicit by defining orthogonality from execution. It is also possible to have an alternative definition of linear implication which would be closer to the definition of realisability.

§33.29 **Proposition.** Let $\mathbf{A}$ and $\mathbf{B}$ be two behaviours with disjoint support. We have:

$$\mathbf{A} \multimap \mathbf{B} = \{\sigma_F \mid \forall \sigma_A \in \mathbf{A}, \mathtt{Ex}(\sigma_F, \sigma_A) \in \mathbf{B}\}.$$

Proof. This is easily proven by using the *adjunction property* which is obtained by defining orthogonality from execution [Sei12a, Definition 26]. □

§33.30 **Generalised multiplicatives.** As illustrated in Figure 33.2, even though proof-net theory can be simulated with permutations, permutations are even more general than proof-structures. It is possible to construct permutations and make them interact without representing proof-structures. In particular, it is possible to define sets of tests not corresponding to MLL types and to obtain generalised formulas [Gir87b, Section 3]. A typical example is to mix tests coming from different formulas (by exchanging some $\parr$ and $\otimes$) because tests coming from MLL sequents are usually *uniform*. In some cases, the corresponding set of tests can have a non-empty orthogonal, meaning that it can be instantiated to actual proofs called *non-sequentialisable* since they do not correspond to sequent calculus proofs. However, the purpose of such generalised multiplicatives living outside usual MLL was not clear. It has been investigated later by Acclavio and Maieli by using partitions instead of permutations [AM20].

§33.31 **What about Danos-Regnier correctness?.** Since permutations are about binary links and that the test for tensor in Danos-Regnier correctness is a ternary edge (hyperedge), the interpretation of proofs with permutation can naturally express long trips but not Danos-Regnier switching hypergraphs. Other solutions will be proposed in the next sections (*cf.* Section 37). The transcendental syntax, which is the subject of this

thesis will also be a solution where both long trips and Danos-Regnier correctness can be expressed. However, it is still possible to characterise MLL+MIX in another way [NS19, Theorem 42] by using acyclicity for orthogonality, which can be expressed with *nilpotency* of the interaction between two permutations.

§33.32 **A paradigm of uniform interaction.** More than a new interpretation of proof-nets, the GoI provides a paradigm in which proofs and tests are computationally interactive entities of same kind. In particular, the cut is a way to put two entities in conflict and cut-elimination is a way to trigger an interaction. The interpretation of types as behaviours provides a way to speak about the computational behaviour of logical entities depending of how they interact with other objects. This idea of paradigm of interaction has been discussed by Abramsky [Abr16, Section 5].

§33.33 **Communication without understanding.** The notion of "communication without understanding" is a recurrent idea in Girard's papers. It appeared since the first paper on GoI [Gir89a, Section V.2] and served as a motivation until his project of transcendental syntax, succeeding the GoI. The idea is that when two logical entities Φ and Ψ interact (permutations or operators representing proof-nets), they are connected by points of reference (for instance a cut between $A^\perp \parr B^\perp$ and $A \otimes B$) making them able to react in a generic way through their "interface". However, this communication is done without understanding since what happens in one entity is independent from the other entity. The computational reaction is only dependent on the points of reference which connects Φ and Ψ. Actually, we could change names or substitute the atoms X_i by more complex proof-nets in Φ, Ψ would not be aware of that private change and still react in the same way. As Girard explains, it is as if Φ and Ψ corresponded to an internal language of thoughts for two persons (their consciousness?). These two persons have an illusion of mutual understanding through points of reference allowing a consensus, but those points are treated generically from each person's point of view. For instance, two persons talk about "democracy" or left/right in politics but although they can recognise the words and react to it, they will both treat the word depending on their own point of view or understanding.

34 Infinitary extension towards full linear logic

§34.1 This idea of representing proofs by permutations has been extended to MELL in Girard's first official technical paper on the GoI [Gir89a]. This is the first paper of a series of six papers. Atomic formulas can now be duplicated or erased, hence there can be infinitely many copies of one atomic formula. We need to extend our interpretation to something which would look like permutations (or matrices since permutations induce matrices) over an infinite countable space. A natural[3] solution has been suggested by Girard. There are a lot of equivalent formulation of this solutions which can be found in Girard's

[3] Apparently it is natural for mathematicians.

article on GoI [Gir89a], Seiller PhD thesis [Sei12b, Section 4.1], one of Shirahata's article [Shi03, Section 3] or Duchesne's PhD thesis [Duc09]. This is due to the fact that the interpretation works in very "large" spaces but not all the power of these spaces is used. It is then sufficient to consider simpler interpretations. In Girard's terms [Gir11a, Section 19.3.10], his solution is basically a *"preposterous dressing of a theory of the partial permutations of* $\mathbf{N}$*"*.

§34.2 A canonical solution[4] is to consider operators on the Banach space $\mathcal{B}(\mathbb{H})$ of bounded operator on the Hilbert space $\mathbb{H} := l^2(\mathbf{N})$. This provides a sort of generalised algebra of matrices. Objects of $\mathbb{H}$ are infinite sequences of complex numbers indexed by natural numbers $\mathbf{z} = (z_i)_{i\in\mathbf{N}} = (z_1, z_2, ...)$ such that $\sum_{i=0}^{\infty} z_i\overline{z_i} < \infty$ where $\overline{a+ib} = a - ib$. These sequences will represent atomic formulas in a proof-structure. The norm of $\mathbf{z}$ is defined $\|\mathbf{z}\| := \sqrt{\sum_{i=0}^{\infty} z_i\overline{z_i}}$. Operators u (which are used to represent proofs by generalising finite permutations) are linear transformation[5] on $\mathbb{H}$ such that $\mathtt{sup}\{\|u(\mathbf{z})\|$ such that $\|\mathbf{z}\| = 1\}$ is finite. Given two sequences $\mathbf{x} = (x_i)_{i\in\mathbf{N}}$ and $\mathbf{y} = (y_i)_{i\in\mathbf{N}}$, it is possible to define their *scalar product* $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i=0}^{\infty} x_i y_i$. From this scalar product, we can define an adjoint operation $u \mapsto u^*$ such that $\langle u(\mathbf{x}), \mathbf{y}\rangle = \langle \mathbf{x}, u^*(\mathbf{y})\rangle$. You do not have to understand what that means[6].

§34.3 We now have a lot of tools from this large space for linear logic. The most important elements we need is:

- a way to *internalise* $\mathbb{H}\oplus\mathbb{H}$ into $\mathbb{H}$, yielding $\mathbb{H}\oplus\mathbb{H} \simeq \mathbb{H}$ in order to interpret binary multiplicative connectives;
- an internalisation yielding $\mathbb{H}\otimes\mathbb{H} \simeq \mathbb{H}$ in order to interpret copies of atoms (for exponentials).

As explained by Seiller [Sei12b, Section 4.1], it is sufficient to use bijections on natural numbers yielding $\mathbf{N}\times\mathbf{N} \simeq \mathbf{N}$ and $\mathbf{N}+\mathbf{N} \simeq \mathbf{N}$ because a separation of the space of indices is sufficient to separate $\mathbb{H}$.

§34.4 **Interpretation of binary connectives.** The internalisation of direct sum takes the form of two operators[7] $l : \mathbb{H} \to \mathbb{H}$ and $r : \mathbb{H} \to \mathbb{H}$ (more precisely, partial isometries). Which are ways to project elements of $\mathbb{H}$ to either the left or right part of a splitted space $\mathbb{H}\oplus\mathbb{H}$, exactly like the rule $\oplus$ of linear logic. We require that they satisfy $l^*r = r^*l = 0$ and $l^*l = r^*r = 1$ where 0 and 1 are respectively the null and identity operators which are guaranteed to exist. The point is that l and r can be composed to encode paths (as operators) in a proof-structure where l means "go down left" and r means "go down right". The adjoint provides dual operations l^* for "going up left" and r^* for "going up right". In particular, the equation $l^*l = r^*r = 1$ is consistent with $\parr/\otimes$ cut-elimination:

[4] According to Girard, it took him one year to achieve the extension to exponentials.

[5] A mapping $u : \mathbb{H} \to \mathbb{H}$ such that $u(\mathbf{x}+\mathbf{y}) = u(\mathbf{x}) + u(\mathbf{y})$ and $u(k\mathbf{z}) = ku(\mathbf{z})$ for $k \in \mathbf{C}$ (the set of complex numbers).

[6] Because neither I do.

[7] Originally named p and q.

going down from a premise of a direction d (left or right) then going up in the premise of same direction d as in Figure 33.3 (the path has to be reversed to follow operator composition). As for $l^*r = r^*l = 0$, it means that it is invalid to not be consistent with the chosen direction of premise because such paths are not preserved by cut-elimination and the composition of operator is hence cancelling.

§34.5 It is possible to represent operators by matrices generalising the matrices associated with finite permutations. In MLL with finite permutations, boolean coefficient are sufficient: we have a coefficient 1 when there is an association $i \mapsto j$ in the permutation where i is the index of rows (source) and j the index of columns(target). The coefficient is 0 otherwise. An axiom link between two atoms 1 and 2 is represented by the permutation $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ which corresponds to the square matrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. As for cuts, they are represented with symmetric matrices since they are two-way linking of formulas. This corresponds to *partial symmetries* in Banach spaces. More generally, proofs can be represented by matrices of operators (recall that 0 and 1 are operators as well). Instead of the presence of a link, it is possible to encode the path with the partial isometries p and q, relatively to a conclusion. Operators can also be interpreted by more compact big matrices featuring information about cuts [Shi03, Section 3.4]. Concrete examples of matrices can be found in Pistone's works [Pis15].

§34.6 **Interpretation of exponentials.** The internalisation of tensor product provides a tensorisation of operators $u \otimes v$ such that $(u \otimes v)(\mathbf{x} \otimes \mathbf{y}) = u\mathbf{x} \otimes v\mathbf{y}$ allowing to attach (by pairing) and stack information. The rough idea is that it allows to add more information (that Girard calls "message space" [Gir11a, Section 19.3.5]) to paths in order to specify what I call a *copy identifier* which says whether we have a left or right copy (for the two branches of contraction link). The promotion turns an operator u (corresponding to a proof) into $1 \otimes u$ (corresponding to a proof in an exponential box) where 1 is the location of a potential copy identifier. Weakening is handled by adding 0 coefficient for a new atom in the matrix of a proof. Contraction is handled with operators $x \otimes 1$ where $x \in \{l, r, l^*, r*\}$ to specify that we are either entering or exiting a left or right copy of an atom (without altering the multiplicative part on the right). This is explained in Shirahata's paper [Shi03, Section 3.7].

§34.7 **Feedback equation and execution.** In the same fashion as in Figure 33.4, it is possible to define an equation in terms of operators for which the solution corresponds to the cut-elimination of a proof. This equation is called *feedback equation* [Gir06]. Imagine that we have an operator $u : \mathbb{H} \oplus \mathbb{H}' \to \mathbb{H} \oplus \mathbb{H}'$ representing axioms such that $u(\mathbf{x} \oplus \mathbf{y}) = \mathbf{x}' \oplus \mathbf{y}'$ for some $\mathbf{x}'$ and $\mathbf{y}'$ (it is the box of σ in Figure 33.4) and an operator $v : \mathbb{H}' \oplus \mathbb{H}'' \to \mathbb{H}' \oplus \mathbb{H}''$ representing another set of axioms such that $v(\mathbf{y}' \oplus \mathbf{z}) = \mathbf{y} \oplus \mathbf{z}'$ for some $\mathbf{z}'$ (it corresponds to the box of τ in Figure 33.4). We obtain the following feedback

equation system:

$$u(\mathbf{x} \oplus \mathbf{y}) = \mathbf{x}' \oplus \mathbf{y}'$$
$$v(\mathbf{y}' \oplus \mathbf{z}) = \mathbf{y} \oplus \mathbf{z}'$$

As in Paragraph 33.12, the equation can be simplified and we obtain:

$$u(\mathbf{x} \oplus \mathbf{y}) = \mathbf{x}' \oplus \mathbf{y}'$$
$$\sigma(\mathbf{y}') = \mathbf{y}$$

where σ is the partial symmetry associated with cuts. Similarly to the right picture of Figure 33.5, when it has a solution, the equation induces an operator $w : \mathbb{H} \to \mathbb{H}$ such that $w(\mathbf{x}) = \mathbf{x}'$. This operator is defined by the *execution formula*:

$$\mathtt{Ex}(u, \sigma) := (1 - \sigma^2)u(1 - \sigma u)^{-1}(1 - \sigma^2).$$

The expression $u(1-\sigma u)^{-1}$ is a fancy way to write $\sum_{k=0}^{\infty} u(\sigma u)^k = u + u\sigma u + u\sigma u\sigma u + \ldots$ which is the sum of all possible alternation between axioms and cuts. The expression is an operator algebraic version of the simplified plugging of Paragraph 33.12. The expression $(1-\sigma^2)$ is a way to filter the result in order to keep paths starting outside the space of cuts and terminating outside of cuts (so that we indeed represent a maximal path between two inputs/outputs). As explained by Shirahata [Shi03, 3.6], σ^2 is an operator projecting inputs into either the other side of the cut or 0. If $(\sigma^2)(\mathbf{x}) = \mathbf{y} \neq 0$, then there is a cut between $\mathbf{x}$ and $\mathbf{y}$. The operator $1-\sigma^2$ projects its inputs to themselves if they are not part of cuts (which is equivalent to applying the identity operator doing nothing) or 0 (which would cancel the whole path).

§34.8 **Orthogonality by nilpotency.** It appears that the execution formula is not always defined (the intuition is that it can loop) but it is defined when σu is *nilpotent*[8], *i.e.* there exists $k \in \mathbf{N}$ such that $(\sigma u)^k = 0$. This ensures that there is a point where the infinite sum $\sum_{k=0}^{\infty} u(\sigma u)^k$ can stop. This naturally defines an orthogonality relation between operators (without even trying to characterise a specific correctness criterion). We say that two operators u and v are *orthogonal*, written $u \perp v$, when uv is nilpotent. From this orthogonality relation (which is directly based on execution this time, unlike our original definition with cyclicity of finite permutations), it is possible to interpret formulas/types as set of operators, exactly as in Section 33. It has later been established that nilpotency is related to acyclicity of graphs, which characterises MLL+MIX correctness (and not MLL) [NS19, Theorem 42].

§34.9 **A mathematisation of algorithms.** Following this first technical paper on the GoI, Girard introduced a second paper [Gir16b] with more ambitions:

[8] Actually, as Seiller explained, weak nilpotency is sufficient [Sei12b, §4.1.17].

> “ *This paper is the main piece a general program of mathematisation of algorithmics, called geometry of interaction. We would like to define independently of any concrete machine, any extant language, the mathematical notion of an algorithm.* ”
>
> – *Jean-Yves Girard*

The geometry of interaction is no more meant to be a mathematical explanation of linear logic or of proof-nets but a step towards the redefinition of the relation between logic and computation. Because the execution formula is able to mathematically express the computation appearing in logic independently of any computational system, Girard defines an algorithm by a pair of operators (u, σ) which can be executed with $\mathtt{Ex}(u, \sigma)$. In order to consider more general computation, Girard wanted to represent untyped λ-calculus by studying more general solutions of the feedback equation using *weak nilpotency* [Sei12b, §4.1.17] instead of nilpotency. The difficult is then to find a setting where this works because $1 - \sigma u$ (appearing in the execution formula) is not invertible in general (and we need it to be invertible because the execution formula is defined by $(1 - \sigma^2)u(1 - \sigma u)^{-1}(1 - \sigma^2))$ under the assumption of weak nilpotency. It seems that this project of *mathematisation of algorithms* has been postponed but recently restored by Seiller and Naibo.

§34.10 **From operators to logic.** In his third paper on GoI [Gir95], Girard introduced a new simplified interpretation by comparing his execution formula with the execution of logic programs (*cf.* Section 23). In particular, he tries to extend the interpretation to MALL. Since this topic deserves its own section (in the context of this thesis), it will be presented later in Section 37. The fourth paper [Gir06] is dedicated to a study of the feedback equation and its solutions [Sei12b, §4.2.6]. Finally, the fifth paper [Gir11b] will take into account an explicit consideration of the notion of *address* or *location*. Solutions of the feedback equation are extended with the notion of *determinant of a matrix* in which the execution formula appears under the assumption of invertibility of $1 - \sigma u$ [Sei12b, §4.2.7]. Shifts to new mathematical frameworks such as Type II Von Neumann algebras will also be considered in order to keep the interpretation of logic (especially exponentials) with a generalised notion of determinant (Fuglede-Kadison determinant). Philosophically speaking, the approach is novel and explicitly stated in the abstract of the fourth paper:

> “ *the equation was essentially studied for those Hilbert space operators coming from actual logical proofs. In this paper, we take the opposite viewpoint, on the arguable basis that operator algebra is more primitive than logic: we study the general feedback equation of Geometry of Interaction".* ”
>
> – *Jean-Yves Girard*

In the abstract of the fifth paper, the claim is even stronger:

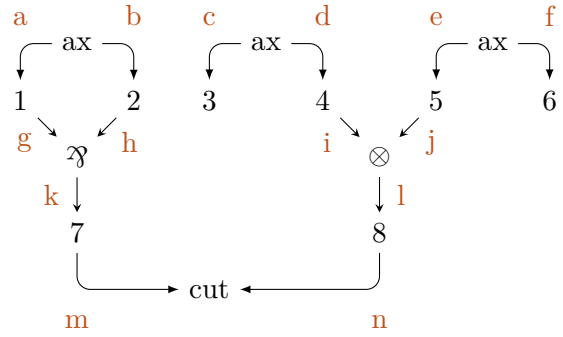


Figure 35.1: Proof-structure with labelled edges over an alphabet Σ of letters.

> *Geometry of Interaction (GoI) reacts against the absence of any satisfactory explanation for logic. The usual one is that of a symbolic calculus of truth values, which supposes that truth values pre-exist and formulas as well. [...] The aim of GoI is therefore to find a space where truth, commuting diagrams, etc. are no longer primitive and where dynamical processes (proof-search, rewriting, a.k.a. normalisation) are primitive.*
>
> – *Jean-Yves Girard*

35 Danos and Regnier's algebra of paths for MLL

§35.1 Danos and Regnier suggested a very accessible reformulation of Girard's GoI in their paper *"Proof-nets and the Hilbert space"* [DR95]. This approach directly applies the novelties of the GoI to original proof-nets. We are, again, interested in characterising the paths in proof-nets but more especially the *persistent paths* which are preserved by cut-elimination. These paths are the essence of proof-nets. An algebra $\mathtt{L}^*$ is defined to express weight on paths represented by words over $\{1, l, r, l^*, r^*\}$. If paths are not preserved, their corresponding weight will be cancelled. This is a simplified version of Girard's partial isometries l and r.

Paths in a proof-structure

§35.2 We consider proof-structures $(V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ extended with a labelling of edges. Since we defined proof-structures as hypergraphs, we have to consider a bijective labelling $\ell_p : V \times E \to \Sigma$ where Σ is a fixed alphabet of edge identifier. We write ℓ instead of ℓ_p when the labelling being used is obvious. An example of such labelled proof-structure is given in Figure 35.1.

§35.3 **Definition** (Path)**.** Let $\mathcal{S}$ be a proof-structure extended with a labelling $\ell_p : V \times E \to \Sigma$ for some Σ. A *path* of $\mathcal{S}$ is a finite word over $(\Sigma \cup \Sigma^{\mathcal{R}})^*$ where $\Sigma^{\mathcal{R}} := \{x^{\mathcal{R}} \mid x \in \Sigma\}$. The symbol $a^{\mathcal{R}} \in \Sigma^{\mathcal{R}}$ represents the reverted edge associated with a. In particular, it defines an involution $(\cdot)^{\mathcal{R}}$ such that $(a^{\mathcal{R}})^{\mathcal{R}} = a$ and $(ab)^{\mathcal{R}} = b^{\mathcal{R}}a^{\mathcal{R}}$.

If ${}_pi = a_1 ... a_n$ is a path then we must have that a_k and a_{k+1} are adjacent, *i.e.* if $\ell_p^{-1}(a_k) = (v, e)$ and $\ell_p^{-1}(a_{k+1}) = (v', e')$ then either $v = v'$ or $e = e'$.

§35.4 **Example.** An example of path from 1 to 6 in Figure 35.1 is $gkmn^{\mathcal{R}}j^{\mathcal{R}}e^{\mathcal{R}}f$ (read from left to right).

§35.5 Danos and Regnier distinguished several classes of paths in order to represent good and ill-behaving paths [DR95, Section 2.2]. A path is

- *non-bouncing* if it does not contain $a^{\mathcal{R}}a$ for any a (it does not retract his choice by going back in the previous point);
- *non-twisting* if it does not contain $a_1^{\mathcal{R}}a_2$ with a_1 and a_2 distinct premises of a same link (it does not go from one premise to another);
- *straight* if it is non-bouncing and non-twisting (the good paths we want).

§35.6 **Example.** The path $gkmn^{\mathcal{R}}j^{\mathcal{R}}e^{\mathcal{R}}f$ of the previous example for the proof-structure in Figure 35.1 is straight. An example of bouncing path from 7 to 7 is $k^{\mathcal{R}}g^{\mathcal{R}}g$. An example of twisting path from 8 to 8 is $l^{\mathcal{R}}i^{\mathcal{R}}j$.

Weight of a multiplicative path

§35.7 In this section, we associate weights with paths. These weights are words over

$$\{1, l, r, l^*, r^*\}$$

specifying which are the direction taken in binary connectives. In particular, we would like to follow the rules of multiplicative cut-elimination: preserved paths are the one consistent with directions by going down left (respectively right), going through the cut, and going up left (respectively right) on the other side. We first define the algebraic language of those weights.

§35.8 **Definition** (Involutive monoid $\mathtt{L}^*$)**.** Let $\mathtt{L}^*$ be the involutive monoid[a] generated by $\{\mathtt{l}, \mathtt{r}, 0\}$ and the following equations:

$$0x = x0 = 0 \qquad \mathtt{l}^*\mathtt{r} = \mathtt{r}^*\mathtt{l} = 0 \qquad \mathtt{l}^*\mathtt{l} = \mathtt{r}^*\mathtt{r} = 1$$

[a] Monoid with a set E, an associative multiplication operator $\cdot$ with neutral element 1 and an involution $(\cdot)^* : E \to E$ such that $x^{**} = x$, $0^* = 0$ and $(xy)^* = y^*x^*$.

§35.9 **Definition** (Regular path)**.** A path ρ in a proof-structure $\mathcal{S}$ is *regular* if and only if $\omega(\rho) \neq 0$ [DR95, Definition 4].

§35.10 Regular paths correspond to the paths which are not cancelled by the algebra of weight $\mathtt{L}^*$, hence which are valid *w.r.t.* multiplicative cut-elimination. We now associate weights with paths.

§35.11 **Definition** (Weight association)**.** Let $\rho = a_1...a_n$ be path in a proof-structure $\mathcal{S}$. The weight $\omega(\rho)$ associated with ρ is defined by $\omega(a_n)...\omega(a_1)$ where the weight $\omega(a)$ of a label a such that $\ell_p^{-1}(a) = (v, e)$ is defined as follows:

- if $a \in \Sigma^{\mathcal{R}}$, then $\omega(a) := \omega(a^{\mathcal{R}})^*$;
- if v is the left premise of e with $\ell_E(e) \in \{\otimes, \parr\}$, then $\omega(a) = \mathtt{l}$;
- if v is the right premise of e with $\ell_E(e) \in \{\otimes, \parr\}$, then $\omega(a) = \mathtt{r}$;
- $\omega(a) = 1$ otherwise.

Notice that the path is reversed. This is because of a technical detail: the natural reading order for paths is from left to right but algebraic composition is read from right to left.

§35.12 **Example.** In the proof-structure of Figure 35.1, we have:

- an illegal staight path $c^{\mathcal{R}}dilnm^{\mathcal{R}}k^{\mathcal{R}}h^{\mathcal{R}}b^{\mathcal{R}}agkmn^{\mathcal{R}}l^{\mathcal{R}}j^{\mathcal{R}}e^{\mathcal{R}}f$ from 3 to 6, of weight
$$\omega(fe^{\mathcal{R}}j^{\mathcal{R}}l^{\mathcal{R}}n^{\mathcal{R}}mkgab^{\mathcal{R}}h^{\mathcal{R}}k^{\mathcal{R}}m^{\mathcal{R}}nlidc^{\mathcal{R}}) = \mathtt{r}^*\mathtt{l}\mathtt{r}^*\mathtt{l} = 00 = 0;$$
- a legal regular straight path $c^{\mathcal{R}}dilnm^{\mathcal{R}}k^{\mathcal{R}}g^{\mathcal{R}}a^{\mathcal{R}}bhkmn^{\mathcal{R}}l^{\mathcal{R}}j^{\mathcal{R}}e^{\mathcal{R}}f$ from 3 to 6, of weight
$$\omega(fe^{\mathcal{R}}j^{\mathcal{R}}l^{\mathcal{R}}n^{\mathcal{R}}mkhba^{\mathcal{R}}g^{\mathcal{R}}k^{\mathcal{R}}m^{\mathcal{R}}nlidc^{\mathcal{R}}) = \mathtt{r}^*\mathtt{r}\mathtt{l}^*\mathtt{l} = 11 = 1 \neq 0.$$

§35.13 **Definition** (Execution formula)**.** The *execution* of a proof-structure $\mathcal{S}$ is defined by
$$\mathtt{Ex}(\mathcal{S}) := \sum_{\rho \in \mathtt{Straight}(\mathcal{S})} \omega(\rho)$$
where $\mathtt{Straight}(\mathcal{S})$ are the straight paths from conclusion to conclusion (hence maximal paths). This assumes that we have a disjoint union $+$ extending our algebra of weights.

§35.14 **Example.** Let $\mathcal{S}$ be the proof-structure in Figure 35.1. We have $\mathtt{Ex}(\mathcal{S}) = 1 + 1$ (two valid paths).

§35.15 As shown in Danos and Regnier's paper, the execution computes all straight and persistent (equivalently, regular) paths which are the paths preserved by cut-elimination.

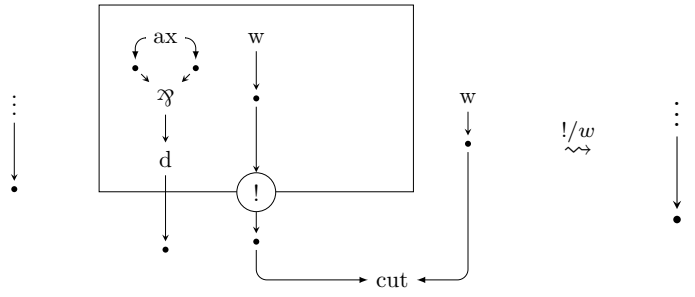


Figure 35.2: Valid path (traversing the ⅋ link) erased during cut-elimination for MELL proof-structures.

§35.16 **Theorem.** Let $\rho \in \texttt{Straight}(\mathcal{S})$ for a proof-structure $\mathcal{S}$. We have that ρ is regular if and only if ρ is persistent (not formally defined here).

Proof. Proven in Danos and Regnier's paper [DR95, Theorem 14]. □

§35.17 **The case of exponentials.** The difficult is to treat the behaviour of paths for boxes and structural rules. It has been investigated by Danos and Regnier [DR95] but I believe that it is better understood from the point of view of the *token machine* (also called *Interactive Abstract Machine* or IAM) [Lau01]. I only sketch the basic idea and do no develop more. We consider generalised ?-links as in Figure 29.14.

- Paths ending with a weakening link are not considered by the execution which only keeps paths from conclusion to conclusion.
- The only thing we require from dereliction links is that if go out of it from the top through a cut, we have to go to a box. And conversely, if we entering it from the bottom, we have to be coming from a box. We have to keep an information telling whether we are in an exponential path.
- Since boxes can be duplicated, we have to be able to tell which box we are talking about. In particular, depending on whether we are coming from the left or right premise of a contraction link, we are speaking about two distinct copies of a same box. A way to do that is to lift edge labels a into a tensor $1 \otimes a$ (as in Girard's algebraic GoI). Then we can compose $1 \otimes a$ by $k \otimes 1$ to get $k \otimes a$ with $k \in \{\texttt{l}, \texttt{r}\}$ telling that we are either speaking about the left or right copy of a potential box. We have imbricate boxes by tensoring again: $1 \otimes a \mapsto 1 \otimes (1 \otimes a)$.

§35.18 **No exact preservation of exponential paths.** I expose a little problem regarding the GoI in the context of exponentials[9]. The GoI studies cut-elimination only by considering set of paths to characterise proofs. However, in proof-net theory (which tries

[9] I thank Olivier Laurent who told me about this.

to mimic sequent calculus), some paths are valid but not preserved by cut-elimination. There are three sort of exponential paths:

1. the ones entering the box by the ! link and going out through it. These paths are preserved and there are not problems about it;
2. the ones entering the box the ! link but go out through an auxiliary door (the context ?Γ of an exponential box). In this case[10], paths are preserved in an alternative presentation of exponentials replacing the promotion rule by two rules: digging and functorial promotion (*cf.* Paragraph 27.21);
3. the most problematic paths are the one entering a box by an auxiliary door and going out from an auxiliary door. If the box containing this path interacts with a weakening, the path is erased even though it did not directly interact with the weakening and was valid. This is a collateral damage illustrated in Figure 35.2. The problem is that the GoI does not see that and it will appear in the result of the execution. In the other cases (dereliction and contraction) there should not be any problem.

This shows that the GoI is unable to exactly capture the behaviour of proof-nets only by considering paths. For that reason, approaches based on GoI often forbid ?A formula in the conclusion of a sequent since they may correspond to formulas going out of a box without passing through a ! link. It is also possible to simply exclude weakening links. But this is not necessarily a problem since these *junk residuals* can simply be ignored. Another solution is to shift to the paradigm of *intuitionistic game semantics* where sequents $\Gamma \vdash A$ has a distinguished output. This limits the space of path and it is then impossible to enter and exit by an auxiliary door for the same path.

36 Token machine for the geometry of interaction

§36.1 Studying cut-elimination only from the exploration of paths in a proof-structure has for consequence that the graph rewriting of proof-structure is not necessary: the execution of proofs and programs can be done by a process of exploration of a structure. In particular, it is possible to reduce λ-terms without syntactic transformation. Actual duplication is not necessary as well. This suggests ideas of sharing used in optimal reduction for λ-calculus (*cf.* Paragraph 29.29). The GoI also constitutes a tool for the study of dynamical processes occurring in logic.

§36.2 Since typed λ-terms can be encoded with proof-nets (and untyped λ-terms by proof-structures), it is possible to provide an execution of λ-terms by a computation of paths in the corresponding structure [AL95, DR99]. In order to make it implementable and

[10] According to a discussion I had with Olivier Laurent.

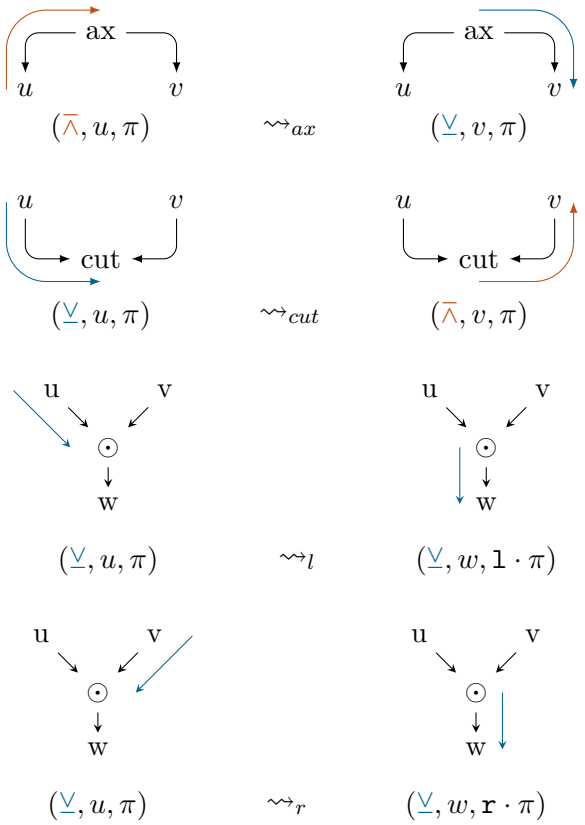
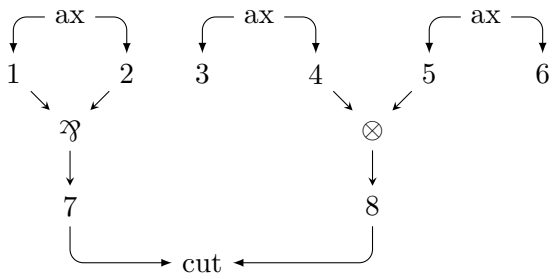


Figure 36.1: Transition rules of the IAM for MLL. For each transition $c \rightsquigarrow_x c'$, the reverse transition $c' \rightsquigarrow_x^{-1} c$ is valid as well with reversed directions for c and c'. The binary link $\odot$ corresponds to either $\parr$ or $\otimes$.

closer to computer science, this execution is usually defined by a *token machine*[11] which is a machine exploring a proof-structure and collection information during its travel [Lau01]. This machine is often called *Interaction Abstract Machine* (IAM). I describe the IAM for the multiplicative case.

§36.3 Let $\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ be a proof-structure. A *configuration* is a tuple (d, v, π) where $d \in \{\overline{\wedge}, \underline{\vee}\}$ is the direction of the current state, $v \in V$ the current vertex the machine is reading and π is a stack defined by the grammar $\pi ::= \emptyset \mid \mathtt{l} \cdot \pi \mid \mathtt{r} \cdot \pi$. A configuration is *initial* when it is $(\overline{\wedge}, x, \emptyset)$ where $x \in V$ is a conclusion of $\mathcal{S}$. It is *final* when it is $(\underline{\vee}, x, \emptyset)$ where $x \in V$ is a conclusion of $\mathcal{S}$. The IAM is defined by reversible transition rules between configurations given in Figure 36.1. Directions can be reversed with an involutive operation $(\cdot)^{\mathcal{R}}$ such that $\overline{\wedge}^{\mathcal{R}} = \underline{\vee}$ and $\underline{\vee}^{\mathcal{R}} = \overline{\wedge}$. For each transition $(d, v, \pi) \rightsquigarrow_x (d', v', \pi')$, there is an additional reverse transition $(d'^{\mathcal{R}}, v', \pi') \rightsquigarrow_x^{-1} (d^{\mathcal{R}}, v, \pi)$. A *run* is

[11] This use of the GoI is so famous that if you mention the GoI, a lot of people (if not most) who heard about it will say something like "Oh you're talking about the token machine?". I think it is not exaggerated to say that this application replaced Girard's original definitions and motivations.



$$(\overline{\wedge}, 3, \emptyset) \rightsquigarrow (\underline{\vee}, 4, \emptyset) \rightsquigarrow (\underline{\vee}, 4, \emptyset) \rightsquigarrow (\underline{\vee}, 8, \mathtt{l}) \rightsquigarrow (\overline{\wedge}, 7, \mathtt{l}) \rightsquigarrow (\overline{\wedge}, 1, \emptyset)$$
$$\rightsquigarrow (\underline{\vee}, 2, \emptyset) \rightsquigarrow (\underline{\vee}, 7, \mathtt{r}) \rightsquigarrow (\overline{\wedge}, 8, \mathtt{r}) \rightsquigarrow (\overline{\wedge}, 5, \emptyset) \rightsquigarrow (\underline{\vee}, 6, \emptyset).$$

Figure 36.2: Example of run from 3 to 6 with the IAM.

a sequence of transitions from an initial to a final one. We write $\rightsquigarrow$ for $\rightsquigarrow_{ax} \cup \rightsquigarrow_{cut} \cup \rightsquigarrow_{l} \cup \rightsquigarrow_{r} \cup \rightsquigarrow_{ax}^{-1} \cup \rightsquigarrow_{cut}^{-1} \cup \rightsquigarrow_{l}^{-1} \cup \rightsquigarrow_{r}^{-1}$ and $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$.

The rules for binary connectives presented push symbols and reverse rules pop symbols. The idea is that the combination of push and pop will cancel $\mathtt{l}^*\mathtt{l}$ and $\mathtt{r}^*\mathtt{r}$ so to reproduce the algebra $\mathtt{L}^*$. An example of run is given in Figure 36.2.

§36.4 **Machine for exponentials.** A machine for full linear logic can be found in Olivier Laurent's works [Lau01]. The idea of exponentials is that we add transitions for boxes, auxiliary doors and structural rules. During transitions, we have to keep the exponential box (relatively to boxes) we are in and an identifier of copy so that we speak about the right copy of a box. To give a simplified and incomplete idea of Laurent's IAM, we can add two stacks δ for the exponential depth and σ for exponential information. We have two injections $left : \mathbf{N} \to \mathbf{N}$ and $right : \mathbf{N} \to \mathbf{N}$ such that $\mathtt{img}(left) \cap \mathtt{img}(right) = \emptyset$.

- If we go down on a dereliction, we have $(\underline{\vee}, x, \pi, \delta, \sigma) \rightsquigarrow (\underline{\vee}, x', \pi, \delta, 0 \cdot \sigma)$. We add a special symbol 0 to specify that we are in an exponential path and expecting a box. The reverse transition pop the symbol because we have finished to explore a box and the machine exit the exponential zone created by the dereliction.
- A path ending on a weakening link will never be valid as it will never reach a final configuration.
- If we go down on the left premise of a contraction, we have $(\underline{\vee}, x, \pi, \delta, i \cdot \sigma) \rightsquigarrow (\underline{\vee}, x', \pi, \delta, left(i) \cdot \sigma)$. We have a symbol i meaning that we are in an exponential path and we split the index i to say that we are targetting the left copy of a potential box.
- If it the same for the right premise: $(\underline{\vee}, x, \pi, \delta, i \cdot \sigma) \rightsquigarrow (\underline{\vee}, x', \pi, \delta, right(i) \cdot \sigma)$.

- If we enter a box from the bottom of a ! link, we have $(\overline{\wedge}, x, \pi, \delta, i \cdot \sigma) \rightsquigarrow (\overline{\wedge}, x', \pi, i \cdot \delta, \sigma)$ meaning that we have an exponential identifier that we push on δ to say that we enter a box (are that the machine enters in the next depth).
- If we go out of a box but through an auxiliary door and not through the ! link of the box, we have $(\underline{\vee}, x, \pi, i \cdot \delta, j \cdot \sigma) \rightsquigarrow (\overline{\wedge}, x', \pi, \delta, \#(i,j) \cdot \sigma)$ where $\#(i,j) \in \mathbf{N}$ is an encoding of the pair (i,j) as a natural number (as in Paragraph 16.9). We remove a depth (because we exit a box) but change the "signature" of the exponential path by saying that we are in an exponential path identifier by the box i and the exponential path j.

§36.5 **Machine for lambda-terms.** Although λ-terms can be encoded as proof-structures, their translation always have a specific shape. It is possible to directly define a machine for λ-terms with more synthetic transitions. In particular, a transition for λ-terms corresponds to several transitions of the IAM. Such machines work by distinguishing a context and a part of the term which is read. And the syntax tree of the term is explored. It is possible to find definitions of that in Mazza's HdR thesis [Maz17, Section 3.3.4] (machine itself inspired by some notes of Accattoli and Dal Lago, according to Mazza).

§36.6 **Space-efficient computation.** Schöpp designed a type system capturing computation in logarithmic space, together with a related realisability interpretation inspired by the GoI [Sch06]. The idea later reappeared in the context of functional programming for sub-linear space with Dal Lago and Schöpp [LS10]. Using the GoI for λ-calculus has the particularity of executing by analysis of a static structure (the structure of the term). This seems space-efficient but the complexity is hidden in how the machine is treated (in particular, the data accumulated in stacks).

37 Alternative approaches

§37.1 We have already seen that there are several ways to approach the ideas proposed by the GoI. With operator algebras, with persistent paths of a proof-structure or with an abstract machine exploring a proof-structure. In this section I present notable alternative to these ideas. I present reformulations of the GoI with their own formalisms and problems but also related but not directly connected topics such as ludics.

Flows and wirings

§37.2 In his third paper about GoI, Girard propose a new presentation of his algebraic GoI by using ideas coming from term unification logic programming (*cf.* Section 23):

> “ *Geometry of interaction is most naturally handled by means of C^*-algebras; this yields surely more elegant proofs, but it obscures the concrete interpretation. So we prefer to follow a down to earth description of the interpretation. An unexpected feature will help us : the C^*-algebras used can in fact be interpreted in terms of logic programming, since the basic operators are very elementary PROLOG programs, and composition is resolution!* ”
>
> – *Jean-Yves Girard [Gir95, Section 1.8]*

Hence, he remarked that what he was doing with operator algebra only used a small portion of the power of operator algebra. The interpretation of logic could equivalently be done with mechanisms of term unification similar to the ones of logic programming.

§37.3 **Dialects (idioms).** A novelty of this third paper is that Girard tries to extend his interpretation to MALL with what he calls *dialects* [Gir95, Section 1.6] (later renamed *idiom*). The idea is, similarly to the interpretation of exponentials, operators of a Hilbert space $\mathbb{H}$ are extended with an additional space of information $\mathbb{H}\otimes\mathbb{H}'$ where $\mathbb{H}'$ is a *dialect*, meant to be private. This is inspired by Girard's communication without understanding (*cf.* Paragraph 33.33) where logical entities have their own internal and private language. Technically speaking, the point is that we would like to superpose a proof of $\vdash \Gamma, A$ and a proof of $\vdash \Gamma, B$ to obtain a proof of $\vdash \Gamma, A \,\&\, B$. This is done by linking the two occurrences of proofs of $\vdash \Gamma$ to, a dialect associated with A and a dialect associated with B respectively. In case we have a proof of $\vdash A \oplus B$ coming from a proof of $\vdash A$, the proof of $\vdash \Gamma$ associated with A will react correctly to it but the interaction fails for the other.

§37.4 Apart from this interpretation of MALL that I do not detail, this third paper allows the study of the GoI only from term unification by interpreting links by *flows* and proof-structures by *wirings*. These notions have been developed by Bagnol [Bag14] and Aubert [AB14, AS16b, AS16a]. Since it will be useful for the understanding of the rest of this thesis, I formally introduce these new objects. I refer to the chapter on logic programming for basic definitions about term unification (*cf.* Section 23). We fix a signature $\mathcal{S}$ for terms.

§37.5 **Definition** (Flow)**.** A *flow* f is a pair of terms (t, u), written $t \leftharpoondown u$, such that $\mathtt{vars}t = \mathtt{vars}u$ (there exists relaxation with $\mathtt{vars}t \subseteq \mathtt{vars}u$).

It is possible to compose two flows $t \leftharpoondown u$ and $v \leftharpoondown w$ by a product using term unification:

$$(t \leftharpoondown u)(v \leftharpoondown w) := \theta\alpha t \leftharpoondown \theta w$$

where $\theta = \mathtt{solution}\{\alpha u \stackrel{?}{=} v\}$ for a renaming α such that $\mathtt{vars}\alpha u \cap \mathtt{vars}v = \emptyset$. The requirement of the renaming makes the two flows' variables independent: their variables are bound the flow they are in. We write $f \approx_\alpha g$ for two flows equivalent up

to renaming.

§37.6 **Example.** $(X \leftharpoonup f(X))(f(g(X)) \leftharpoonup X) \approx_\alpha (X \leftharpoonup f(X))(f(g(X')) \leftharpoonup X') = \theta X \leftharpoonup \theta X' = g(X') \leftharpoonup X'$ with the most general unifier $\theta = \{X \mapsto g(X')\}$.

§37.7 **Definition** (Wiring)**.** A *wiring* F is a set (or multiset or formal sum depending on the definitions) of flows.

The product of two wirings F and G is defined by:

$$FG := \bigcup_{f \in F, g \in G} \{fg \text{ when it is defined}\}.$$

We distinguish an absorbing wiring 0 such that $0F = F0 = 0$ for any wiring 0. The identity wiring 1_F is the set of flows $t \leftharpoonup t$ for all terms t appearing in F.

§37.8 Flows represent the directed flow of information, as in Girard's long trips. The idea of the interpretation in modern presentation[12] is that an atom v of a proof-structure is represented as a term $v(X)$. For instance, an axiom between u and v becomes the wiring $\{u(X) \leftharpoonup v(X), v(X) \leftharpoonup u(X)\}$. We will write such two-way flows as a single expression $u(X) \leftrightharpoons v(X) := \{u(X) \leftharpoonup v(X), v(X) \leftharpoonup u(X)\}$. A product $(u(X) \leftrightharpoons v(X))(u'(X) \leftrightharpoons v'(X))$ then corresponds to a composition of path. Switching flow graphs can be represented as well and it is then possible to formulate Girard's correctness criterion.

§37.9 The variable X is not necessary for MLL (we could simply work with constants) but it is useful for extensions to exponentials. The idea is that if we have a term $u(X)$ then it can be splitted into two terms $u(l(X))$ and $u(r(X))$. This corresponds to a use of contraction rule for atoms. If something wants to match terms of form $u(t)$, for instance some $v(t')$ through a cut $v(X) \leftrightharpoons u(X)$, then it has to re-used in order to match both $u(l(X))$ and $u(r(X))$.

§37.10 Let F be the wiring associated with the test of switching φ, and G be the wiring associated with the axioms of $\mathcal{S}$. We have that $\mathcal{S}$ is correct *w.r.t.* long trips if there exists a $k \in \mathbf{N}$ such that $(GF)^k = 1_{FG}$ (which corresponds to the condition of cyclicity).

§37.11 **Definition** (Execution of a wiring)**.** A wiring F interacting with a wiring G can be *executed* with an execution formula:

$$\mathtt{Ex}(F, G) := (1 - G^2)F \sum_{k=0}^{\infty} GF(1 - G^2)$$

[12] Which has actually never been clearly written but since the idea is used in transcendental syntax (which has a very similar interpretation), we will explain it directly in the next chapters instead of developing the interpretation with flows. References about flows can be found in Bagnol or Baillot's works [Bag14, BP99].

where $(1 - G^2)$ is the usual filtering of Paragraph 34.7 which has to be defined (this could be done by an algebraic way exactly as in Paragraph 34.7).

§37.12 **Unary Horn clauses.** The link with logic programs comes from the fact that a flow $t \leftharpoonup u$ may be seen as a Horn clause $P(t) \vdash P(u)$ for a superficial predicate P wrapping all terms of flows. The resolution rule (which can be seen as a cut-rule) connects the output of one clause to the input of another one. From $P(t) \vdash P(u)$ and $P(v) \vdash P(w)$, we get $P(\theta t) \vdash P(\theta w)$ where $\theta = \texttt{solution}\{\alpha u \stackrel{?}{=} v\}$. This is exactly the composition of flows. A whole program (set of clauses) then becomes a wiring. The execution formula corresponds to inferring all possible clauses from a given set of clauses, similarly to how we compute the answers of a query in logic programming.

Seiller's interaction graphs

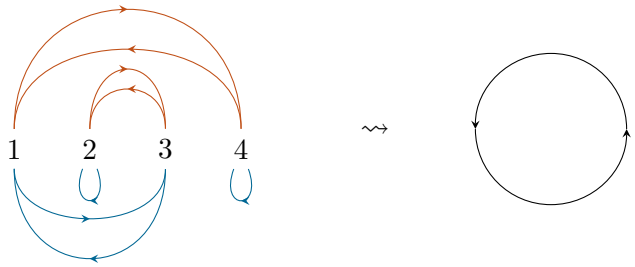
§37.13 Permutations over natural numbers can be represented by graphs. If we take this analogy seriously, then we can actually define the GoI by using interactive graphs which are connected along identical addresses. These graphs, introduced by Seiller [Sei16b] are a very convenient way to present the GoI on which various notions of graph theory can be applied. In particular, the execution is simply a computation of maximal alternating paths.

§37.14 For instance, we present the permutations of Figure 33.1a as a plugging between interaction graphs in Figure 37.1. We can then compute all maximal paths. All the ideas of GoI will then be about analysis and extension of these interactive graphs.

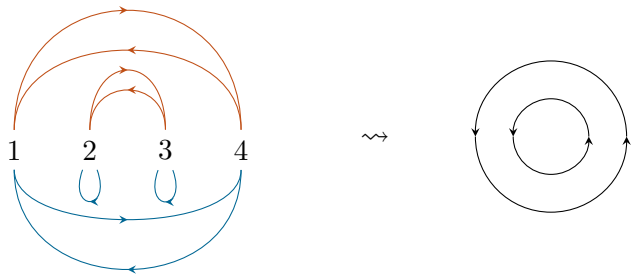
§37.15 **Definition** (Graph plugging)**.** Let $G = (V, E_G)$ and $H = (V', E_H)$ be two graphs. Their *plugging* is defined by a graph $G \square H = (V \cup V', E_G \uplus E_H)$ and a colouring function $\delta : E_G \uplus E_H \to \{0, 1\}$ such that $\delta(e) = 0$ when $e \in E_G$ and $\delta(e) = 1$ when $e \in E_H$. This makes the edges of E_G distinct from the edges of E_H.

§37.16 **Definition** (Alternating paths)**.** Let $G = (V, E)$ be a graph. An *alternating path* in G is a sequence of edges $(e_i)_{0 \leq i \leq n}$ for some n such that $\delta(e_i) \neq \delta(e_{i+1})$ for $0 \leq i \leq n-1$.

§37.17 **Weighted interaction graphs.** Given a graph $G = (V, E)$, it is possible to add weights to edges with a function $\omega : E \to \Omega$ where Ω is a monoid of weights with multiplication $\cdot$ and neutral element ε. The weight $\omega(\rho)$ of a path $\rho = (e_1, ..., e_n)$ in G is then given by $e_1 \cdot ... \cdot e_n$. By using these weights, we can analyse interaction graphs in a finer way. What we are interested in is weight of cycles which can help us recover the long trip criterion or various other orthogonality relations in a very generic way. Seiller defines the *quantification of cycles* between two graphs F and G as a function $[\![F, G]\!] := \sum_{\rho \in \texttt{Cycles}(F,G)} m(\omega(\rho))$ for a function $m : \Omega \to \mathbf{R}^+ \cup \{\infty\}$. We then have to choose a wise weighting function and a function m to have the interpretation we want. Notice that converging and diverging cycles are distinguished.



(a) Correct proof-structure.



(b) Incorrect proof-structure.

Figure 37.1: Connexion of interaction graphs and their execution.

§37.18 **Interpretation of MLL proofs.** Proofs are interpreted by *projects* $\mathfrak{a} = (a, A)$ where $a \in \mathbf{R}^+$ is the *wager*, that is a primitive weight associated with the project and A is a weighted graph (representing the proof-structure). We can then *measure* the interaction between two projects $\mathfrak{a} = (a, A)$ and $\mathfrak{b} = (b, B)$ with $\langle\!\langle \mathfrak{a}, \mathfrak{b} \rangle\!\rangle := a + b + [\![A, B]\!]$. Seiller defines his orthogonality relation as $\mathfrak{a} \perp \mathfrak{b}$ when $\langle\!\langle \mathfrak{a}, \mathfrak{b} \rangle\!\rangle \notin \{0, \infty\}$. Then the usual definition of types inspired by classical realisability can be done.

§37.19 **Genericity of interaction graphs.** The theory of interaction graphs has the advantage to be able to express various orthogonality relations (hence correctness criteria) in a same framework by using generic tools such as quantification of cycles and measure of interaction.

- By choosing $m(x) = \infty$ in the quantification of cycles, one obtains orthogonality by nilpotency;
- By choosing $m(x) = -log(1-x)$ one obtains Girard's orthogonality relation based on determinants.

Seiller is also able to express a property called *trefoil property* [Sei16a, Section 2.2] which generalises the adjunction property. This property guarantees that we have a sound model of MLL. Categorical interpretations with $*$-autonomous categories can be constructed as well [Sei12a, Section 3].

§37.20 **Graphings.** Seiller later generalised interaction graphs to *graphings* [Sei17]. Graphings generalise both usual finite graphs for MLL and Girard's use of operator algebras for full linear logic. Instead of directed edges between vertices (points), we have (measurable) maps over subsets of some (measure) space. This makes plugging more complex but give a very general framework to speak about cycles and paths. Graphings are expressive enough to naturally define various models of computation such as automata and Turing machines [Sei18] by actions over spaces of states or configurations.

Proofs as partitions of a set

§37.21 Permutations are very convenient if we consider the long trips criterion but it is not natural for Danos-Regnier correctness which focus on the undirected structure of correctness and not its directed flows of information. An alternative to permutations using *partitions of a set* has been mentioned by Danos and Regnier [DR89] and later been developed by Acclavio and Maieli [AM20, MP05]. Partitions are able to naturally represent Danos-Regnier correctness hypergraphs.

§37.22 **Definition** (Partition)**.** Let $X = \{1, ..., n\}$ be a finite sequence of natural numbers. A *partition* P over X is a set of sets $P = \{E_1, ..., E_k\}$ such that for $1 \leq i \leq k$, $E_i \subseteq X$ and:

- $\emptyset \notin P$;

Figure 37.2: Example of partitions associated with Danos-Regnier tests. The first correctness hypergraph corresponds to the partition $\{\{1, 2, r_5\}, \{3, r_6\}, \{4\}\}$ and the second to $\{\{1, 2, r_5\}, \{3\}, \{4, r_6\}\}$.

- all sets are disjoints, *i.e.* for all $i \neq j$, $E_i \cap E_j = \emptyset$;
- the reunion of all sets constitutes X, *i.e.* $\bigcup_{E \in P} E = X$.

§37.23 For instance, if we have $X = \{1, 2, 3, 4\}$, then some partitions of X are $\{\{1, 2, 3, 4\}\}$, $\{\{1, 2\}, \{3, 4\}\}$ and $\{\{1, 2\}, \{3\}, \{4\}\}$. You can think of it as cutting a cake representing X as you wish. You can make bigger slices if you want.

§37.24 **Interpretation of vehicles.** Vehicles are symmetric permutations over two natural numbers. They can be represented by partitions of binary set over a set of atoms X (notice that the direction is irrelevant for axioms). For instance, the two vehicles of Figure 33.1 are both represented by $\{\{1, 4\}, \{2, 3\}\}$. As for the vehicle of Figure 35.1, it is represented by the partition $\{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$.

- Not any partition corresponds to a permutation: the partition $\{\{1, 2, 3\}, \{4\}, \{5\}\}$ cannot be represented by a permutation;
- Conversely, it is possible to construct permutations such as $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{bmatrix}$ which cannot be represented by partitions.

The approaches are then concurrent and non-equivalent.

§37.25 **Interpretation of tests.** I use a variant of Acclavio and Maieli's pointed partitions [AM20, Definition 30] (sketched by Girard [Gir16b, Appendix A.1]) which I find more convenient to explain notions which will appear later in this thesis. A correctness hypergraph can be seen as a way to create a partition of the set of atoms. Tensor links will reunite atoms with $\otimes$ and the two switchings for $⅋$ are two different ways to separate atoms. Tests then induce several connected components. The partition associated with a test $\mathcal{S}^\varphi$ of switching φ is given by $P_\varphi := \{E_1, ..., E_k\}$ where $E_1, ..., E_k$ are uniquely associated with connected components $C_1, ..., C_k$ of $\mathcal{S}^\varphi$ and $e \in E_i$ when e is an atom in C_i. Moreover, each E_i is extended with a special value r_j for all conclusion j appearing in C_i. Examples of partitions for Danos-Regnier tests are given in Figure 37.2. The additional values r_j are called *roots* and they guarantee that we faithfully represent correctness hypergraphs by specifying where conclusions are located.

(a) Correct proof-structure.

(b) Incorrect proof-structure.

Figure 37.3: Example of connexions of partitions corresponding to correctness hypergraphs.

Figure 37.4: Example of tests for a non-sequentialisable MLL formula.

§37.26 It is possible to recover full correctness hypergraphs by connecting a vehicle with a test. Since vehicles and tests are independent entities in the GoI, we must patch things together by constructing a graph with edges linking identical atoms. If we connect the two tests of Figure 37.2 together with a same vehicle $\{\{1,3\},\{2,4\}\}$, then we obtain the two connexions of partitions illustrated in Figure 37.3a.

§37.27 **Definition** (Connexion of partitions)**.** The *connexion* of two partitions P and Q of a same set $X \subseteq \mathbf{N}$ is defined by a graph $G(P,Q) = (P \uplus Q, E)$ where vertices are the sets of P and Q, and there is an edge between $E_P \in P$ and $E_Q \in Q$ for each elements of $E_P \cap E_Q$.

§37.28 **Definition** (Orthogonality)**.** We say that two partitions P and Q are *orthogonal*, written $P \perp Q$, when $G(P,Q)$ is a tree containing all roots (or a forest if we are interested in MLL+MIX correctness).

§37.29 The fact that graphs of Figure 37.3a are trees means that the vehicle $\{\{1,3\},\{2,4\}\}$ passes the two tests and can be typed with a sequent $\vdash A^\perp \parr B^\perp, A \otimes B$ (from which the tests can be deduced). However, if we plug the tests to another vehicle such that $\{\{1,2\},\{3,4\}\}$ as in Figure 37.3b, then we obtain two connected component and a cycle. It means that the vehicle does not pass the tests.

§37.30 **Interpretation of types.** Types are constructed as in the GoI with permutations (*cf.* Section 33) except that we translate Danos-Regnier correctness hypergraphs instead of

$$\dfrac{\dfrac{\vdash X_1 \quad \vdash X_2, X_3}{\vdash X_1 \otimes X_2, X_3}\otimes}{\vdash (X_1 \otimes X_2) \parr X_3}\parr \quad \rightsquigarrow \quad \dfrac{\vdash X_1 \quad \vdash X_2, X_3}{\vdash F(X_1, X_2, X_3)}\, R_1$$

$$\dfrac{\dfrac{\vdash X_1, X_3 \quad \vdash X_2}{\vdash X_1 \otimes X_2, X_3}\otimes}{\vdash (X_1 \otimes X_2) \parr X_3}\parr \quad \rightsquigarrow \quad \dfrac{\vdash X_1, X_3 \quad \vdash X_2}{\vdash F(X_1, X_2, X_3)}\, R_2$$

Figure 37.5: Generalised rules corresponding to two ways for splitting atoms from $\vdash (X_1 \otimes X_2) \parr X_3$ for the generalised connective F.

Girard's switching flow graphs. As in Paragraph 33.30, it is also possible to construct generalisations of MLL formulas. An example proposed by Acclavio and Maieli (which initially comes from Girard and which has been reformulated by Danos and Regnier) is the following set of partitions (we omit roots but it does not matter):

$$\mathbf{G} := \{\{1,3\},\{2\},\{4\}\} \text{ and } \{\{2,4\},\{1\},\{3\}\}$$

which is a set of tests that cannot be expressed by $\parr$ and $\otimes$. It can be seen as taking the two tests of Figure 37.4 which is *hybrid* and cannot come from the same proof-structure. However, they *do* yields an orthogonal:

$$\{\{1,2\},\{3,4\}\} \text{ and } \{\{2,3\},\{4,1\}\} = \mathbf{G}^{\perp}$$

which can be considered as vehicles of type $\mathbf{G}$.

§37.31 **Interpretation of connectives.** It is possible to directly interpret *connectives* instead of formulas [Gir87b, Section 3]. It could have been defined with permutations but I find it more explicit with partitions. A *multiplicative connective* is defined by a parametrised expression $F(1,...,n)$ where $1,...,n$ represent occurrences of atoms (in natural order from left to right). Connectives are seen as collections of ways to *split* occurrences of atoms. Each splitting yields a generalised rule for that connective as illustrated in Figure 37.5. These generalised rules naturally yield partitions by how they split atoms. In Figure 37.5, the rule R_1 yields $\{\{1\},\{2,3\}\}$ and the rule R_2 yields $\{\{1,2\},\{3\}\}$. Connectives (including non-sequentialisable ones) are then associated with set of partitions (hence pre-behaviours). In particular, the two usual binary connectives $\parr$ and $\otimes$ are orthogonal special cases: $\parr(1,2) := \big\{\{\{1,2\}\}\big\}$ and $\otimes(1,2) := \big\{\{\{1\},\{2\}\}\big\}$, which correspond to reuniting and separating atoms.

§37.32 **Cut-elimination.** As far as I know, cut-elimination has never been expressed directly on partitions but it is possible to represent cuts exactly as for axioms. Examples of vehicles P (top) and cuts Q (bottom) interacting are given in Figure 37.6. Cut-elimination is defined as a graph rewriting procedure starting from $G(P,Q)$ which contracts edges

Figure 37.6: Example of cuts interacting with vehicles.

$$\frac{A_{[k,1]} \vdash \Delta_1 \quad \dots \quad A_{[k,n_k]} \vdash \Delta_{[n_k]}}{\vdash (A^\perp_{[1,1]} \otimes \dots \otimes A^\perp_{[1,n_1]}) \oplus \dots \oplus (A^\perp_{[m,1]} \otimes \dots \otimes A^\perp_{[m,n_m]}), \Delta} \ (+,\mathrm{k})$$

$$\frac{A_{[1,1]}, \dots, A_{[1,n_1]} \vdash \Delta \quad \dots \quad A_{[m,1]}, \dots, A_{[m,n_m]} \vdash \Delta}{(A^\perp_{[1,1]} \otimes \dots \otimes A^\perp_{[1,n_1]}) \oplus \dots \oplus (A^\perp_{[m,1]} \otimes \dots \otimes A^\perp_{[m,n_m]}) \vdash \Delta} \ (-)$$

Figure 37.7: Synthetic rules for linear logic. The use of negation ensures that we alternate between positive and negative formulas. For the positive rule $(+,k)$, in a bottom-up reading, we choose the k-th tensor of formulas (with $1 \leq k \leq m$) and split it. We have $\Delta = \Delta_1 \uplus \dots \uplus \Delta_{n_k}$. For the negative rule $(-)$, (which corresponds to a sequence of & over clauses of $\parr$ but on the left), from a bottom-up reading, all sequences of tensors of formulas are proven independently with a shared context Δ and all formulas of these sequences are put into the same space of interaction.

and remove the associated values in the two connected sets. However, it is not very easy nor interesting to define, so I omit the formal definition. In Girard's transcendental, a more convenient model extending partitions will be defined with a formal definition of cut-elimination.

Ludics

§37.33 Ludics has been imagined by Girard during his project of GoI in a the two-part paper *"The meaning of logical rules"* [Gir99, Gir00]. It has then be clearly defined in his paper *"Locus Solum"* [Gir01]. To quote Seiller [Sei12b, §4.2.1], if the GoI is seen as an abstraction of proof-nets removing as many superfluous details as possible, then ludics is the same but starting directly from sequent calculus instead. I choose to briefly present ludics because I find it interesting by itself and because it actually inspired Girard for his fifth paper on GoI where the notion of *location* takes an important role. It also contributed to the philosophy of the GoI and Girard's transcendental syntax which came after. For more details, an introduction to both linear logic and ludics has been proposed by Curien in two parts [Cur05a, Cur05b].

§37.34 **Focalisation.** Ludics is related to the idea of logic-as-game (*cf.* Paragraph 28.3). A sequent calculus proof is seen as a sequence of moves between two opponents and only one wins. It sounds fine except that the two players are not very explicit. Starting

$$\frac{}{\vdash \Gamma}\ \text{Ⱶ} \qquad \frac{\cdots \quad \xi.i \vdash \Delta_i \quad \cdots}{\vdash \Delta, \xi}\ (+, \xi, I) \qquad \frac{\cdots \quad \xi.I \vdash \Delta_I \quad \cdots}{\xi \vdash \Delta}\ (-, \xi, \mathfrak{R})$$

Figure 37.8: Rules of ludics. We have $i \in I$ a set of indexes, $I \in \mathfrak{R}$ and $\Delta_I \subseteq \Delta$. The symbol ξ represents the subject of the rule (which will be splitted into sublocations).

$$\frac{\xi.1 \vdash \alpha \quad \xi.2 \vdash \beta}{\vdash \xi, \alpha, \beta, \gamma}\ (+, \xi, \{1,2\}) \qquad \frac{\xi.\{1,2\} \vdash \alpha, \beta \quad \xi.\{1,3\} \vdash \alpha}{\xi \vdash \alpha, \beta, \gamma}\ (-, \xi, \{\{1,2\},\{1,3\}\})$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \xi.2.\{1,2\}}\ (+, \xi.2.\{1,2\}, \emptyset) \quad \dfrac{}{\vdash \xi.2.\{3\}, \alpha}\ \text{Ⱶ}}{\xi.2 \vdash \alpha}\ (-, \xi.2, \{\{1,2\},\{3\}\}) \quad \dfrac{\dfrac{}{\vdash \xi.3.\{7\}}\ \text{Ⱶ}}{\xi.3 \vdash}\ (-, \xi.3, \{\{7\}\})}{\vdash \xi, \alpha}\ (+, \xi, \{2,3\})$$

Figure 37.9: Example of application of rules in ludics. In the application of $(+, \xi, \{1,2\})$ and $(-, \xi, \{\{1,2\},\{1,3\}\})$, notice that the distribution of context is free. It is possible to simulate a weakening by providing a partial set of addresses for the positive rule so that we run out of addresses and are not being able to place every formula in a branch. As for the negative rule, it happens because all contexts of branches Δ_I must be included in Δ but they do not have to be disjoint. So it is possible to forget some locations such as γ. As for the last example, notice that we run out of addresses in the application of $(+, \xi.2.\{1,2\}, \emptyset)$.

from linear logic, we have already seen that connectives could be divided into positive (irreversible) and negative (reversible) connectives such that proving can be seen as applying all negative rules then positive rules. This idea initially comes from Andreoli [And92]. A proof is said to be *focalised* when it is an alternation between sequences of only positive or only negative rules. This yields a logical system called *hypersequentialised sequent calculus* [And92, Gir99]. The two players are therefore represented by positive and negative application of rules.

§37.35 **Synthetic rules.** In ludics, we consider *synthetic rules* (that I present in a bilateral fashion): one positive rule generalising $\otimes$ and $\oplus$ and one negative rule generalising $\parr$ and & (*cf.* Figure 37.7). Starting from that, ludics try to abstract from all the logical content to obtain interaction between alogical entities (exactly like proof-structures, permutations or partitions). We could use a monolateral definition of synthetic rules with $\parr$ and & for the negative rule but it would make the two rules structurally similar whereas the bilateral version exhibit structural differences as shown below.

§37.36 **Only location matters.** Now forget everything you assume about logic. Forget formu-

las. Sequent calculus rules are only about sequential manipulation of addresses. What we have done is dividing a formula into several disjoint locations. Even forget the sacred axiom rule which says that two locations are related. It is then only possible to *copy* from a A location to another A' which is a sort of axiom but impossible to say that two locations refer to the same thing[13]. Because axioms are forbidden, proofs have now infinite height. But because we can always splits the current context into sublocations, proofs also have infinite width. If the interaction between two proofs by cut is a game or a dialogue then we have the possibility to never finish. A proof is completed either with a rule ($\maltese$) called *Daimon* so that a player can abandon and stop, or if it cannot split any more. Proofs are called *designs* and use the rules of Figure 37.8. Simple examples of applications of rules taken from Seiller's master thesis[14] are given in Figure 37.9.

§37.37 **Chronicles and interaction.** A technical problem occurs: since the distribution of context is free, it is possible that two proofs only differ from how they manage their context. However, there is no way to distinguish these proofs by interaction. For that reason, we instead use *chronicles* which are trees of rule application satisfying some conditions such that the alternation between positive and negative rules or the subject of rules being all distinct. The *interaction* (representing cut-elimination) between two designs with root of opposite polarity is given by a maximal alternating path (as in GoI) between them. This path represents a dialogue or play between the two designs.

§37.38 **Reconstruction of formulas.** Formulas can be reconstructed in the usual way (*cf.* Section 33) by considering set of designs as pre-behaviours. Correct formulas correspond to behaviours. We usually want interaction to be terminating, hence two designs are orthogonal when their interaction terminates.

§37.39 Ludics has been developed by Basaldella and Faggian to include exponentials [BF11]. Quatrini and Lecomte developed applications to linguistics [LQ09] (especially in the context of *pragmatics*) which have been presented in books such as Lecomte's *"Meaning, logic and ludics"* [Lec11] or Fouquéré et al.'s *"Mathématique du dialogue"*. One recent work of interest is Fouquéré, Pinto and Quatrini's studies of dialogue with schizophrenic persons using ludics [FPQ21]. Finally, Terui has proposed a computational variant of ludics [Ter11] which could serve as a tool to analyse automata theory and computational complexity.

38 Discussion: new insights on the notion of proof

§38.1 So, what GoI says about reasoning? First, that linear logic can be understood from very basic operations: connecting locations in space. Axioms and cuts are about plugging wires (*cf.* Figure 32.1). Of course, it is much more complicated beyond MLL but the intuition is still there.

[13] The same idea appears in game semantics with *copycat strategies*.

[14] https://www.seiller.org/documents/ludique.pdf

§38.2 An old and serious problem of formal logic was the problem of *justification of logic* (*cf.* Paragraph 12.4). A common solution was to resort to external explanations such as models or languages commenting logic. But what we were doing was delegating explanation of logical syntax to... another logical syntax. But what justifies the latter? Probably another syntax. If we are not concerned with foundations of logic, this can be enough. The BHK interpretation (*cf.* Paragraph 21.1) and relations between computation and logic in general provide alternative computational explanations: we have ways to *construct* logical objects and *use* them. This already tells us that reasoning is about constructing tools and using them. However, these tools have to be *justified*. In the CHL correspondence, although logical entities have a computational ground, the shape of proofs is still decided by a sort of book of "what proofs should look like". Of course, it is possible to use external mathematical tools and discover linear logic, then proof-nets. But how far can this reductionist approach go? Going from one book to another gives us new tools but does not necessarily enlighten us about the foundations of logic.

§38.3 **A uniform space of interaction.** In realisability interpretation (*cf.* Section 22), we can ground logic on almost anything providing it works. But in its original presentation, it still suffers from the conceptual problems described above: the objects are constructed but not justified apart from a priori normativity. Formulas represent computational behaviours. The case of classical realisability (*cf.* Section 22) is especially interesting because it puts λ-terms against stacks. Terms are justified by stacks and stacks are justified by terms in a dual and symmetric interaction. Hence, two classes of constructions are mutually justified. This suggests that logical syntax could justified by another logical syntax itself justified by the first syntax, hence "closing the space" and preventing the infinite loop of external semantic explanations.

§38.4 The GoI (which is not a reductionist approach but a true *change of format* as in classical realisability) offers a uniform space for linear logic where interacting objects are of same kind and same degree of freedom in their construction and interaction. In Girardian terms, this is a *monist approach* instead of a *dualist* one. We do not assume any separation between two interacting classes of computational entities but study the interaction occurring within the same space of computation.

§38.5 The monist approach can be seen as an ideal for an interpretation of Schütte's completeness proof [Sch56, BT09] in which a statement either has a proof or else a partial aborted proof can yield a counter model[15]. In the GoI, this counter model is turned into a pre-proof, exactly as proofs themselves. Meaning that we can construct both a pre-proof of A and a pre-proof of its negation $A^{\perp}$ and making them interact in GoI.

§38.6 **A general method.** The idea of GoI is generic. We start from a model of computation such as:

- finite permutations;

[15]Note that it looks like the term "monism" for logic arose with ludics.

- finite matrices;
- operators in some algebra;
- wirings (made of flows);
- interaction graphs and graphings;
- designs (in ludics);
- partitions of a set

and use a realisability interpretation for linear logic. This interpretation is related to a categorical model known as *double glueing with biorthogonality* [HS03]. They all have in common that they can natively express *linear objects* as wires linking points (except for ludics which is directly related to the sequent calculus, considered as already correct *w.r.t.* proof-net theory). We then have more of less sophisticated mechanisms to handle exponentials. The common point between these interpretations is that they see proofs as sort of paths/networks (except for ludics). It is then natural to consider alternative or exotic models of paths to generalise the notion of logic.

§38.7 **The existentialist hell.** In Girard's papers, *essentialism* in logic refers to the approach of assuming types as primitive constructions and then programs are constructions following some typing rules. This is also known as *Church typing* in reference to Alonzo Church. It is the opposite of *existentialism* (or *Curry typing*) where types are constructed after programs. The GoI, ludics and classical realisability can be seen as existentialist approaches whereas usual approaches of logic are essentialist (type theory, model theory, truth interpretations, ...). The advantage of existentialism is that we are able to provide a type to *any* program whereas essentialist approaches can only construct programs from some given types (for instance, $\lambda x.xx$ is excluded as a type in simply typed λ-calculus). There is nevertheless a problem with existentialism: undecidability (*cf.* Paragraph 22.3). We are not always able to type a given program Φ. In terms of realisability or GoI, this is due to the potentially infinite set of tests orthogonal to a program. In order to assert if $\Phi \in \mathbf{B}^{\perp}$ for some set of tests $\mathbf{B}$, Φ would have to pass infinitely many tests. We are not able to *name* things. As if we understood the concept of cat perfectly and studied it from all possible perspectives but could not even tell whether we had a cat in front of us[16]. Imagine if I was explaining the meaning of a word with all its possible contexts. The solution is to take inspiration from proof-nets where correctness is certified in an effective way with correctness criteria. This gives a *finite* definition by showing how our objects are *shaped*; what I like to call a *pictorial semantics*.

§38.8 **A space of decomposition.** What the GoI does is not opposing proof-nets or proof-structure against other objects as in classical realisability. Surprisingly, it rather shows that a proof-structure can *already* been seen as a connexion between two distinct and

[16] This occurred several times in the history of science (not for cats). For instance, researchers in biology were familiar with works related to living beings but are still having difficulties to properly define what a living being even is.

dual parts: the vehicle (top, axioms) and a test (bottom, syntax tree of sequent). This decomposition is not innocent because it says that the objects we were manipulating actually feature a *computational* and a *logical* part. It is because we have a monism instead of a dualism. The computational content of MLL proofs lies only on axioms since the ax/cut elimination identifies some atoms and the $⅋/\otimes$ cut-elimination only re-organises the connexions later pushed to the top. Moreover, proof-structures can be seen as *pre-typed* since the bottom part specifies constraints on axioms. It opens the idea that we can speak of programs and tests independently in the context logic and study how they are related. In this point of view on logic, we do not manipulate "proofs" but computational entities (vehicle) relatively to some objects materialising normative constraints over computation (tests).

§38.9 **Internalising correctness.** In proof-net theory, we start from proof-structures as general computational objects and discriminate correct ones by an *external* method by considering alterations of $⅋$ nodes (for the Danos-Regnier correctness). One achievement of the GoI is putting both proof-nets and what makes them correct in the same space of interaction. Hence what makes proofs correct is *internal* to them. This offers a solution to the problem of justifying logical correctness. However, once a computational entity is certified by some finite set of tests (Danos-Regnier tests for instance), how can we tell that it will behave as expected, *i.e.* that the tests are sufficient? This is one important subject of the transcendental syntax introduced in the next chapter.

Chapter 6

Towards a transcendental syntax

The transcendental syntax (TS) is a series of four papers recently introduced by Girard as a natural successor for the geometry of interaction. It is a complete re-structuration of logic *from* computation and not an extension of the Curry-Howard-Lambek correspondence which merges scientific practices. It is meant to be a philosophical and technical search for fully internal and finitist explanations of logic. It offers new tools for the analysis and criticism of logical notions by providing a new point of view on logic, computation and their relations by taking inspiration from a lot of topics such as classical realisability, logic programming and developments of linear logic (mainly geometry of interaction and proof-nets). It can be understood as a sort of *critical theory* for logic.

Ideas of TS has been scattered for a long time in Girard's writings.

- In the end of the chapter about proof-nets in his *"Blind Spot"* [Gir11a, Section 11.C.5], he writes:

  > “ *Sequentialisation is not a dogma, it is a tool, which enabled one to find the procedural contents of nets; [...] But, on the whole, one has nothing against the idea of non-sequentialisable nets, as long as one can manipulate them : the ultimate meaning of logic is this ability to manipulate.* ”
  >
  > – *Jean-Yves Girard*

  This emphasizes the idea of logic as exit doors stated in Paragraph 31.4.

- In the fifth paper on GoI, Girard remarks that negation internalises normative constraints:

  > “ *For instance, formulas do no proceed from the sky; they proceed from their own operationality. What can be internalised by means of the negation, which thus takes in charge logical normativity: before*

> *refuting, negation forbids.*
>
> *– Jean-Yves Girard* ”

- The recurrent idea of communication (*cf.* Paragraph 33.33) which is materialised with flows (*cf.* Section 37) will be a key point of the computational foundation of TS.

The first papers which can be as pre-figurations of TS are Girard's *“Geometry of Interaction VI: a blueprint for transcendental syntax”* [Gir13a] and the paper associated with his invited talk *“Three lightings of logic”* [Gir13b]. Technically speaking, the TS proposes:

- a philosophical and conceptual developments of ideas coming from the GoI, proof-nets and ludics;
- a Turing-complete computational basis for logic which I call *“stellar resolution”*;
- a reconstruction of linear logic from stellar resolution (inspired by his works on GoI);
- a new architecture of the logical activity;
- a new definition of first and second-order logic;
- new connectives for linear logic;
- an attempt at giving a better explanation of what happens in innocent operations such as using a deduction rule;
- an opening to new investigations about truth and the normative constraints of logical systems.

These novelties are only partially explored in this thesis.

39 Learning from the past

I recommend Michele Abrusci and Paolo Pistone's paper [AP14] on the Kantian nature of the TS project for a precise and statement of problems occurring in formal logic, the potential solutions offered by the TS and a description of its philosophical ambitions.

§39.1 **Linguistic turn.** In Abrusci and Pistone's paper, two main characteristics of mathematical logic are mentioned [AP14, Section 2], which are primitive definitions:

> “ *The necessity to provide, in advance to the definition of “a logic” , a formal description of the syntactic devices to be used within such a definition (i.e. a definition of what is to be considered a formal language and a formal*

system). ”

and linguistic explanations:

> “ *The fact that the rules and axioms which constitute, within a formal system, “the logic” under consideration, have to be validated by reference to an interpretation, that is, a function assigning specific values, belonging to a certain mathematical structure, to linguistic entities, so that the interpretations of those rules and axioms result in transformations preserving some of those semantic values (typically, truth).* ”

This implies that primitive and rigorous manipulation of syntactic objects coming from a shared conception of logic is a necessary condition for any logical discussion. Hence, logic would be purely based on linguistic practices. But even more than that: it implies an *impossibility* of analysing logic itself as it always escape any attempt. In particular, Pistone has mentioned [Pis15] former Wittgenstein's idea that *“it takes rules to justify the application of rules”*, making any explanation of logic circular.

§39.2 **Geometric/morphologic turn.** TS (technically based on GoI) can be seen as an answer to these points. GoI and proof-nets already provided answers but only TS clearly states interpretations and refinements of the technical advancements of GoI.

1. The GoI shows that it is not necessary to start from logical definitions to speak about logic. It is possible to start from a computational basis (permutations, operators, graphs, ...) to reconstruct logical objects. However, it is only in TS that this idea of reconstructing logic is pushed to the extreme, by trying to justify points that the GoI did not treat such as the axiom rule, equality, finite verification, Danos-Regnier correctness and more that will be explained in this chapter.

2. Proof-nets show that logical objects can be justified by *shape*. Given a computational object, correctness criteria provide objects (correctness graphs) able to characterise the “shape of logical objects” (indeed with respects to our practice of logic in sequent calculus). However, this operation is still external (tests not expressed as proof-structures). In the GoI, tests and vehicles are objects of the same kind which are able to internally explain logic. However, this explanation is only partial and a lot are still yet to be justified in logic (as explained in the first point).

§39.3 **A pictorial semantics.** The lesson is that if we trap formal logic into a linguistic world, any linguistic object is necessarily explained by another linguistic object. Think of a dictionary in which everything is defined by words which are themselves explained by other words. Ultimately, either the explanation is circular or we have to stop with some words left unexplained and simply assumed because we “know and understand the

words". A way to get out of this trap is to provide a *pictorial* explanation. You want to understand what a tree is? Forget your dictionary, I will show you few representative trees. This is what TS does by using *morphologic explanations* [Gir18a, Section 1.1.2]: the logical has a specific shape which can be made explicit by correctness criteria. Speaking about shapes breaks the separation between syntax and (external) semantics. This is because some classes of shapes (trees in forests) are already naturally present in our perception of reality that logical discussions are even possible (speaking about trees). Surprisingly, this looks like a return to... Aristotle (*cf.* Section 1) and (ironically) to a form of... essentialism (but a more refined one).

§39.4 **Archeofuturism.** TS is not a rupture with the past erasing previous conceptions of logic. We have to forget as much as possible what we know about logic but still with traditional logic in mind:

> "*Even if we want to construct logic from scratch, this can only be a reconstruction, which means that we roughly know what we are aiming at. For an obvious reason, by the way: logic is a very healthy activity, which only suffers from a metaphysical, prejudiced, approach, that of analytic philosophy.*"
>
> – *Jean-Yves Girard [Gir17, Section 1.6]*

Girard's first objective in TS was to reconstruct logic from a more mature version of proof-net:

> "*The general task is to reconstruct logic [...]. Which amounts basically at finding the definite version of proof-nets.*"
>
> – *Jean-Yves Girard [Gir17, Section 1.6]*

The idea is to justify some parts of tradition instead of totally denying it or suggesting new cultures of logic totally replacing the previous ones. The future of logic would then be constructed upon justified vestiges of the past and old intuitions (which led to traditions).

§39.5 **The blind spot of logic.** In his course *"The Blind Spot"*, Girard explains the title which reflects his own philosophy of logic:

> "*About the title: it is while revising the text (Summer 2005) that I noticed the recurrence of the expression 'blind spot'. The blind spot, this is what one does not (is not) see(n), and one does not even know that one does not see it.* – *Jean-Yves Girard*"

It simply means "beware the appearance". The logical concepts directly accessible to our intuitions hide a lot of mechanisms. We cannot content ourselves with simple explanations and justifications which would hide all the complexity of logic. In order to analyse the different blind spots of the past, Girard defines several levels called *hells* [Gir11c, Section 1.2]. Lower the level is, the more we see. In particular, you will remark that what changes is the complexity of the space of answers.

§39.6 **Girard's hells: level -1 (alethic).** Formulas are sort of questions for which proofs are answers. The level -1 corresponds to truth interpretations, modalities and models. The answer to a question is either yes (true) or no (false). We do not have more information. Proofs of $A \vee B$ coming from A or coming from B are not distinguished. They all have the same purpose of leading to the truth of $A \vee B$. All we have to do is to follow definitions of what being true means. This level is sensitive to incompleteness where a mismatch between truth and provability occurs.

§39.7 **Girard's hells: level -2 (functional).** We add the "why" to the answer. Answers are subject to constructions (categorical interpretations, CHL correspondence, intuitionistic logic) but this construction still follow some rules of "good practices" with a zoology of constructions. It is like having allowed and disallowed correctness tests and computational objects. Proofs are distinguished by their structure: a proof of $A \vee B$ coming from A is different from a proof coming from B. They correspond to two different programs (left and right injection) which do not behave in the same way. This is also at this level that we find Kleene realisability and similar realisability interpretation (although they are more free in the choice of computational basis, logic is still constrained).

§39.8 **Girard's hells: level -3 (interactive).** We add the "how" to the answer. The answer is explained by all its possible interactions. Answers are proofs in the orthogonal of a set of tests associated with the question-formula (the negation/orthogonality is seen as an exchange of players in game semantics). At this level, we have game semantics (*cf.* Paragraph 28.3), Krivine realisability (*cf.* Section 22) and some approaches of GoI in which logic is primitive (interaction abstract machine). Some criteria make our objects *correct* and they define the notion of provability of a formula:

- *winning strategies* in game semantics;
- *proof-likeness* in Krivine realisability (that a term t does not contain a continuation constant $\mathbf{k}_\pi$);
- *correctness criteria* for a given formula in the GoI,

but these conditions still follow the rules of a given logical system and we cannot go outside of it.

§39.9 **Girard's hells: level -4 (deontic).** This deontic[1] level makes explicit what makes possible the answer in a lawless world; what are its *conditions of possibility*. The root of

[1] Relative to duty and obligations.

Level	Answers	Instance
-1 (alethic)	true/false	model theory truth interpretations
-2 (functional/static)	functions	category theory CHL correspondence Kleene realisability
-3 (interactive/dynamic)	strategies logical paths/tests	game semantics Krivine realisability interaction abstract machine
-4 (deontic)	justified alogical paths justified alogical tests	ludics geometry of interaction graphings transcendental syntax

Figure 39.1: Girard's hells.

possibility is taken to be of computational and structural nature. Nothing is assumed, not even the axiom rule relating two occurrences of atoms. Not even the notion of occurrence itself. Every physical point of a proof is nothing but a location void of meaning. In this level, we make explicit all the logical constraints which were implicit in the previous levels. In particular, the use of logical rules assumes that the shape of our objects induces some expected behaviour by computational interaction. In order to obtain the previous levels, we have to *reconstruct* logical rules. In this level, we have the GoI in general and ludics but the transcendental syntax is the ideal instance of it.

§39.10 Girard's four hells are summarised in Figure 39.1.

§39.11 **The phantom of transparency.** After the idea of "blind spot", the idea of *transparency* constitutes Girard's next thesis [Gir16a]. According to Girard, what directed logic (especially around the crisis of mathematical foundations) was the wish for *absolute transparency*; the ability to see, know and understand everything through x-rays of knowledge. The wish of a transparent world with no secrets, that Girard associates with a dogmatic consideration of science (scientism). Science, by its wish for extreme rationalisation is trapped into systems as if we could only go from a chapel to another. By trapping ourselves into locked systems of thoughts, we leave no space for intuition[2] and doubts[3]. Girard claims that all works on logic until today show that absolute transparency in logic *is not possible* (in particular because of incompleteness and undecidability). It is impossible to directly access reality from immediate perceptions. The access to reality has to be subject to a construction made intelligible to us.

[2] What the *mystical* had the most, while being the polar opposite of reason.

[3] Some (often social and political) systems of thought leave almost no space to doubts or incomplete data, to the point that some causes and consequences are almost logically derived. For instance, the materialistic world view seeing capitalism as the root of all evil and problems. Another example is the very structured world view of some conspiracy theories.

§39.12 **Soft scepticism.** The refusal of absolute transparency leads to an unavoidable scepticism. However, this scepticism has to be measured. In TS, the approach is to start from as less assumptions as possible and try to justify what we could already do. In the case of proof-nets, we consider links between points without any meaning. It is even more radical than proof-structures which are implicitly typed/pre-constrained (*cf.* Paragraph 38.8). On the top of that, in order to fully recover (at least) MLL proof-structures, we must add explicit constraints such as the fact that axioms link dual occurrences. We will also be led to a bit of controversial revisionism by questioning the most elementary things and rethink foundations.

- What is a proof? What is a formula?
- What is truth? What is provability?
- What is a logical rule? What allows us to use logical rules?
- What is a program? What is an algorithm?
- What is the purpose of logic?
- Why does $A \Rightarrow A$?
- Why does $a = a$?
- ...

This scepticism which can sound extreme is still limited since we cannot doubt of everything (otherwise we would do nothing and just wait for our death).

§39.13 **The search for internal explanations.** TS searches for internal explanations. Such explanations have already been found in the GoI (*cf.* Section 38) but an old (unintended) pre-figuration that Girard especially likes comes from Herbrand [Her30] works on predicate calculus and unification. The idea is better understood from *herbrandisation* which is dual to *skolemisation* (*cf.* Paragraph 23.10) by eliminating occurrences of existential quantifiers preceding a universal quantifier. We write $A[x_1, ..., x_n]$ and $t[x_1, ..., x_n]$ for a formula A and a term t containing variables $x_1, ..., x_n$. The idea is that the formula $\exists y.\forall x.A[x, y]$ can be proven by selecting some $y := t$ such that $A[x, t]$ holds for any x. However, this t cannot depend upon x because $\forall x$ comes after $\exists y$. There is, however, a dependency between quantifiers. This can be written by an equation $x = f(y) = f(t)$. The point is that if t contains x (which is written $t[x]$), then we have the unsolvable equation $x = f(t[x])$. The meaning of quantifiers is given by internal dependencies, without any reference to an external semantics. TS will be based on this idea applied to a new version of proof-nets.

§39.14 **No direct access to reality.** TS assumes no direct access to reality (in contrary to semantic realism). In particular, in the GoI, logic has always been defined from a choice of model of computation: permutation, partition, graphs, operators etc. Logic

is always accessed through a *subjective medium*[4] providing an access to reality. There are indeed better choices but no absolute and definitive choice[5]. It is only from a given representation of reality that some patterns and structures will be distinguished and define the logical. On the technical side, TS begins by choosing an appropriate model of computation on which logic would be based. Since Girard uses a lot of inspiration coming from proof-nets, this model should feature the elementary mechanics of proof-structures while being as "large" as possible.

40 The logical avant-gardism of computer science

§40.1 In TS, we are not exactly looking for connexions with computer science but rather inspirations and roots *within* computer science. Girard shows (in some recent papers but more especially in his book *"Le fantôme de la transparence"* [Gir16a]) several times that simple ideas coming from computer science are enlightening for logic. I would like to develop and make explicit how. I personally believe that logic has a lot of things to learn from computer science and programming in general, even from the most elementary and practical notions[6]. This section emphasizes the idea of *logic as tool* (or Organon in Aristotelian terms).

§40.2 **The procedural content of logic.** First, from a computational point of view, it is possible to distinguish notions which are not distinguished by semantical explanations [Gir16a, Appendix B.2]. As explained in Paragraph 11.7, $\vdash A \Rightarrow B$ and $A \vdash B$ have the same semantics but computationally speaking, $A \Rightarrow B$ can be seen as a frozen function made of symbols. It can be erased or duplicated like any data, as if we were handling the code of a program instead of the program itself. However, $A \vdash B$, makes the function participate as a program. It puts the data of the function in the space of interaction of a sequent which can be subject to cuts and cut-elimination. The formula $A \Rightarrow B$ is therefore of a static nature whereas $A \vdash B$ is a of a dynamic nature. In the GoI, the meaning of an object is given by its possible interactions.

§40.3 **Radical inclusivity.** In traditional logic and type theory, we tend to reject some objects because they are not "logical". Similarly to Curry's Urlogik (*cf.* Paragraph 15.8) which accepts paradoxes, there is less rejection in computer science. In programming, we have viruses which are self-duplicating malicious programs. Servers are finitely described programs which never terminate but are still essential[7]. While we are thinking about whether there are several or only one single logic, programming admits several

[4] This may be related to Kant's transcendental subject but I do not have enough philosophical knowledge to tell more.

[5] I asked Girard why the model of stars and constellation that he uses was the best analytic space and he told me that this was because there was nothing simpler. Apparently, he tried several things and it was the most natural formulation.

[6] Funnily, it was originally the converse. At the time of the CHL correspondence, logical ideas were discovered several years before their computational counterpart.

[7] Jean-Baptiste Joinet would probably say that there are "queer" λ-terms escaping type systems.

(sometimes weird) languages with their own characteristics, use and culture even though it seems there is only one notion of classical computation (an idea reinforced by the Church-Turing thesis). There are programs (compilers and interpreters) translating one language to another.

§40.4 **Test of programs.** In realisability and the GoI, the term of "test" is used to describe the computational opponents of programs. It makes even more sense with proof-nets where Danos-Regnier correctness hypergraphs (or Girard's switching flow graphs) are finite and effective ways to tell if a given set of axioms has a sound behaviour *w.r.t.* cut-elimination, *i.e.* that it is correct. This idea of test can be explored from the point of view of computer science. In software engineering, there are several tools to test programs. For instance;

- in *unit testing*, a portion of a program (typically a function used in a bigger program) is tested as an independent part. Typically by using representative cases for a partition of the domain (possible inputs);
- in *black-box testing*, a program is tested without having any knowledge of how the program is implemented or how it works. It is similar to tests on living beings such as psychology tests, laboratory tests on animals, Turing tests or Voight-Kampff tests in the movie *Blade Runner*;
- in *white-box testing*, we are aware of how the program being tested in shaped. It is similar to how circuits are tested;
- it is also possible to test the security of a software (for instance in *destructive testing*) by attacking it. We can check if a software is sensitive to some types of attack or if it behaves well in case of true failure. For instance, code injection is a type of attack exploiting a vulnerability of a program in order to inject unwanted instructions which will be executed[8].

§40.5 **File format.** The concept of *format* is essential in the philosophy of TS and can be easily understood from computer files [Gir16a, Section II.3]. For Girard, logic is a formatting of raw data which guarantees that logical entities will have a specific shape and interact as expected. It is exactly the same for computer files. We have raw data (which can be represented with binary or hexadecimal digits) and files can be associated with file extension (`.pdf`, `.exe`, `.png` or `.jpg`). File extensions indicate that a file containing raw data is organised in a particular way. For instance, picture files are mostly represented by a matrix of codes for the colour of each pixel of the picture. Pictures having the `.png` extension can represent transparent pixels while `.jpg` cannot. There are also extensions for the programming language of a source code. A file with `.c` extension is written in the C language and will not behave well when given to an interpreter for the Haskell language. Proof-structures are examples of raw data and correctness criteria

[8] I like to see Gödel's first incompleteness theorem as a sort of code injection on an arithmetic system. We take advantage of the freedom of arithmetic expression to inject Gödel's formula which, when interpreted, leads to a contradiction.

are examples of formats for proof-nets. Proof-structures can be given the extension `.proofnet` when it passes the tests of some sequent.

§40.6 **Hidden files.** In a computer, some files are hidden. The reason is simple: we do not need to see them but it is possible to see them if we really want. These files typically contain configurations read by some software. They are important for the architecture of the system but not necessarily for us. According to Girard [Gir20a, Section 1.3], these hidden files exist in logic and can be revealed (however, since no direct access to reality is assumed, this is only a representation accessed by cognition and not *the* hidden files). For existence, the contradiction **0** of linear logic is usually thought to be empty (it has no proofs) but in the TS, it has a computational content. This is not so surprising if we consider that it is possible to interact with erroneous programs. This idea is directly related to Girard's "blind spot". In his fourth paper on TS [Gir20a], Girard roots his notion of truth in the idea of *visibility*. Are true the objects which are visible.

§40.7 **Modules and libraries.** In his recent unpublished papers, Girard compares logical systems to bunkers [Gir19, Section 2]. The reason is that in a logical system, we established fixed ways to do things correctly. Different logical systems lives separately like hermetic clubs. One formula true in a system can be false in another and we avoid mixing systems to avoid inconsistency. According to Girard, logic should be more like modules in programming [Gir20b, Section 1.3]. In a same programming language, it is possible to import files containing several functions. These files are called *libraries* or *modules*. For instance, if we would like to use mathematical functions in the C language, we can write `#include <math.h>`. It is also possible to do partial importations. If we apply the idea to logic, it is like importing connectives from different logical systems and using them in a same context (a sort of "logical ecumenism"[9]).

§40.8 **Reverse engineering.** Reverse engineering corresponds to the process of opening up and dissecting a system to understand how it is functioning. It is typically used in software and electronic engineering but can be applied to other fields as well. For instance, it is possible to *disassemble* programs by translating the content of an executable program to an assembly language (a low-level language). By using this assembly representation of the program, it is possible to track some mechanisms and behaviours in order to partially understand the whole architecture of the software. It is then possible to modify or enhance some parts as we wish. It is a *reverse* approach in the sense that, usually, we first design the architecture of a system then implement it but in the case of reverse engineering, the system is already there and we would like to infer its underlying architecture. I personally like to see TS as a *reverse engineering of logic*[10].

[9] Expression suggested by Jean-Baptiste Joinet.
[10] We can also say *deconstruction*.

	Analytic / Brut	**Synthetic / Formatted**
A posteriori / Explicit	Constat	Usine
A priori / Implicit	Performance	Usage

Figure 41.1: Cognitive knitting of the transcendental syntax.

41 A new architecture for logic

All alternative references for this section are unfortunately in French. I recommend Girard's book *"Le fantôme de la transparence"* [Gir16a] (I think we ironically have to understand the TS first before understanding what Girard says in this book) and Sidney Congard's unpublished paper *"La logique face à l'arbitraire"* [Con22] which explains in simple terms the project and architecture of TS.

§41.1 TS suggests a new architecture to understand logic in light of computation. This architecture, called *cognitive knitting* by Girard, is apparently of Kantian inspiration[11]. This knitting is made of four interrelated categories (given in Figure 41.1) in which logic is about answering questions.

- The *analytic* is a world of lawless computation. This is where answers are expressed and where proofs/programs are shaped (only in their structure and not in their meaning). We can express "yes" or "no" but without context. It is objective.
- The *synthetic* corresponds to logic which has the role of *formatting* computation. It is where questions (formulas) are shaped by formatting answers (giving a context to computational objects to yield a proof). We provide a context for which words such as "yes" or "no" are answers to a question. It is subjective.
- These two categories are either *a posteriori* (explicit) or *a priori* (implicit). The correspondence with Kant[12] is that the explicit (a posteriori) *depends on experience* because we need a method to reach it from the implicit, while the implicit (a priori) does not depends on experience. I can construct a program out of my mind but I need the process of execution to reach the result.

Unlike Kant's epistemology in which "analytic" and "synthetic" are characteristics of assertions, in TS, they refer to spaces of objects (answers and questions) which can be realised in mathematics. In the following sections, we explain this cognitive knitting in details.

[11] I cannot tell how serious this is since I have never read a single line of Kant (but it seems that Girard neither, see https://www.youtube.com/watch?v=f7sT0J74pHI#t=3m18s).

[12] Probably. Because I have never read Kant.

42 Analytic space / Answers

§42.1 It can be intriguing that TS is introduced from the notion of *answer* instead of questions, because to have answers we must have questions first. How and what can we answer otherwise? Is it only provocation? Is it only an expression of anti-conformism? Not really[13]. Starting from answers is justified by the wish to justify logic from as less assumptions as possible. We would like to define a lawless and objective world of computation where logic does not exist. An anchor to all discussions. Starting from questions would mean that we already assume a format in which questions are expressed (a logical system for instance). Then the space of answers would necessarily be limited because of this constraint.

§42.2 Answers are objective because not engaged in a meaningful context. When I say "yes" or "no", it means nothing without a context. If I have a sound confirming an operation (coming from a machine for instance) and isolate it from its context, it is just a raw sound but if it happens after I do something on my computer or any other machine, it can mean something. Since "yes" and "no" are very limited answers, we would like the largest space of answers as possible. In particular, answers should be reducible to more explicit answers (similarly to the explicitation occurring in explanations).

§42.3 A model of computation has to be chosen for this space of answers. However, it is possible to describe Girard's architecture independently of this choice. We write $\mathfrak{A}$ for the analytic space which is a countable space of objects Φ, Ψ called *computational objects/entities*[14] (a term that I used a lot but which is only made clear here). The analytic space is understood from two categories: the Constat and the Performance which characterise requirements for the objects of $\mathfrak{A}$.

Constat and performance

§42.4 **Constat (result).** In the analytic space $\mathfrak{A}$, we must have objects corresponding to (final) *results*. Those results, called Constats are just there. Nothing happens with them. They are *explicit* (nothing is hidden). In the category of Constat, symbols can only be accumulated. Girard compares it with typewriters in which symbols can only be added without being subject to any dynamics (such as erasure or duplication). In the λ-calculus, it corresponds to terms in normal form which cannot be reduced. For instance $\lambda x.x$ or simply x.

§42.5 **Performance (dynamics).** The category of Performance corresponds to pairs (Φ, Ψ) with $\Phi, \Psi \in \mathfrak{A}$. Two objects can interact by *execution* (defined in $\mathfrak{A}$) and produce a new object $\mathtt{Ex}(\Phi, \Psi) \in \mathfrak{A}$. For instance, if we encoded circuits, Φ would be the structure of a circuit and Ψ its evaluation function (turned into an interactive object of $\mathfrak{A}$). Objects of

[13] Probably a little bit actually.

[14] They are sometimes called "epistate" in Girard's terminology [Gir11c].

the Performance are *implicit* since we need a method to extract the content they hide. Performance corresponds to reducible objects such as redexes of λ-calculus of the form $(\lambda x.M)N$ evaluated into $\{x := N\}M$ by β-reduction. Girard compares it with computer keyboards which can do unexpected things such as launching a program.

§42.6 Similarly to the λ-calculus, we can choose to not execute a reducible term. It means that any performative object can be seen as a Constat in the case we are not interested in evaluating it. Girard uses the analogy of a cheque (or bill) which can be used to pay something but which can also be exposed on a wall (in reference to Paul Erdős' cheques). Hence, in $\mathfrak{A}$, we should be able to make some parts impossible to reduce or to make more explicit: there is an end to the explicit.

§42.7 **Separating Constat and Performance.** What links Performance and Constat is the notion of execution reducing objects of the former to objects of the latter. It is a way to go from the implicit to the explicit. But what separates them is *undecidability* (*cf.* Paragraph 13.9) which can be reformulated as follows: Performance cannot be reduced to Constat. We cannot always foresee what will happen if a function interacts with an input. The computation could be undefined or looping.

The chosen one: stellar resolution

§42.8 There are several possible choices of model of computation. We could start from any Turing-complete (or even stronger) model of computation. But we cannot choose any model. We have to choose something as close as possible to proof-nets (which is considered as the right notion of proof by Girard but which as to be improved). This model should be:

- *self-executing* with local interaction so that we do not assume an external way to execute our computational objects. As Girard says, everything is on the table, including the table itself. We want to make every computational mechanisms explicit. For instance, if we had boolean circuits, the evaluation function would be implemented as well and explicitly interacting with the circuit itself. Such mechanisms of self-interaction appear in tile systems (*cf.* Section 17). The untyped linear λ-calculus could be a good candidate but because β-reduction is an external procedure and that we require self-interaction, it is not satisfying enough. We could argue that the execution is a global and external computational procedure but the point is that we would like our objects to perform a computation when locally plugged to each other;
- *partially executable* by preventing computation on some parts of our objects. In λ-calculus, weak strategies choose to not execute under abstractions: M in $\lambda x.M$ is not executed. In programming in general, it is useless to execute the inside of a function when it is not used;

	GoI	**TS**
Correctness criterion	Long trips	Danos-Regnier
Minimal model (MLL)	Permutations	Partitions
Nature of tests	Directed graphs	Hypergraphs
Unification model	Flows	Stellar resolution
Seiller's models	Graphings	?

Figure 42.1: Comparison between GoI and TS.

- expressing computation by *reducible links between addresses* so to interpret proof-structures and cut-elimination but also the idea of sending messages across a structure (as in the token machine for the GoI). This is also something which appears in automata theory in general where automata can be described as links between states in which messages are sent. This can also be related to concurrent computation with processes;
- *as less constrained as possible* and hence the model of computation should be untyped and be able to be "very free". We cannot choose something like the simply typed λ-calculus for instance. A possible interpretation is that the analytic space contains at least all computable functions. Ideally, it is non-deterministic, symmetric (no distinction between input or output which is a constraint), parallel and asynchronous (order in sequential computation can be seen as a constraint).

§42.9 The model considered by Girard [Gir89a] is a variant of flows and wirings (*cf.* Section 37) made of *stars* and *constellations*. I call this model of computation "stellar resolution" for its similarity with usual first-order resolution (*cf.* Section 23). Flows are binary directed expressions and stars are n-ary *undirected* flows. If flows are sort of unification-based directed graphs then stars are unification-based hypergraphs. Flows can naturally express Girard's long trips and stars can naturally express Danos-Regnier correctness. At a technical level, this makes TS a GoI for Danos-Regnier correctness. This difference between the two approaches are described in Figure 42.1. Notice that there is currently no equivalent to Seiller's graphings [Sei17] in the TS but we could imagine sort of *hypergraphings*.

§42.10 We leave the proper definition of stellar resolution for the next chapter and keep on with the description of Girard's architecture independently of a choice of model of computation.

43 Synthetic space / Questions

§43.1 The synthetic space is where questions are formulated, where everything takes its meaning. It is the logical language. If my answers were "yes" or "no" then I can formulate questions such that "Are you a man?" or "Are you a woman?". If the space of answers is large enough, we can even express more or less convincing proofs coming together with the answer. The activity of logic will revolve around the adequacy between questions and answers. Question should also be able to interact: if I have a positive answer to "Does A implies B?" and a positive answer to the question A then I should be able to produce an answer for the question B (this corresponds to the modus ponens).

§43.2 The synthetic is meant to *format* (or to give a format) to the analytic. A way to do that is to *classify* objects of $\mathfrak{A}$ by grouping them into sets. We write $\mathfrak{S}$ for the countable synthetic space made of sets $\mathbf{A}, \mathbf{B} \subseteq \mathfrak{A}$ called *pre-formulas*[15].

§43.3 There exists two ways to give meaning to objects of $\mathfrak{A}$: Usine (factory) and Usage (the use). They respectively correspond to a variant of Church and Curry typing in the context of TS.

§43.4 **Choice of point of view.** In the synthetic, we have to choose symmetric orthogonality relations $\perp \subseteq \mathfrak{A} \times \mathfrak{A}$ formalising a compatibility between two objects of $\mathfrak{A}$ in order to express that their interaction is sound. The soundness of interaction is subjective, it can be chosen freely. In particular, different choices lead to different logical interpretations. This orthogonality relation can be understood as a point of view on computational interaction.

Usage / Curry typing

§43.5 Usage corresponds to meaning as use. The meaning of $\Phi \in \mathfrak{A}$ is given by all its partners of interaction. It is like defining a word by all the contexts it can appear in. How do you know that you have a DVD player in front of you? Bring all the possible compatible DVD and insert them in the DVD player. If everything works fine then you do have a DVD player.

§43.6 It corresponds to Krivine realisability (*cf.* Section 22) and to the GoI interpretation (*cf.* Chapter 5). Given a pre-formula $\mathbf{A}$, we can define all its partner of interaction by another pre-formula $\mathbf{A}^\perp := \{\Phi \in \mathfrak{A} \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ which can be understood as a set of tests for $\mathbf{A}$. To check whether Φ is in the class $\mathbf{A}$ (DVD players for instance), then we have to test it against all elements of $\mathbf{A}^\perp$ (DVD). We are then interested in *formulas* (or *behaviours*) which are pre-formulas $\mathbf{A}$ such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$. Equivalently, it means that there exists some $\mathbf{B}$ such that $\mathbf{A} = \mathbf{B}^\perp$, meaning that $\mathbf{A}$ is fully characterised by a set of tests $\mathbf{B}$, or that $\mathbf{A}$ is *testable*. In the case of behaviours, $\mathbf{A}^\perp$ are tests for $\mathbf{A}$ and $\mathbf{A}$

[15] Or sometimes *dichologies* in Girard's terminology [Gir11c].

are tests for $\mathbf{A}^\perp$. It is only a matter of point of view: a DVD can be tested by the set of all DVD players. This interpretation has been formally implemented in Section 33 and can be generalised by starting from any model of computation. Usage is an *existential* (Curry) way to type objects because types (essences) come after the object being typed (existence).

§43.7 **Ideal concepts.** In Usage, it is possible to define concepts by interaction and yield other concepts by orthogonality. It is in Girard's Usage that the tensor product of two formulas $\mathbf{A}\otimes\mathbf{B}$, its dual $\mathbf{A}⅋\mathbf{B} := (\mathbf{A}^\perp\otimes\mathbf{B}^\perp)^\perp$ or the linear implication $\mathbf{A} \multimap \mathbf{B} := \mathbf{A}^\perp ⅋ \mathbf{B}$ are defined "by interaction". However, these notions are *idealised concepts* of tensor, par and implication. The possibility of having infinite tests makes formulas sort of god-given concepts which are perfectly defined and inferred from pure reason. However, this is also what makes them *too powerful* and unpracticable.

§43.8 It is in Usage that elimination rules of natural deduction are justified. Typically, the modus ponens is seen as an interaction between $\Phi \in \mathbf{A} \multimap \mathbf{B}$ and $\Psi \in \mathbf{A}$. It is possible to prove that under the right conditions, we have $\mathtt{Ex}(\Phi, \Psi) \in \mathbf{B}$ for any $\Phi \in \mathbf{A} \multimap \mathbf{B}$ and $\Psi \in \mathbf{A}$. Elimination rules are justified by their *use*, accordingly to old intuitions which are now made explicit (*cf.* Paragraph 12.4). See Pistone's works for more about how Girard's approach can serve as a justification of logical rules [Pis15].

§43.9 **The existentialist hell (again).** The problem (already stated in Paragraph 38.7) is that we cannot pass infinitely many tests in practice. It makes no sense to bring all possible compatible DVD to say that I indeed have a DVD player in front of me. It is like certifying a car in a factory by making it interact with all possible use cases including driving it forever to ensure that customers will be able to do the same. But how can you even sell such a product? It would not even get out of the factory. In the real world, we have no choice but to use *definitions* corresponding to a partial view of the object. We need *decidability* to be able to put words over things. If we are only interested in inferences on perfect, abstract and universal ideals, then we do not even need the orthogonality relation to be computable.

Usine / Church typing

§43.10 The solution to the previous problem of infinite testing is to limit tests to a wisely chosen finite sampling. This is the meaning as finite testing corresponding to Girard's Usine which is the main novelty of TS. It is like testing a DVD player with a chosen set of representative DVD or to apply some sufficient tests to a car. Such partial view on an object correspond to partial definitions which allow to actually *name* and *qualify* the object we manipulate. However this notion of definition should not be understood as a linguistical definition but as a *recognition of shape*, which is independent of language (otherwise we would return to the problems stated in Paragraph 39.1).

§43.11 Such finite testing typically appear in vaccines. The choice of tests is either too strict (and we miss some use cases) or too lax (and we may accept many irrelevant use cases). Either vaccines get too much time and efforts to be validated or we kill some people by mistake (but in the first case some may die as well). Another illustration is chemical/biological testing. For instance, in allergy tests, we put a patient in contact with some allergen and wait for reactions. This is also similar to how tests for viruses or pregnancy tests work. It is acceptable to use several orthogonality relations in case we are expecting different sort of outputs.

§43.12 Once we have a method for effective testing, it is possible to *certify* objects like how we certify a vaccine or a car which passes all our tests. Common examples of our everyday life (especially in France) are various labels attached to products such as "BPA-free", "Agriculture biologique", "Made in France", "Origine France Garantie". We trust some tests of our everyday life without always understanding what they are about. In particular, the "Made in France" label is rather lax and makes us think that our product is entirely made in France while it only has to be partially made in France. In the case of proof-nets, a proof-structure is certified with a label "Proof-net" when it passes all the tests corresponding to a given sequent.

§43.13 We define a *type* as a syntactic expression A (representing a type label) associated with a finite set of tests $\mathtt{Tests}(A)$ and fix a *decidable* orthogonality relation $\perp$. We say that $\Phi \in \mathfrak{A}$ is of type A, written $\Phi : A$, when $\Phi \in \mathtt{Tests}(A)^{\perp}$. Type checking (verifying whether $\Phi : A$ for a given A) can then be done in an effective way. It is an *essentialist* (Church) typing because types (essences) are defined *before* the objects being tested (existence). However, this is not plain essentialism because in TS, types will be *justified*: there will be some proof that the set of tests is sufficient to ensure that it is part of a corresponding formula. In terms of programming: that the tests are sufficient to guarantee the expected computational behaviour. Plain essentialism would correspond to certifying a vaccine but without verifying if the tests ensure that it will work well. We just have to trust our intuition. This is why I sometimes compare it with a sort of modern or rational essentialism. In proof-nets, type labels correspond to sequents. From a switching on a sequent $\vdash \Gamma$, it is possible to infer a set of tests (Danos-Regnier tests for instance) $\mathtt{Tests}(\vdash \Gamma)$ such that if Φ is a set of axioms, $\Phi \in \mathtt{Tests}(\vdash \Gamma)^{\perp}$ implies that Φ is a set of axioms for a proof-net of conclusion $\vdash \Gamma$.

§43.14 **Partial approximation of an ideal.** Another illustration of Usine that I especially like[16] is the design of... a chair (yes). When we design a chair, we have in mind all the possible (idealised) positions and use of the chair. But it is impossible to capture them all. All we can do is approximating it as much as possible. The best designers are the ones who can perceive the most accurately those use cases. Formally speaking, tests

[16] Which comes from a discussion I had with Paul Séjourné after he made me watch a video about Charlotte Perriand's work on modern design and architecture (I was not expecting that it could be related to logic).

of Usine are usually approximations of behaviours of Usage. If we have a behaviour $\mathbf{A}$, then we would like to find a finite set of tests $\mathbf{B}$ such that $\mathbf{B}^\perp \subseteq \mathbf{A}$.

§43.15 **In defence of prejudice.** My personal interpretation is that Usine is a place where we can express the structure of our prejudices. By prejudices, I mean intuition or unverified belief, which is directly accessible from immediate perception. Those prejudices can then be justified and confirmed (or considered too inaccurate). Tests usually come from a generic structuration of our experience and perceptions. In programming, when designing tests or specifications (or simply writing a program), we are trying to produce a material structuration of a mental representation. Although prejudices can be socially shared (like prejudices thought to come from ignorance), in our case, they have a computational form allowing (more or less certain) checking and confirmation. Usine corresponds to an *awareness* of the structure of our own prejudices.

Adequacy and certainty / Cut-elimination

§43.16 Adequacy is what links Usine and Usage. It may sound exaggerated but adequacy is the *heart* of the rational: the justification of our prejudices or the fact that the shape of our objects (given by the tests of Usine) guarantee their use (Usage). It is what conveys *certainty*. If I design a chair, then it would be (in some sense) irrational to design a chair far from any possible use. For instance a chair with spikes. Modern art or fashion, in particular, is often playing with the irrational by materialising it. It seems to me that what is rational is justified beliefs (where defining "justified" is the job of logic).

§43.17 In logic, adequacy corresponds to the cut-elimination theorem (which may sound very surprising). The idea is that in TS, $\Phi \perp \Psi$ formalises and identifies two notions:

- that Φ passes the test Ψ;
- that the interaction (cut-elimination) between Φ and Ψ representing vehicles of proof-structures behaves as expected.

This is because tests are seen as computational objects of the same kind as vehicles (similarly to the GoI interpretation). We obtain a new interpretation of the cut-elimination theorem. Assume that vehicles and tests are interpreted in a subspace $\mathfrak{P} \subseteq \mathfrak{A}$. The cut-elimination theorem says that for all $\Phi, \Psi \in \mathfrak{P}$, $\Phi \perp \Psi$ (their interaction by cut-elimination behaves well). This is not true in general (since vehicles are more general than proof-structures for which cut-elimination can be ill-behaving). This is where logical correctness appears: for all $\Phi, \Psi \in \mathfrak{P}$, if there exists $\vdash \Gamma$ and $\vdash \Gamma^\perp$ such that $\Phi \in \mathtt{Tests}(\vdash \Gamma)^\perp$ and $\Psi \in \mathtt{Tests}(\vdash \Gamma^\perp)^\perp$ (they are correct and dual), then $\Phi \perp \Psi$. In terms of proof-structures, if two proof-structures are dual and correct (they correspond to proof-nets) by passing correctness tests of some dual sequent then their interaction is sound.

"If objects have the right shape, they will behave well."

§43.18 **Relating answers and questions.** Usine and Usage correspond to two ways to tell whether an object is an answer to a question. However, only the adequacy justifies this relation by providing an effective verification that it corresponds to an approximation of an ideal. Notice a little change of terminology: in the GoI, ludics and game semantics, answers and questions are terms related to positive and negative connectives but in TS, they correspond to computation and logic. It seems to me that the two contexts are not related and that the terms "answers" and "questions" are too inaccurate in the situation of positive and negative connectives which rather corresponds to an interactive dialogue alternating between *expectations* and *assertions*.

§43.19 **Separating Usine and Usage.** In case we have a cut-elimination theorem, we are able to express an absolute certainty. However, some good logical systems do not enjoy a cut-elimination theorem. In that case, we have a gap between Usine and Usage meaning that Usine is never able to perfectly reach Usage. According to Girard, this limitation corresponds to Gödel's incompleteness theorems (*cf.* Paragraph 5.4) corresponding to the logical version of undecidability. Our choice of effective tests in Usine leads either to a lack of consistency or a lack of completeness (it is either too strict or too lax). We cannot always capture all use cases of logical objects with finite and effectively checkable definitions. To illustrate this fact, think of trying to define a musical notation for all possible sounds. A good (French) article explaining the gap between Usine and Usage in my opinion is the first section of Regnier's *"Les limites de la correspondance de Curry-Howard"*[17].

§43.20 **Reasonable certainty.** Formal systems not enjoying a cut-elimination theorem introduce some doubts. However, as Girard claims, certainty can still be reasonable in some cases. For instance, it is exaggerated to doubt of the foundations of mathematics. A huge number of theories have been established and used without any foundational fears. Mathematics lies on a solid ground even though it is not absolutely certain. The more we do mathematics, the more doubts disappear (but doubts will always remain) [Gir16a, Envoi: le doute].

Towards a justification of logical rules

§43.21 We would like to justify the rules of natural deduction. As explained in Paragraph 11.3, introduction rules correspond to definitions of symbols (how they are shaped) and elimination rules to use of symbols (how they interact). A prowess of sequent calculus is to reorganise this relationship between definition and use: all logical rules are introduction rules (either to the left or the right of $\vdash$ but in the monolateral case, it is sufficient to consider only the right). We then have the cut rule which materialise the use. This is made explicit in the translation of natural deduction into sequent calculus (*cf.* Figure 11.3) where translating elimination rules reveals an interaction by cut between two proofs.

[17] https://www.i2m.univ-amu.fr/perso/laurent.regnier/articles/ch.pdf

§43.22 This distinction between definition (cut-free proofs) and use (cut between proofs) coincides with Usine and Usage respectively. Sequent calculus rules can be analysed from proof-nets which focus on the structure of proofs and how formulas are related independently of the sequential application of rules.

- Usine will yield a translation of cut-free proof-nets. In this case, we are interested in how logical objects are shaped/defined independently of how they interact with other objects. It is the very principle of factories.
- Usage will yield a notion of interaction between cut-free proofs. Given two cut-free proofs, they can interact by using cuts which are sort of adapters linking points of the two proofs so that they can agree on common references.

44 Derealism / Animism

§44.1 In Girard's terminology, the term *derealism* (but also *animism*) refers to a subtle intertwining between computation (analytic/object) and logic (synthetic/subject) which is always present in the study of logic and computation. An ambition of TS is to go beyond this asymmetric distinction.

Logic and computation entangled

§44.2 I claim that most objects manipulated in the context of logic and computation (especially in the CHL correspondence) are hybrid objects with a computational and a logical part. In Section 38, I explained that the GoI was able to separate the computational (vehicle) from the logical (test) content of a proof. Ultimately, the vehicle itself is a sort of test for tests so the roles of test and tested is symmetric and interchangeable. Nonetheless, we are still able to make this distinction.

§44.3 If we look at the CHL correspondence, the correspondence between natural deduction and λ-calculus is not a link between logic and computation but an identification of logico-computational hybrid objects. Since proof-nets are able to encode typed λ-terms, it is possible to analyse λ-terms by separating their computational core (vehicle) from their logical core (what makes the shape of the term, influencing its computational behaviour). Similarly, proof-nets come from sequent calculus proofs which analyse natural deductions proofs. Proofs in proof-systems are then both of a computational (interaction) and logical nature (shape).

§44.4 If we want to be more radical, then untyped λ-calculus is actually already implicitly logical because it treats computation in the context of specific shapes (the structure of λ-terms). These shapes are sort of primitive constraints for computation. This is not so surprising since we were able to split not only proof-nets but also *proof-structures* which encodes untyped λ-terms. However, cut-elimination (ax/cut), corresponding to

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A \wedge B}\ \wedge i \qquad\qquad \dfrac{\begin{array}{ccc} & [A] & [B] \\ \vdots & \vdots & \vdots \\ A \vee B & \mathbf{C} & \mathbf{C}\end{array}}{\mathbf{C}}\ \vee e$$

Figure 44.1: Local (left) and global (right) natural deduction rules.

the execution of terms, is very primitive: as shown in the GoI (*cf.* Chapter 5), it is only about reducing links by identifying points (atoms of proof-structures), which can also be understood as sending messages or making a current flow. I do not count $⅋/\otimes$ or other logical cut-elimination cases because they are not purely computational but logical as they only reorganise links by specifying right paths for messages, hence enforce a structure for the computational current. They are implicitly part of tests (hence the logical part of proof-structures). Proof-structures are also logical because of their specific shape which makes the computational current flow in a specific way during cut-elimination.

§44.5 From the radical statement above, we obtain hints about the nature of logic and computation:

- computation is *physical*. It is about current freely flowing, the process of sending information without any constraint. It is related to time;
- logic is *structural*. It is about primitive or constructed paths in which the current of computation flow. It is related to space.

This makes the analytic space a bit logical but it is not a problem since it is meant to be an "anchor" space where any shape can be constructed (a playground for computational constructions). The synthetic space then associates some shapes with a name or a use.

§44.6 When manipulating syntactic objects (in mathematics for instance), both space and time naturally enter in the play together because they have to be materialised by a shape and are subject to (usually sequential) syntactic transformations (which need time). I conjecture that this is why those logico-computational hybrid objects naturally appear everywhere. Logic and computation are constantly and naturally mixed together and derealism studies how the various mixes occur in syntax.

Globality and locality in logical systems

§44.7 Mathematical logic historically mixed global and local operations. A lot of entities are generic without being defined as such. For instance, theorems of propositional calculus (*cf.* Section 6) such as $a \Rightarrow a$ are generic. They have the same interpretation for any valuation of a. Actually, it could be rewritten as a second-order formula: $\forall X.X \Rightarrow X$. Propositional calculus implicitly uses second-order quantification.

§44.8 In natural deduction, some rules are constructed in a *local* way by juxtaposition of objects already there. Figure 44.1 presents an example of local and global rule. The introduction $\wedge i$ is a typical local rule. We have A and B and put them next to each other to construct an expression $A \wedge B$. Some other rules are *global* or *generic* such as the elimination rule $\vee e$. The elements $A \vee B$, A and B are the elements already there but C is a generic formula. Any formula C can be chosen.

§44.9 **Subformula property and internal explanations.** These generic elements such as the C in $\vee e$ or the X in $\forall X$ invoke the huge *space of all propositions*. However, we cannot tell only from proofs or formulas what this space is made of. This space is externally formatted: this is what we mean by *logical system*. Therefore, generic statement implicitly refer to a system. This is related to the *subformula property* (*cf.* Paragraph 10.4). This property is a condition for internal explanation. It is preserved by local rule but not by global rules which summon something external. We obtain the following correspondences:

$$\text{Global} = \text{Generic} = \text{External} \qquad \text{Local} = \text{Specific} = \text{Internal}$$

In particular, by only considering local rules without any genericity, we have a fully internal explanation of logic which can be expressed out of any system. This is what Girard calls *system-free logic* [Gir20a].

§44.10 This reference to external structures can be found in the second-order equality defining $a = b$ as $\forall X.X(a) \Leftrightarrow X(b)$ (*cf.* Paragraph 8.7). The variable X refers to the set of all properties. A recurring criticism of Girard is that X cannot refer to properties such as "being on the left of $=$" or "being on the right $=$" which would distinguish between the two occurrences of a in $a = a$ and lead to $a \neq a$. But from what basis are we considering those properties logically wrong? An ambition of TS is to make explicit this structuration of the space of all propositions which is assumed in logical systems. This does not contradict "traditional logic" but relativise it by opening it to many choices of formatting the space of all formulas.

§44.11 From all these considerations, Girard adds to TS a distinction between:

- *apodictic*[18] (first-order logic) in which all rules are local and no reference to external systems exist. It is system-free. We can find connectives such as $\otimes, \parr, \multimap$ and limited exponentials;
- *epidictic* (second-order logic) in which we are summoning of the space of all propositions which has to be formatted by a logical system. We can find connectives such as $\oplus, \&, \forall, \exists$, neutral elements and full exponentials ! and ? (because of the global behaviour of boxes). It appears that $\oplus, \&, !, ?$ and neutral elements can be expressed with $\forall, \exists$ and first-order connectives.

[18] That Girard compares with anarchy because it works without a system.

Note that first-order logic usually means predicate calculus. Girard actually suggests a change of terminology which can be confusing: first-order and second-order logic in TS do not correspond to usual predicate calculus and second-order logic. In particular, because of its genericity, predicate calculus itself is subsumed in Girard's second-order logic (epidictic).

§44.12 The distinction between apodictic and epidictic is close to the distinction between nature and culture[19]. In apodictic, logic is emerging, natural and self-organising. It is where cut-elimination and adequacy are not problematic and where absolute certainty can be expressed. The epidictic is man-made and has to be wisely designed. Certainty is not absolute in this case but only reasonable at best.

Apodictics / First-order

§44.13 The difficulty of apodictics is to find it. If most logical objects we have are actually epidictic, then what can be apodictic? According to Girard, the only place where apodictics is clearly expressed is in proof-nets. If we take proof-nets without considering the generic nature of atoms/variables, they are indeed purely structural objects. It is only by external considerations that atoms can be replaced by other formula and hence by other proof-nets in the corresponding proof-net. This makes proof-structures independent from any logical system providing we forget genericity.

§44.14 In his fourth paper on TS [Gir20a], Girard defines a formula ㇇ ("fu") for first-order atoms. By using it, it is possible to establish a purely apodictic fragment of TS. In the same paper, Girard develops an apodictic fragment of TS and suggest notions of truth together with proofs of some axioms of Peano arithmetic.

§44.15 **How is absolute certainty possible?.** Good question. Because we cannot be sure of anything, right? It seems to me that what we mean by absolute certainty is that in the apodictic Usine, there are finitely many tests, orthogonality is computable and the adequacy is simple enough to only rely on simple combinatorics without even relying on parts of mathematics sensitive to incompleteness [Gir17, Section 4.4]. One can argue that it is exaggerated to speak about "absoluteness". Absolute certainty in Girard's terminology is a very reliable certainty.

Epidictics / Second-order

§44.16 Epidictics, because of its implicit external summoning of a system, is harder to justify. In his third article on TS, Girard proposed a way to define epidictics in TS. The structuration of the space of all propositions mentioned above is called *epidictic architecture*. It can be defined in both Usine and Usage.

[19] This nice illustration comes from a discussion I had with Pablo Donato.

§44.17 **Case of Usage.** The epidictic architecture for Usage is defined by considering a subspace $\mathfrak{E} \subseteq \mathfrak{S}$. The set $\mathfrak{E}$ has to be closed under a set of connectives $\{C_1, ..., C_n\}$, orthogonal, bi-orthogonal, cut and have to satisfy other essential properties which will be stated later in the thesis and which depends on what we want. Epidictic requires constraints representing the characteristics of the system considered. For instance, in the case of MLL, we can consider a closure under $\otimes$ so that for any $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{A} \otimes \mathbf{B} \in \mathfrak{E}$ and $\mathbf{A}^\perp \in \mathfrak{E}$ (which defines the $\parr$ connective): we are closing the space of questions. Although this looks like a limitation, this is what allows generic reasoning (we know what tools and shapes are available). As for quantifiers, they are interpreted with infinite intersection and union of formulas.

§44.18 **Case of Usine: universal case.** In the case of Usine, we have to consider finite tests for second-order quantifiers (which can generate other second-order connectives). In the case of $\forall X.A$, we have to ask a computational object to pass generic tests for all possible cases of formula that X can range over. For instance, in the case of MLL, we have the following generic tests obtained by taking $\texttt{Tests}(\vdash A)$ and replacing X and $X^\perp$:

- $X := \bullet$ and $X^\perp := \bullet$;
- $X := \bullet \otimes \bullet$ and $X^\perp := \bullet \parr \bullet$;
- $X := \bullet \parr \bullet$ and $X^\perp := \bullet \otimes \bullet$

where $\bullet$ represents an atomic formula (this will be clearer when defining the actual interpretation in stellar resolution later in this thesis).

§44.19 **Case of Usine: existential case.** For $\exists X.A$, we have to design tests but it is impossible to foresee what the existential witness will look like. However, in proof theory, the existential witness comes with the proof. This is where the derealist approach is exploited: testing is usually opposing a computational object Φ against a test Ψ. However, the computational object now comes with a test Ψ_T (called *mould*) for the existential witness T. We then test the union $\Phi \uplus \Psi_T$ against Ψ.

§44.20 **Difference between synthetics and epidictic.** Epidictic is indeed a formatting of computational objects. But then, how does it differ from synthetics? Synthetics is defined with a point of view (orthogonality) and tests (which can be finite or infinite). Epidictics is about meaning where synthetics is not sufficient. Typically, how to express that we "link dual atoms"? We need a human intervention to distinguish atoms and their dual and add the constraint that they must be linked in axioms.

An original proposal: epidictic and apodictic models of computation

§44.21 Remember that I claim that models of computation are actually logical. If the analytic space $\mathfrak{A}$ is the space of answers, then enforcing specific shapes and handling of objects of $\mathfrak{A}$ has consequence over the synthetic/logical space $\mathfrak{S}$. If we start from the lawless world of $\mathfrak{A}$ and try to define an automata or a circuit, we must define a starting point (initial

state or inputs) and describe how information flow in the structure (synchronised doors for circuits and linear run for automata). It makes models of computation restrictions over the space of information flows within a structure.

§44.22 An interesting idea which will be illustrated later in this thesis, is to first find what all classical models of computation have in common and then understand what makes their identity. I claim that we can distinguish between apodictic and epidictic models of computation.

§44.23 Apodictic models of computation are those which are self-organising and autonomous. Typically, tile systems such as Wang tiles are self-interacting objects with no external control. They only interact locally *w.r.t.* their primitive structures (colours on edges). Other examples such as the asbtract tile assembly model with cooperating tiles (used in DNA computing) will be presented in this thesis.

§44.24 Epidictic models of computation are those who need human intervention or external control. Typically, in a lawless world, we could start from the final state and loop eternally without consuming a word. There is a right path to take and we have to enforce it to lawless objects of $\mathfrak{A}$. This control over computation can be related to focussing in linear logic (*cf.* Paragraph 28.7).

§44.25 An interesting question is whether external control can always be internalised, *i.e.* is epidictic reducible to apodictic. My conjecture is that the answer is negative. But it is still interesting to study those cases in which control can be internalised (it is the case of "cooperative" tile systems).

45 Illustration: Transcendental Syntax applied to lambda-calculus

§45.1 The architecture of TS is not made for the λ-calculus but we can try to apply it to λ-terms in order to illustrate the ideas of this chapter[20].

§45.2 **Analytic of lambda-calculus.** For the analytic space, we can either choose Krivine's λ_c-terms and stacks (*cf.* Section 22) or Riba's λ-terms and contexts (*cf.* Section 22). I choose Riba's interpretation leading to simple types. We have $\mathfrak{A} := \Lambda \cup E$. Constat is the set of all normal terms extended with contexts. Performance are redexes and we consider β-reduction with any strategy. Note that TS splits λ-terms themselves into vehicle and tests in a homogeneous monist space.

§45.3 **Synthetic of lambda-calculus.** Usage corresponds to Riba's realisability interpretation which can reconstruct simple types as behaviours. Although simple types are constructed, the interpretation is open to more types by choosing another orthogonality

[20] This is something Pablo Donato wanted to see because he wanted to find a better way to explain and communicate ideas of TS without depending on Girard's formalisms. It would also be a way to makes TS more acceptable in front of the community of logic in computer science.

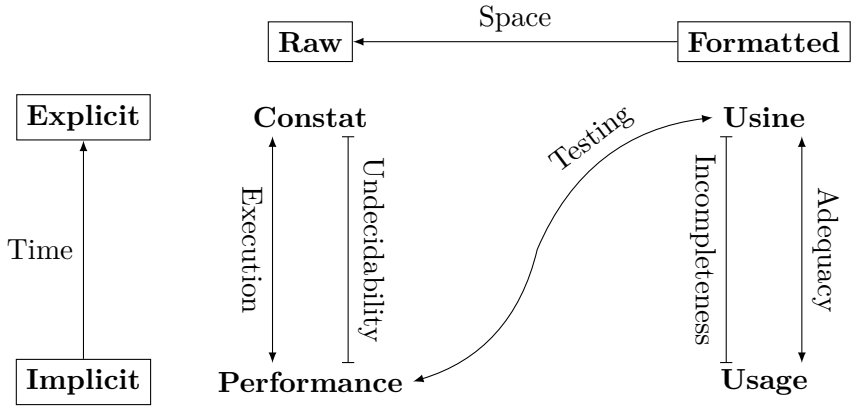


Figure 46.1: Relations between the four cases of Girard's cognitive knitting.

relation. Usine defines tests by finite sets of either terms or contexts. However, is it always possible to guarantee membership in a type by a finite set of tests? How to finitely check whether $M \in \alpha \Rightarrow \beta$ for some primitive types α and β? Moreover, in TS, contexts should be testable as well: everything should be justified. It seems that this space of testing is rather weak: it looks like the interpretation is not able to characterise a lot of interesting computational behaviours.

§45.4 **Apodictic and Epidictic of lambda-calculus.** We can now imagine what the apodictic fragment of our TS interpretation of λ-calculus would look like. The idea of the epidictic is to restrict $\mathfrak{A}$. We could for instance restrict to linear λ-terms by allowing only terms having exactly one occurrence of their argument. However, this does not translate in the synthetic space $\mathfrak{S}$: we cannot restrict the set of all behaviours so to obtain linear logic because interaction between terms and contexts is not expressive enough.

46 Discussion: is it a right understanding of logic?

§46.1 A fair question I encountered about TS is "why should we believe it is the right way to understand logic?". The TS is not meant to be *the* right answer to the foundation of logic. It does not even assume the possibility to directly access to the "true" content of logic but only to access a representation of it through a subjective medium (a choice of model of computation) as explained in Paragraph 39.14. However, it provides an interesting and different way to look at logic and computation.

§46.2 **Solid foundations.** The architecture of TS is grounded on two unavoidable limits:

- *undecidability* splitting analytics by separating Constat and Performance;
- *incompleteness* splitting synthetics by separating Usine and Usage.

The four cases of the cognitive knitting are also linked with well-known central notions:

- *execution* linking Constat and Performance;
- *cut-elimination* linking Usine and Usage.

These relations are illustrated in Figure 46.1. It shows that TS lies on cold and down-to-earth facts and not abstract ideals. On the technical side, TS is based on GoI which succeed a series of analysis from natural deduction to proof-nets. In TS, one still wish for as much conservativity as possible by rescuing as many concepts and results of mathematical logic (but not necessarily all). Hence, TS is more about reorganising the notions we already know and putting order in mathematical logic.

§46.3 **The three layers of logic and computation.** Explanations of how logic works in transcendental syntax do not need an infinite hierarchy of semantic level as in the usual linguistic spiral. Here we only have three layers:

- the analytic space $\mathfrak{A}$;
- the synthetic space $\mathfrak{S}$ which constraints the space of answers;
- a choosen *epidictic architecture* which constraints the space of questions (we choose what questions can be asked).

The epidictic architecture is a sort of meta-level above the relationship between analytic (object) and synthetic (subject) in logic. Although the expression "meta" is taboo in Girard's papers, it is not always problematic. We can, for instance, speak about *metapolitics* without any problem. The problem with the theory of metalogic is that we were barely touching what logic was really about. Metalogic did not say a lot of things about the mechanisms of logic. Hence, although we can argue that transcendental syntax still lives in some (mathematical, combinatorial or syntactical) metatheory, it is something we can tolerate. In the same fashion, computer scientists can build tools (programs), make them work and check whether they fit some specification (with a reasonable degree of reliability) without ending in a conceptual maze.

§46.4 In the next chapter, I formally present the stellar resolution which serves as a computational ground for TS. The chapter will be followed by a chapter thoroughly demonstrating the computational expressivity and power of stellar resolution in order to show that it is a model of computation of interest by itself.

Chapter 7

Stellar resolution

I refer to Appendix B for definitions related to term unification. These definitions are necessary to understand the definitions of this chapter.

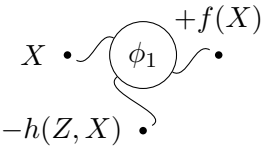
Stellar resolution is the model of computation used as a computational ground for transcendental syntax. It can be understood from several different points of view:

- it is a n-ary generalisation of Girard's model of flows and wirings (*cf.* Section 37);
- it is a flexible tile system (*cf.* Section 17) based on term unification and Robinson's resolution (*cf.* Section 23);
- it is an extension of Robinson's resolution with disjunctive clauses (*cf.* Section 23) (on which logic programming is based);
- in a more conceptual way, it is a model of computation describing computation as pieces of information flowing inside a graph-like structure.

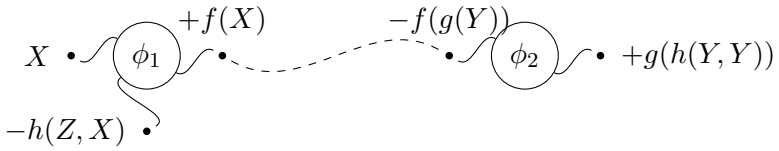
As we will see, stellar resolution actually generalises several models of computation such as: automata, circuits, logic programs, functional programs (by their translation to proof-nets), tile systems and more. I named it "stellar resolution" in reference to Girard's terminology of stars and constellations and for its link with usual first-order resolution[1]. At the end of the chapter, comparisons with ideas of logic programming are made.

The idea is that objects of stellar resolution will encode proof-structures. The interpretation is similar to the one of flows and wirings but the difference is that whereas flows are able to represent the long trip correctness criterion, stellar resolution naturally interprets the Danos-Regnier correctness criterion.

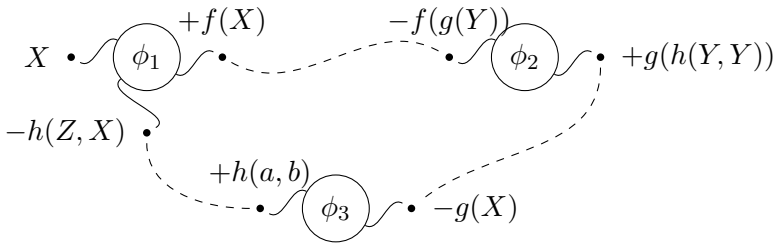
[1] But the main reason is because the name sounds pretty cool.



(a) Lone star. Poor guy has no friend.



(b) The star finds a partner and interact with it. They form a finite diagram. They communicate through dual rays.



(c) Another star is added to the diagram. The diagram gets bigger.

Figure 47.1: Connexion of stars.

47 Intuition behind stellar resolution

§47.1 In this section, I present the informal intuition behind the model of stellar resolution. The intuition can be understood from several ways depending on your background (as stated in the introduction). You may see it as logic programming, flexible tile system or extension of flows in GoI. Formal definitions are given in the next section. It is necessary to have knowledge on term unification (*cf.* Appendix B) and graph theory (*cf.* Appendix C) to understand this chapter.

§47.2 Figure 47.1a illustrates a star, which is a sort of flexible tile holding terms. It can also be written as a non-ordered finite sequence of terms $[X, +f(X), -h(Z,X)]$ where some function symbols are polarised (in $\{+,-\}$) or not. Several stars form a constellation. In a constellation, stars can interact by connecting to each other along dual rays as in Figure 47.1b. Dual rays are those rays which are polarised with opposite polarity ($+$ against $-$) and which are unifiable up to renaming (what I call α-unification), considering the compatibility relation $\frown$ between potentially polarised function symbols. Connecting stars along rays allows to construct tilings called *diagrams*. As shown in Figure 47.1c it

$$[g(X), +a(X), -b(X)]$$
$$|$$
$$[-a(f(Y)), +c(Y)]$$

(a) Step 1: we have two stars connected along two rays in a diagram.

$$[g(X), +a(X), -b(X)]$$
$$\theta = \{X \mapsto f(Y)\} \,|$$
$$[-a(f(Y)), +c(Y)]$$

(b) Step 2: we choose a link to contract and compute the solution associated with its equation.

$$[g(X), -b(X), +c(Y)]$$

(c) Step 3: the two stars interact by merging and erasing the rays related to the link.

$$[g(f(Y)), -b(f(Y)), +c(Y)]$$

(d) Step 4: the solution θ of the link is propagated to the resulting star.

Figure 47.2: Evaluation of diagrams by resolution. The two terms $+a(X)$ and $-a(f(Y))$ are dual and then can be connected to form a diagram.

is possible to construct bigger diagrams involving more stars. We are usually interested in connected and saturated diagrams which are diagrams which cannot be extended by connecting more stars to the diagram.

§47.3 **Dynamics.** Now that we have diagrams, it is possible to evaluate them by an edge contraction merging stars by annihilating the connected rays and propagating the solution of their equation to other rays. The steps of this evaluation are presented with an example in Figure 47.2. The execution of a constellation evaluates all possible connected and saturated diagrams. It is a generalisation of the execution of wirings.

§47.4 It is possible to imagine two ways of executing constellations. One we just described which constructs tilings and evaluates them. It has an *abstract* version which is more of a denotational nature and does not say how to construct tilings (they just mathematically pop out) and a *concrete* version which iteratively constructs diagrams. Another way called *interactive execution* starts from some stars and successively adds stars which directly interact on the fly with the available stars. It can be implemented more easily and

is more faithful to classical computation but do not exactly correspond to the previous natural notion of execution.

48 Stars and constellations

§48.1 Tiles are called stars and their flexible arms are *rays*. A tile set is called a *constellation*. Rays can contain special polarised function symbols analogous to the colours of Wang tiles. For instance, $+f$ and $-f$ are two *dual* terms. We expect terms such as $+c(f(X))$, $f(X)$, Y, $+d(X,-e)$ and $+c(f(+f(X),Y))$ to be rays.

§48.2 **Definition** (Polarised signature)**.** A *polarised signature* is a tuple

$$\mathbb{P} = (V, F, \mathtt{ar}, \supset, \lfloor\cdot\rfloor)$$

where $(V, F, \mathtt{ar}, \supset)$ is a signature (*cf.* Appendix B). The set F is defined as a partition $F_+ \uplus F_- \uplus F_0$ defining presence (polarity $+$ and $-$) or absence (polarity 0) of polarities over symbols. Function symbols of $F_+ \uplus F_-$ are said to be *coloured* (or polarised) and those of F_0 are *uncoloured* (or neutral).

The *underlying symbol* of a function symbol is defined by a function $\lfloor\cdot\rfloor : F \to F_0$ satisfying the following requirements:

- $f = \lfloor f \rfloor$ for $f \in F_0$;
- the restriction of $\lfloor\cdot\rfloor$ to F_+ (*resp.* F_-) is bijective, hence every neutral element of polarity 0 has a positive and negative version.

We define two maps attaching polarities to function symbols:

- $+ : F_0 \to F_+$;
- $- : F_0 \to F_-$;

such that $+$ and $-$ are inverse of $\lfloor\cdot\rfloor$, *i.e.* for all r, we have $\lfloor +r \rfloor = r$ and $\lfloor -r \rfloor = r$.

Finally, the compatibility relation $\supset$ is defined by $c \supset d$ if and only if $\lfloor c \rfloor = \lfloor d \rfloor$ and one of the following requirements hold:

- $c \in F_+$ and $d \in F_-$;
- $c \in F_-$ and $d \in F_+$;
- $c \in F_0$ and $d \in F_0$

§48.3 **Definition** (Opposite)**.** We first define the *opposite* $\mathtt{op}(f)$ of a function symbol. If $f \in F_0$ then $\mathtt{op}(f) = f$. If $f \in F_+$ then $\mathtt{op}(f) = -\lfloor f \rfloor$ and if $f \in F_-$ then $\mathtt{op}(f) = +\lfloor f \rfloor$.

We define the *opposite* $\mathtt{op}(r)$ of a polarised ray r by:

$$\mathtt{op}(c(r_1, ..., r_n)) = \mathtt{op}(c)(\mathtt{op}(r_1), ..., \mathtt{op}(r_n)).$$

§48.4 **Example.** Let $\mathbb{P} = (V, F, \mathtt{ar}, \frown, \lfloor\cdot\rfloor)$ be a polarised signature with $V = \{X\}$ and $F = F_0 \uplus F_+ \uplus F_-$ such that $F_0 = \{c, d\}$, $F_+ = \{+c, +d\}$ and $F_- = \{-c, +d\}$ (we omit arity). We have $+c \frown -c$ and $+d \frown -d$.

§48.5 **Convention.** We assume the existence of a polarised signature $\mathbb{P} = (V, F, \mathtt{ar}, \frown, \lfloor\cdot\rfloor)$ unless we explicitly use a specific one.

§48.6 **Internal colours.** Unlike the simplified description of the previous section, rays are not simply defined as terms with a polarity as prefix. For the interpretation of simple fragments of linear logic such as MLL, it is sufficient to consider polarised terms which indeed correspond to first-order atoms but if we are interested in interpreting Girard's second-order logic (epidictics), then rays with internal colours must be used [Gir18b, Section 4.1] (making it far more complicated than the dynamics of logic programming). The intuition is that rays with internal colours will be used for the interpretation of moulds (*cf.* Paragraph 44.19). Because these two class of rays are both used in different context, we will explicitly distinguish them.

§48.7 **Definition** (Rays)**.** A *ray* on a signature $\mathbb{P} = (V, F, \mathtt{ar}, \frown, \lfloor\cdot\rfloor)$ is a term $r \in \mathtt{Term}(\mathbb{P})$ constructed with variables in V and function symbols in F.

A ray r is *coloured* if it contains a colour and it is *uncoloured* otherwise.

The underlying term of a ray is defined inductively as follows:

$$\lfloor X \rfloor = X \qquad\qquad \lfloor c(r_1, ..., r_n) \rfloor = \lfloor c \rfloor (\lfloor r_1 \rfloor, ..., \lfloor r_n \rfloor)$$

with $x \in V$ and $c \in F$.

Let r be a ray. It is either:

- *objective* if it is either uncoloured or it is a coloured ray $r = c(r_1, ..., r_n)$ such c is coloured and $r_1, ..., r_n$ are uncoloured. They correspond to uncoloured terms prefixed by a colour;
- or *subjective* otherwise, if it is a coloured ray $r = f(r_1, ..., r_n)$ such that at least one ray of $\{r_1, ..., r_n\}$ is coloured.

§48.8 **Definition** (Set of colours of a ray)**.** The set of colours appearing in a ray is defined by:

- $\mathtt{colours}(t) = \emptyset$ if t is uncoloured;
- $\mathtt{colours}(c(r_1, ..., r_k)) = \{c\} \cup \left(\bigcup_{1 \leq i \leq k} \mathtt{colours}(r_i)\right)$ if $c \in F_+ \uplus F_-$ and $k \in \mathbf{N}$;

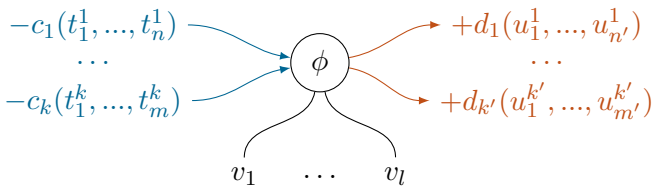


Figure 48.1: Objective star with rays seen as either input, output or unpolarised.

- $\texttt{colours}(f(r_1, ..., r_k)) = \left(\bigcup_{1 \leq i \leq k} \texttt{colours}(r_i)\right)$ otherwise.

§48.9 **Example.** Examples of objective rays are: $+c(X)$, $f(X, Y)$ and $-d(h(X))$. Examples of subjective rays are $f(X, +h(Z))$, $+c(+d, -c(Y))$ and $+c(-d(+h(X, Y)))$. We have $\texttt{colours}(+c(-d(+h(X, Y)))) = \{+c, -d, +h\}$.

§48.10 **Definition** (Star)**.** A star ϕ over a polarised signature $\mathbb{P}$ is a finite indexed family (*cf.* Appendix A.2) of *rays* $(\texttt{Term}(\mathbb{P}), I_\phi, \phi[\cdot])$. The set of variables appearing in ϕ is defined by $\texttt{vars}(\phi) := \bigcup_{i \in I_\phi} \texttt{vars}(\phi[i])$. For convenience, stars will be written as an unordered sequence $[r_1, ..., r_n]$.

The empty star is written [] and is defined by the set of indexes $I_{[]} = \emptyset$.

Let $\phi = [r_1, ..., r_n]$ be a star. It is either:

- *objective* if $r_1, ..., r_n$ are objective;
- *subjective* if $r_1, ..., r_n$ are subjective[a];
- or *animist*[b] otherwise (it has a least one objective and one subjective ray).

The shape of objective stars in illustrated in Figure 48.1.

[a] I asked Girard what was the point of internal colours and he answered me "Ça ne sert à rien du tout" which means (in French) that it was totally useless. He then told me that he just needed a trick so to make sublocations of subjective rays subjective as well. My point of view is that it adds more combinatorics to express truth and meaning. In particular, correct proofs require a clear separation between fully subjective and fully objective stars.

[b] Because it mixes object and subject.

§48.11 **Definition** (Substitution applied on a star)**.** Let θ be a substitution (*cf.* Appendix B) and ϕ a star. The application of θ on ϕ is defined by a star $\theta\phi$ with $I_{\theta\phi} := I_\phi$ and $(\theta\phi)[i] = \theta\phi[i]$.

§48.12 **Definition** (Alpha-equivalence of stars)**.** We say that two stars ϕ_1 and ϕ_2 are α-*equivalent*, written $\phi_1 \approx_\alpha \phi_2$, when there exists a renaming α (*cf.* Appendix B) such that $\phi_1 = \alpha\phi_2$.

§48.13 In this thesis, stars will be considered up to α-equivalence. We therefore define $\texttt{Star}(\mathbb{P})$ as the set of all stars over a polarised signature $\mathbb{P}$, quotiented by $\approx_\alpha$.

§48.14 **Definition** (Constellation)**.** A *constellation* Φ is a countable indexed family of stars $(\mathtt{Star}(\mathbb{P}), I_\Phi, \Phi[\cdot])$ with a possibly infinite set of indexes I_Φ. For convenience, a finite constellation will be written as a sum of stars $\Phi = \phi_1 + ... + \phi_n$.

We define the set of rays of a constellation Φ by $\mathtt{IdRays}(\Phi) = \{(i,j) \mid i \in I_\Phi, j \in I_{\Phi[i]}\}$ (we keep track of the instance of star from which rays come) and $\pm\mathtt{IdRays}(\Phi) := \{r \in \mathtt{IdRays}(\Phi) \mid r \text{ is coloured}\}$ by its restriction to coloured rays.

The empty constellation is written $\emptyset$ and is defined by $I_\emptyset = \emptyset$.

§48.15 Constellations are meant to be sort of programs. As in logic programming (*e.g.* Prolog) or functional programming (*e.g.* λ-calculus), variables will be considered bound to their star (which can be seen as sort of declarations), hence the two x in $[+f(x)] + [-f(x), y]$ are unrelated. This is similar to how the two x in the λ-term $\lambda x.(\lambda x.M)$ are different.

§48.16 Now that all the elementary objects of the stellar resolution are defined, we give a very standard encoding of natural numbers which will be useful.

§48.17 **Definition** (Encoding of natural numbers)**.** We define the function symbol $\overline{n}$ for a natural number $n \in \mathbf{N}$ by $\overline{0} = 0$ and $\overline{n+1} = s(\overline{n})$ for a unary symbol s and a constant 0.

§48.18 **Example.** We give examples of finite and infinite constellations:

- $\Phi^+_{\mathbf{N}} := [+add(\overline{0}, Y, Y)] + [-add(X,Y,Z), +add(s(X), Y, s(Z))]$ (logic program for addition);
- $\Phi^{n+m}_{\mathbf{N}} := \Phi^+_{\mathbf{N}} + [-add(\overline{n}, \overline{m}, R), R]$ (query for the computation of $n+m$);
- $\Phi_{\mathbf{N}}$ is defined by $I_{\Phi_{\mathbf{N}}} = \mathbf{N}$ and $\Phi_{\mathbf{N}}[i] := [-nat(\overline{i}), +nat(\overline{i+1})]$ (infinite chain)

over the signature defined by the variables $V = \{X, Y, Z, R\}$, the symbols $F = \{+add, -add, add, +nat, -nat, nat, s, 0\}$, $\mathtt{ar}(add) = 3, \mathtt{ar}(nat) = \mathtt{ar}(s) = 1, \mathtt{ar}(0) = 0$. The constellation $\Phi^{n+m}_{\mathbf{N}}$ corresponds to the following Horn clauses [Tär77] where $Add(X,Y,Z)$ states that $X + Y = Z$:

$$Add(0,Y,Y) \qquad \text{and} \qquad Add(X,Y,Z) \Rightarrow Add(s(X), Y, s(Z)).$$

49 Abstract execution

§49.1 The first method of computation constructs tilings and evaluate them. In order to do so, we first construct a dependency graph showing how rays can be linked. Diagrams are extracted from this dependency graph by unfolding loops by using hypergraph homomorphisms. This method is purely mathematical and not effective. It does not say how to actually construct diagrams. In particular, it is *always defined*. Even when there are infinitely many diagrams, they do not appear at all in the result. The method is not

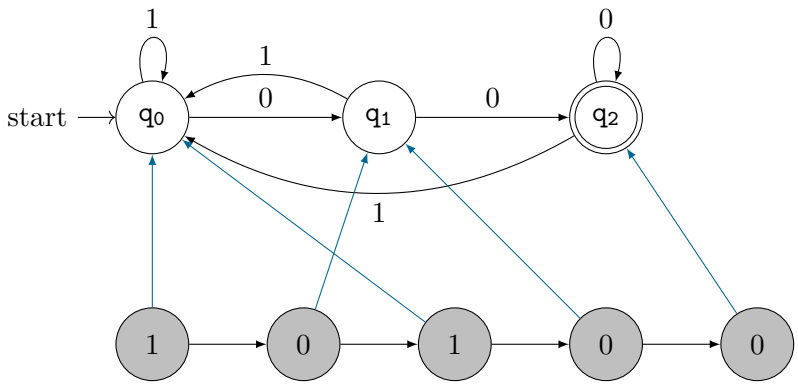


Figure 49.1: Example of finite deterministic automaton with a mapping from a word graph to its state graph.

bothered by actual infinity. It makes abstract execution something of a *denotational* nature, similar to the denotational semantics of programming languages interpreting programs as mathematical objects.

Evaluation of diagrams

§49.2 We are now interested in the formation of *diagrams* which correspond to tilings of stars (without any planarity constraints). Unlike tilings with Wang tiles or flexible tiles, it is possible to evaluate these diagrams by an edge contraction using Robinson's resolution rule (*cf.* Section 23).

§49.3 We first define the *dependency graph* of a constellation which defines the allowed connexions between stars along dual rays. A diagram corresponds to an actual plugging of stars along dual rays, following those allowed connexions. The edges linking stars will induce an equation between terms and the whole diagram will induce a unification problem. The *evaluation* of a diagram will correspond to solving its associated unification problem and producing a new star.

§49.4 In order to approach this idea more intuitively, we explain a common occurrence of it in automata theory. A finite deterministic automaton is a machine reading an input word character by character. It starts from an initial state and moves from a state to another accordingly to the current character it reads. If it ends on the final state, it *accepts* the input word. Otherwise, it rejects the input. An example of automaton of final state q_2 is given in Figure 49.1 (on the top with vertices q_0, q_1, q_2). A word can be represented as a linear graph (on the bottom with vertices $1, 0, 1, 0$ and 0) and finally, the reading of a word can be represented as a mapping from characters to states (links between the word graph and the state graph).

§49.5 The state graph of the automaton shows the allowed transitions (where loop can appear) and the word graph can be seen as tiling of states or a traversal of graph which follows those allowed transitions (sometimes by *unfolding* loops).

§49.6 Our diagrams generalise this idea. The state graph corresponds to a dependency graph and the word graph corresponds to a diagram. The difference is that a dynamics of term unification is present in our dependency graphs and diagrams can be any graph, not necessarily limited to the linear case as for automata. Mathematically speaking, a diagram will be associated with a graph homomorphism between a graph (representing the tiling) and the dependency graph, exactly like how word graphs are related to state graphs in automata theory.

§49.7 **Definition** (Duality between rays)**.** Two polarised rays r and r' are *dual* or *matchable* written $r \bowtie r'$, when r and r' are α-unifiable (with the compatibility relation $\supset$ for rays).

§49.8 **Proposition.** The relation $\bowtie$ is symmetric but anti-reflexive and anti-transitive.

Proof. Symmetry follows from the symmetry of α-unifiability (*cf.* Lemma B.1.14). We show the two other statements.

- We cannot have $r \bowtie r$ since r and r have same polarities and two function symbols of same polarity cannot be compatible by $\supset$.
- Assume that $r_1 \bowtie r_2$ and $r_2 \bowtie r_3$. It means that r_1 and r_2 are of opposite polarities and so are r_2 and r_3. Since there are only two polarities ($+$ and $-$), r_1 and r_3 must have same polarities. Hence they cannot be dual (by definition of $\supset$).

□

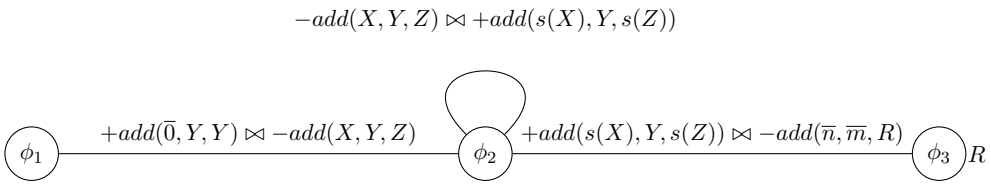
§49.9 **Example.** We have $+c(X) \bowtie -c(0)$ and $-d(X) \bowtie +d(f(X))$ but not $+c(X) \bowtie f(Y)$ (presence of unpolarised ray), $+c(X) \bowtie -d(X)$ (different head symbol), $+c(X) \bowtie +c(f(Y))$ (polarities are not opposite) nor $+c(f(X)) \bowtie -c(g(Y))$ (terms are not α-unifiable).

§49.10 **Definition** (Dependency graph)**.** The *dependency graph* of a constellation Φ *w.r.t.* a set of colours $C \subseteq F_+ \uplus F_-$ is a multigraph (*cf.* Definition C.3.1)

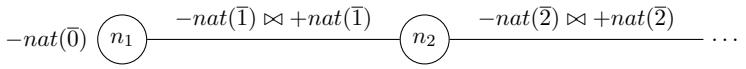
$$\mathfrak{D}[\Phi; C] := (V, E, \texttt{end})$$

where:

- $V := I_\Phi$ (the indexes of stars in Φ);
- edges between two stars of index i and i' are unordered pairs $\{(i, j), (i', j')\}$ such that for $r := \Phi[i][j]$ and $r' := \Phi[i'][j']$:

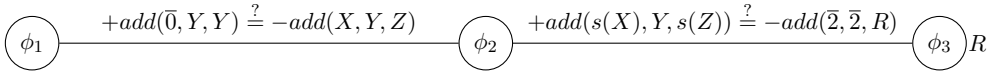


(a) Dependency graph of $\Phi_{\mathbf{N}}^{n+m}$ when $n > 0$.

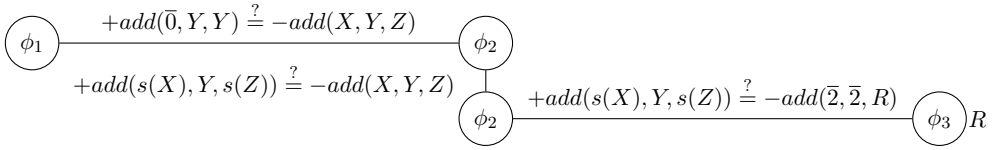


(b) Dependency graph of $\Phi_{\mathbf{N}}$.

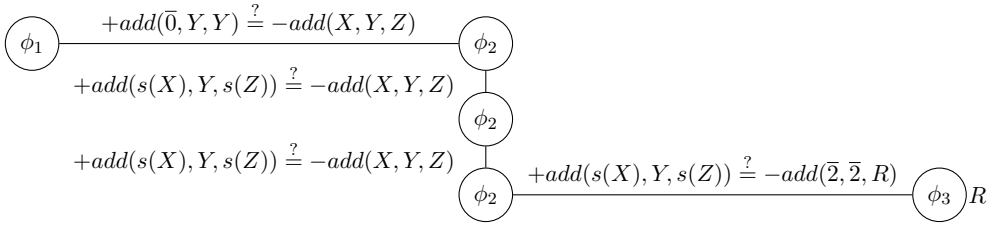
Figure 49.2: Examples of dependency graphs for constellations of Example 48.18.



(a) 0 recursive call.



(b) 1 recursive call.



(c) 2 recursive calls.

Figure 49.3: Examples of diagrams for the constellation $\Phi_{\mathbf{N}}^{2+2}$. The number of occurrences of ϕ_2 corresponds to the number of recursive calls. They correspond to unfolding of the loop of Figure 49.2 corresponding to the possibility of recursive call.

  - $r \bowtie r'$ and
  - $\texttt{colours}(r) \cup \texttt{colours}(r') \subseteq C$;
- $\texttt{end}(\{(i,j),(i',j')\}) = \{i, i'\}$.

We simply write $\mathfrak{D}[\Phi]$ when links for all colours appearing in Φ are considered, *i.e.* when $C = F_+ \uplus F_-$.

§49.11 **Example.** Two examples of dependency graphs for the two constellations $\Phi_{\mathbf{N}}^{n+m}$ and $\Phi_{\mathbf{N}}$ of Example 48.18 are presented in Figure 49.2.

§49.12 **Definition** (Adjacent of a ray)**.** Let Φ be a constellation and $\mathfrak{D}[\Phi; C] = (V, E, \texttt{end})$ its dependency graph *w.r.t.* a set of colours $C \subseteq F_+ \uplus F_-$. The set of rays *adjacent* to some given ray index (i,j) of Φ is defined by:

$$\texttt{adj}_\Phi^C(i,j) := \{(i',j') \mid \exists e \in E, e = \{(i,j),(i',j')\}\}.$$

§49.13 **Definition** (Degree of a ray)**.** Let Φ be a constellation and $\mathfrak{D}[\Phi; C] = (V, E, \texttt{end})$ its dependency graph *w.r.t.* a set of colours $C \subseteq F_+ \uplus F_-$. The *degree* of a ray (i,j) is defined by $\deg_\Phi^C(i,j) := |\texttt{adj}_\Phi^C(i,j)|$. We simply write $\deg_\Phi$ when the set of colours is omitted.

§49.14 **Definition** (Free, deterministic and branching rays)**.** Let Φ be a constellation and $C \subseteq F_+ \uplus F_-$ a set of colours. We say that a ray r of Φ is:

- *free* if $\deg_\Phi^C(r) = 0$;
- *deterministic* if $\deg_\Phi^C(r) = 1$;
- *branching* if $\deg_\Phi^C(r) > 1$;
- *co-branching* if there is some branching ray $r' \in \texttt{adj}_\Phi^C(r)$.

§49.15 **Remark.** Unpolarised rays $f \in F_0$ are always free but polarised rays can be free as well if it has no dual rays in the constellation it is in.

§49.16 **Definition** (Diagram)**.** A *C-diagram* (or simply *diagram* when $C = F_+ \uplus F_-$) over a set of colours $C \subseteq F_+ \uplus F_-$ and a constellation Φ of dependency graph $\mathfrak{D}[\Phi; C] := (V_\mathfrak{D}, E_\mathfrak{D}, \texttt{end}_\mathfrak{D})$ is a pair (G_δ, δ) where δ is hypergraph homomorphism (*cf.* Definition C.1.12) extended with other functions described below

$$\delta : G_\delta \to \mathfrak{D}[\Phi; C]$$

from a non-empty finite connected multigraph $G_\delta = (V_\delta, E_\delta, \texttt{end}_\delta)$. The homomorphism δ is made of:

- a map $\delta : V_\delta \to V_\mathfrak{D}$ between vertices;
- a map $\delta : E_\delta \to E_\mathfrak{D}$ between edges;

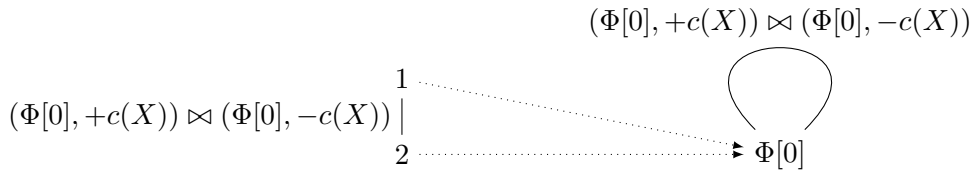


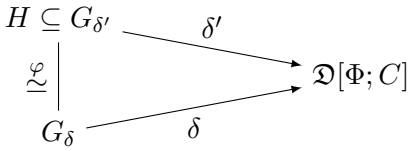
Figure 49.4: Technical remark about the definition of diagram. On the left we have a diagram which is associated with a dependency graph on the right. Since we only have the information of a link from a star to itself in the dependency graph, we cannot infer the membership of rays in the diagram.

- for all $v \in V_\delta$, an injection $\delta_v : \texttt{Neigh}(v) \to \texttt{IdRays}(\delta(v))$ associating edges incident to v in $\texttt{Neigh}(v) := \{e \in E_\delta \mid v \in \texttt{end}(e)\}$ to a ray of $\delta(v)$ such that:
  - for all $e \in E_\delta$, $(\delta(v), \delta_v(e)) \in \texttt{end}(\delta(e))$ (it is coherent *w.r.t.* δ) and
  - for all $v, v' \in V_\delta$ and $e \in E_\delta$ such that $\texttt{end}(e) = \{v, v'\}$, we have $\delta_v(e) \neq \delta_{v'}(e)$ ensuring that all selected rays are distinct (in particular, it implies that a same ray cannot be connected to other rays since it appears only once).

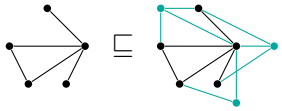
The graph G_δ is considered up to renaming of the vertices and edges and for convenience, we will often consider $V \subseteq \mathbf{N}$ in practice.

§49.17 **Example.** An example of three diagrams for the constellation $\Phi_{\mathbf{N}}^{2+2}$ (which is an instance of the constellation $\Phi_{\mathbf{N}}^{n+m}$ of Example 48.18) is given in Figure 49.3.

§49.18 **Remark.** The most natural definition of dependency graph and diagram uses undirected edges because edges represent equations linking rays. Since edges make explicit which star the rays come from, there is usually no ambiguity. However, in the case of loops, a problematic ambiguity occurs: when unfolding a loop by linking a star with one of its copies in a diagram, we do not know which rays is related to the link. Consider the constellation $\Phi := [+c(X), -c(X)]$ made of a unique star $\Phi[0] := [+c(X), -c(X)]$. The problem is illustrated in Figure 49.4. The dependency graph is made of a link from $\Phi[0]$ to itself. When constructing a diagram of size 2 (for instance), we have two occurrences of $\Phi[0]$ but the membership of rays are not distinguished. There are two possibilities: either $+c(X)$ is coming from the first instance or from the second instance. Although in this case it is not really a problem (the result will be the same), it shows that considering undirected edges does not give a faithful interpretation of the graphical and intuitive presentation of the model, which may be problematic in some cases. Hence, two directions (for each rays of the associated equation) must be distinguished. If we really want a problematic case, we can consider the constellation $[+a(f(X)), -a(X), +b(X)] + [-b(X)]$. There is a loop between $-a(X)$ and $+a(f(X))$ in the first star. But if it linked to a copy of itself then to $-b(X)$, depending on if it is linked through $-a(X)$ or $-a(f(X))$ we may (or not)



(a) Saturation expressed with graph homomorphisms.



(b) We have $\delta \sqsubseteq \delta'$ when δ' is an extension of δ with further links and possibly repetitions of stars. Both follow the connexions of a dependency graph $\mathfrak{D}[\Phi; C]$.

Figure 49.5: Order $\sqsubseteq$ on diagrams representing an idea of saturation.

> transmit $f(X)$ to $-b(X)$ by unification. This explains the need for injections δ_v for all vertices v of a diagram. It explicitly shows what rays we are referring to.

§49.19 In order to make definitions more accurate, we must state when two diagrams are equivalent. Otherwise, there are confusions because of the fact that the names of vertices in diagrams can be freely changed. Intuitively, diagram equivalence should be related to tiling equivalence. Two diagrams should be equivalent when they refer to an equivalent tiling of stars in a given constellation.

§49.20 **Definition** (Diagram equivalence)**.** Let (G_δ, δ) and $(G_{\delta'}, \delta')$ be two diagrams. They are *equivalent*, written $\delta \simeq \delta'$, when $G_\delta \simeq G_{\delta'}$ with a graph isomorphism φ such that $\delta = \delta' \circ \varphi$.

§49.21 **Notation** (Free rays and closed diagrams)**.** Given a C-diagram (G_δ, δ) of a constellation Φ with $G_\delta = (V_\delta, E_\delta, \mathtt{end}_\delta)$, we define its multiset (*cf.* Appendix A.2) of *free* (unconnected) rays $\mathtt{free}(\delta) \subseteq \mathtt{IdRays}(\Phi)$ by:

$$\mathtt{free}(\delta) := \mathtt{multiset}\{r \in \mathtt{IdRays}(\Phi) \mid \deg^C_\Phi(r) = 0\}.$$

If $\mathtt{free}(\delta) = \emptyset$, we say that δ is *closed*.

§49.22 We usually would like diagrams to be impossible to extend by connecting more stars, which corresponds to a notion of *saturation*. In terms of tiling it is understood as the construction of the largest constructible tiling with occurrences of tiles from a given tile set and in terms of programming, it corresponds to a complete computation to be done.

§49.23 **Definition** (Saturated diagram)**.** We define a binary relation $\sqsubseteq$ (illustrated in Figure 49.5) on C-diagrams over a constellation Φ by: $\delta \sqsubseteq \delta'$ if there exists an isomorphism φ from a graph $H \subseteq G_{\delta'}$ to G_δ such that $\delta = \delta' \circ \varphi$. A maximal C-diagram *w.r.t.* $\sqsubseteq$ is called *saturated*.

§49.24 **Proposition.** The relation $\sqsubseteq$ is a preorder.

Proof. We have $\delta \sqsubseteq \delta$ by taking the subgraph $G_\delta \subseteq G_\delta$. The isomorphism φ is the identity function so we trivially have $\delta = \delta \circ \varphi$.

Assume we have $\delta_1 \sqsubseteq \delta_2$ and $\delta_2 \sqsubseteq \delta_3$. Hence, we have isomorphisms $\varphi_{1,2} : (H_2 \subseteq G_{\delta_2}) \simeq G_{\delta_1}$ and $\varphi_{2,3} : (H_3 \subseteq G_{\delta_3}) \simeq G_{\delta_2}$ such that $\delta_1 = \delta_2 \circ \varphi_{1,2}$ and $\delta_2 = \delta_3 \circ \varphi_{2,3}$. We can construct an isomorphism $\varphi_{1,2} \circ \varphi_{2,3}$ such that $\delta_1 = \delta_3 \circ \varphi$ and G_{δ_3} is indeed an extension of G_{δ_1} following the connexions of the same dependency graph. □

§49.25 Links in a diagram have an underlying equation. It follows that a whole diagram is associated with a unification problem. A minor but important technical problem is that variables appearing in a constellation Φ are meant to be bound to their star. Hence, before evaluating, we must rename variables so to mark their membership to a star of Φ. Fortunately, it is possible to define a canonical renaming by using the star indexes I_Ω.

§49.26 **Convention.** Assume that the signature $\mathbb{P}$ contains a fresh symbol x_v for each $v \in V$ such that all x_v are pairwise distinct.

§49.27 **Definition** (Underlying equation and problem)**.** Let (G_δ, δ) be a C-diagram of a constellation Φ with $G_\delta = (V_\delta, E_\delta, \mathtt{end}_\delta)$. By Convention 49.26, all vertices $x \in V$ induce a family of renamings $\alpha_v(x) = x_v$ for any variable x.

The *underlying equation* of an edge $e \in E_\delta$ such that $\mathtt{end}_\delta(e) = \{v, v'\}$ and $\delta(e) = \{(i, j), (i', j')\}$ is defined by

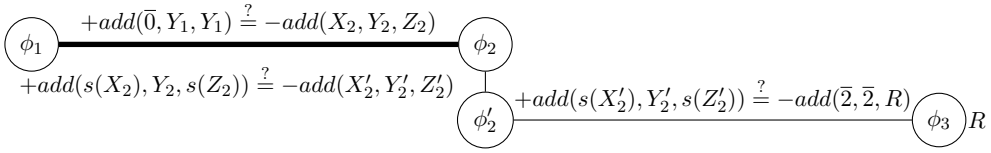
$$\mathtt{eq}(e) := \alpha_v \lfloor \Phi[i][j] \rfloor \stackrel{?}{=} \alpha_{v'} \lfloor \Phi[i'][j'] \rfloor$$

and the *underlying problem* of δ is defined by

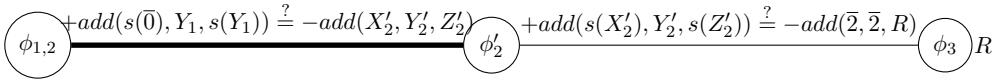
$$\mathtt{Prob}(\delta) = \{\mathtt{eq}(e) \mid e \in E_\delta\}.$$

Underlying problems are considered up to α-equivalence. In particular, the choice of renaming function is not relevant but variables should be distinguished when they come from two distinct stars.

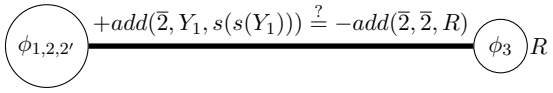
§49.28 In Girard's original paper [Gir17, Section 2.3], the evaluation of diagrams is defined as an edge contraction. An edge e between two stars ϕ and ϕ' contains an equation which is resolved and then the associated solution is propagated to both ϕ and ϕ'. The two connected rays associated with e are finally destructed in the process. It reminds of chemical interactions but also of how information is propagated and organised in a network. This process can fail in presence of errors during the execution of a unification algorithm. This corresponds exactly to Robinson's resolution (*cf.* Section 23) and it generalises the composition of flows (*cf.* Section 37).



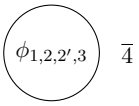
(a) Correct diagram computing $2 + 2$ (1 recursion) with the ray R visible.



(b) Fusion of ϕ_1 and ϕ_2 with $\theta := \{X_2 \mapsto \overline{0}, Y_2 \mapsto Y_1, Z_2 \mapsto Y_1\}$.



(c) Fusion of $\phi_{1,2}$ and ϕ_2' with $\theta := \{X_2' \mapsto s(\overline{0}), Y_2' \mapsto Y_1, Z_2' \mapsto s(Y_1)\}$



(d) Fusion of the two remaining stars with $\theta := \{R \mapsto \overline{4}\}$.

Figure 49.6: Fusion of the diagram from Figure 49.3b.

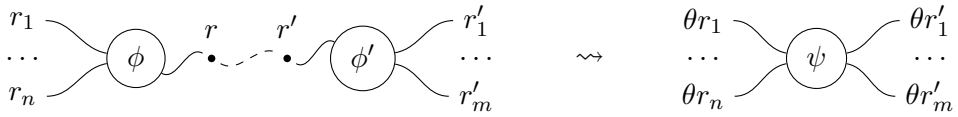


Figure 49.7: Illustration of a step of fusion where θ is the principal unifier of the underlying unification problem of the pair of rays (r, r'). The fusion of the two stars ϕ and ϕ' along the rays r and r' produces a new star ψ.

§49.29 We define this step-by-step procedure of diagram contraction based on an operation of *fusion*, but also an alternative of evaluation called *actualisation* which can simulate steps of fusion by evaluating a diagram by solving the whole unification problem associated. It is similar to how small step evaluation differs from big step evaluation in the theory of programming languages [Ama16, Section 1.1].

§49.30 **Definition** (Fusion)**.** Let $\phi_1 := \phi_1' \uplus \{r_1\}$ and $\phi_2 := \phi_2' \uplus \{r_2\}$ be two stars. We define their *fusion* $\phi_1 \overset{j,j'}{\triangledown} \phi_2$ by $\theta\phi_1' \uplus \theta\phi_2'$ where $\phi_1[j] \bowtie \phi_2[j']$ and $\theta := \texttt{solution}\{\phi_1[j] \stackrel{?}{=} \phi_2[j']\}$.

We write $\phi_1 \overset{j,j'}{\triangledown}_\alpha \phi_2$ for the renaming of variables of both ϕ_1 and ϕ_2 to make them distinct followed by their fusion.

We simply write $\phi_1 \triangledown \phi_2$ (and $\phi_1 \triangledown_\alpha \phi_2$) without indexes above when there is a unique

possible choice of index and we choose to leave them implicit. Fusion is illustrated in Figure 49.7.

§49.31 **Example.** Assume we have two stars $\phi^+_{\mathbf{N}}R := [-add(X,Y,Z), +add(s(X),Y,s(Z))]$ and $\phi^{2+2}_Q := [-add(\overline{2},\overline{2},R),R]$ indexed by natural numbers corresponding to their position. We have:

$$\phi^+_{\mathbf{N}}R \overset{1,0}{\triangledown} \phi^{2+2}_Q := [-add(\overline{1},\overline{2},Z), s(Z)].$$

$$\phi^+_{\mathbf{N}}R \overset{1,0}{\triangledown}_\alpha (\phi^+_{\mathbf{N}}R \overset{1,0}{\triangledown}_\alpha \phi^{2+2}_Q) := [-add(0,\overline{2},Z'), s(s(Z'))].$$

§49.32 **Lemma** (Associativity of fusion)**.** Let ϕ_1, ϕ_2 and ϕ_3 be stars. We have

$$\phi_1 \overset{i,j}{\triangledown} (\phi_2 \overset{i',j'}{\triangledown} \phi_3) \approx_\alpha (\phi_1 \overset{i,j}{\triangledown} \phi_2) \overset{i',j'}{\triangledown} \phi_3,$$

if we assume that all these fusions succeed.

Proof. It does not matter if some stars or all are the same. In case some stars are the same, we will have to reduce loops but in the end, what matters are only the equations involved and loops only add equations with identical variables, *i.e.* we have $t \overset{?}{=} u$ such that variables of t may appear in u because we are in the same star. We can consider without loss of generality that all stars are distinct. Assume we have three stars:

$$\phi_1 := [u_1, ..., u_n, r_1], \qquad \phi_2 := [v_1, ..., v_m, r_2, r'_2], \qquad \phi_3 := [s_1, ..., s_k, r_3]$$

with $\phi_1[i] = r_1, \phi_2[j] = r_2, \phi_2[i'] = r'_2$ and $\phi_3[j'] = r_3$.

Assume that rays are uniquely connected and that we have:

- $\theta := \texttt{solution}\{r_1 \overset{?}{=} r_2\}$, $\theta' := \texttt{solution}\{r'_2 \overset{?}{=} r_3\}$;
- $\psi := \texttt{solution}\{\theta' r_1 \overset{?}{=} \theta' r_2\}$, $\psi' := \texttt{solution}\{\theta r'_2 \overset{?}{=} \theta r_3\}$.

If we try to apply the steps of fusion, we have:

- $\phi_1 \overset{i,j}{\triangledown} (\phi_2 \overset{i',j'}{\triangledown} \phi_3) = \phi_1 \overset{i,j}{\triangledown} [\theta' v_1, ..., \theta' v_m, \theta' s_1, ..., \theta' s_k, \theta' r_1, \theta' r_2]$
  $= [\psi\theta' u_1, ..., \psi\theta' u_n, \psi\theta' v_1, ..., \psi\theta' v_m, \psi\theta' s_1, ..., \psi\theta' s_k]$;
- $(\phi_1 \overset{i,j}{\triangledown} \phi_2) \overset{i',j'}{\triangledown} \phi_3 = [\theta u_1, ..., \theta u_n, \theta v_1, ..., \theta v_m, \theta r'_2, \theta r_3] \overset{i',j'}{\triangledown} \phi_3$
  $= [\psi'\theta u_1, ..., \psi'\theta u_n, \psi'\theta v_1, ..., \psi'\theta v_m, \psi'\theta s_1, ..., \psi'\theta s_k]$.

We can think about how the subsitutions are constructed. Both substitutions can be constructed from the same set of equations $\{r_1 \overset{?}{=} r_2, r'_2 \overset{?}{=} r_3\}$. The substitution $\psi\theta'$ is constructed by solving the equations of $r'_2 \overset{?}{=} r_3$ first then the equations of $r_1 \overset{?}{=} r_2$. The substitution $\psi'\theta$ is obtained by reversing the order: first the equations of $r_1 \overset{?}{=} r_2$ are

solved then the ones of $r'_2 \stackrel{?}{=} r_3$. By Theorem B.2.6, we can solve the equations in any order and obtain the same solution up to renaming. Hence $\psi\theta'$ and $\psi'\theta$ have the same action on terms up to renaming.

Remark that there is no need to consider failure because one failure is expected to make the whole diagram collapse. □

§49.33 **Definition** (Diagram contraction)**.** The operation of *diagram contraction* over a C-diagram (G_δ, δ) for $C \subseteq F_- \uplus F_+$ with:

- $G_\delta := (V_\delta, E_\delta, \mathtt{end}_\delta)$;
- $\mathfrak{D}[\Phi; C] = (V_\mathfrak{D}, E_\mathfrak{D}, \mathtt{end}_\mathfrak{D})$

is defined as a graph rewriting of G_δ along some edge $e \in E_\delta$ such that $\mathtt{end}(e) = \{v, v'\}$ (possibly with $v = v'$) and $\delta(e) = \{(i,j),(i',j')\}$ (possibly with $i = i'$), which updates the homomorphism $\delta : G_\delta \to \mathfrak{D}[\Phi; C]$. It produces a new diagram $\overset{e}{\triangledown}(G_\delta, \delta) = (G'_\delta, \delta')$ with $G'_\delta := (V_{\delta'}, E_{\delta'}, \mathtt{end}_{\delta'})$ and $\delta' : G_{\delta'} \to \mathfrak{D}[\Phi'; C]$, defined as follows:

- Assume $v \neq v'$. We want to do an edge contraction between two different vertices. We remove v, v' and their incident edges and replace them by a new vertex w inheriting their previous bonds, *i.e.* $V_{\delta'} := V_\delta - \{v, v'\} \cup \{w\}$ and for $e_1, ..., e_n \in E_\delta$ with $\mathtt{end}_\delta(e_i) = \{v_i, v\}$ and for $e'_1, ..., e'_m \in E_\delta$ with $\mathtt{end}_\delta(e'_i) = \{v'_i, v'\}$ we have $E_{\delta'} := (E_\delta - \{e, e_1, ..., e_n, e'_1, ..., e'_m\}) \cup \bigcup_{i=1}^n \{g_i\} \cup \bigcup_{i=1}^m \{g'_i\}$ such that $\mathtt{end}_{\delta'}(g_i) = (v_i, w)$ and $\mathtt{end}_{\delta'}(g'_i) = (v'_i, w)$. In the other cases, we have $\mathtt{end}_{\delta'}(x) = \mathtt{end}_\delta(x)$;

  We adapt the homomorphism to the previous change.

  - Φ' is defined by $I_{\Phi'} := I_\Phi - \{i, i'\}$, and $\Phi'[i_w] := \alpha_v \Phi[i] \overset{j,j'}{\triangledown} \alpha_{v'} \Phi[i']$ for some fresh $i_w \notin I_\Phi$ and $\Phi'[k] := \Phi[k]$ otherwise, where α_u comes from Definition 49.27;
  - $\delta'(w) = i_w$ and $\delta'(x) = \delta(x)$ otherwise;
  - if $e_i = \{(\delta(v_i), j_i), (-, j'_i)\}$ then $\delta'(g_i) = \{(\delta(v_i), j_i), (i_w, j'_i)\}$ and if $e'_i = \{(\delta(v'_i), j_i), (-, j'_i)\}$ then $\delta'(g'_i) = \{(\delta(v'_i), j_i), (i_w, j'_i)\}$.
- Assume $v = v'$. We want to do an edge contraction linking one same vertex. We have $V_{\delta'} := V_\delta$ and $E_{\delta'} := E_\delta - \{e\}$. We have $\delta'(x) = \delta(x)$ (but $\delta'(e)$ is not defined anymore). Almost nothing changes except that we remove the edge e and the only star $\delta(v) = \delta(v')$ is updated. The constellation Φ is defined by $I_{\Phi'} := I_\Phi$ such that $\Phi[\delta(v)]$ is a star ϕ such that $I_\phi := \theta(I_{\delta(v)} - \{j, j'\})$ with $\phi[j''] = \theta\Phi[\delta(v)][j'']$ for all $j'' \in I_\phi$ for θ the solution of $\mathtt{eq}(e)$.

We write $(G_\delta, \delta) \rightsquigarrow_e (G_{\delta'}, \delta')$ when $(G_{\delta'}, \delta') = \overset{e}{\triangledown}(G_\delta, \delta)$ along the edge e, and write $(G_\delta, \delta) \rightsquigarrow^n (G_{\delta'}, \delta')$ when there is a series $n \geq 0$ of steps such that $(G_\delta, \delta) \rightsquigarrow_{e_1} ... \rightsquigarrow_{e_n} (G_{\delta'}, \delta')$ for some edges $e_1, ..., e_n$. We leave the edge e implicit when obvious or not important and we can write $\rightsquigarrow^*$ instead of $\rightsquigarrow^n$ when the number of steps is implicit.

§49.34 **Definition** (Correct diagrams and their actualisation)**.** A C-diagram (G_δ, δ) of a constellation Φ is *correct* if $\texttt{Prob}(\delta)$ has a solution.

The *actualisation* of a correct diagram (G_δ, δ) is the star $\Downarrow \delta$ defined by $I_{\Downarrow \delta} := \texttt{free}(\delta)$ such that $(\Downarrow \delta)[(i,j)] = (\psi \circ \theta)(\Phi[i][j])$, where $\psi := \texttt{solution}(\texttt{Prob}(\delta))$ and $\theta := \alpha_{v_1} \circ ... \circ \alpha_{v_n}$ with $V^{G_\delta} = \{v_1, ..., v_n\}$ is the composition of renamings of Definition 49.27.

§49.35 **Lemma** (Termination of diagram contraction)**.** If (G_δ, δ) is correct then there exists $(G_{\delta'}, \delta')$ such that $(G_\delta, \delta) \rightsquigarrow^* (G_{\delta'}, \delta')$ and there is no $(G_{\delta''}, \delta'')$ such that $(G_{\delta'}, \delta') \rightsquigarrow^* (G_{\delta''}, \delta'')$.

Proof. By Theorem B.2.3 and the fact that fusion always succeed for correct diagrams, since (G_δ, δ) is correct, the underlying unification problem only has solvable equations and hence there exists a full execution or the algorithm. □

§49.36 **Corollary** (Confluence of diagram contraction)**.** For some diagrams (G_δ, δ), (G_{δ_1}, δ_1) and (G_{δ_2}, δ_2), if $(G_\delta, \delta) \rightsquigarrow^* (G_{\delta_1}, \delta_1)$ and $(G_\delta, \delta) \rightsquigarrow^* (G_{\delta_2}, \delta_2 1)$ then there exists $(G_{\delta'}, \delta')$ such that $(G_{\delta_1}, \delta_1) \rightsquigarrow^* (G_{\delta'}, \delta')$ and $(G_{\delta_2}, \delta_2) \rightsquigarrow^* (G_{\delta'}, \delta')$.

Proof. By the associativity of fusion (*cf.* Lemma 49.32), and the fact that diagram contraction is a step of fusion inside the graph of a diagram, we can infer that we have commutation of diagram contraction, *i.e.* $(G_{\delta_1}, \delta_1) \rightsquigarrow_e D \rightsquigarrow_{e'} (G_{\delta_2}, \delta_2) = (G_{\delta_1}, \delta_1) \rightsquigarrow_{e'} D \rightsquigarrow_e (G_{\delta_2}, \delta_2)$ for some diagram D. From this result, we have local confluence of diagram contraction. Together with the termination of diagram contraction (*cf.* Lemma 49.35), it is possible to use the so-called Newman's lemma [BN98, Lemma 2.7.2] to infer the confluence of diagram contraction (actually the diamond property which is equivalent). □

§49.37 **Lemma.** If $(G_\delta, \delta) \rightsquigarrow_e (G_{\delta'}, \delta')$ then $\Downarrow(G_\delta, \delta) = \Downarrow(G_{\delta'}, \delta')$.

Proof. Assume $(G_\delta, \delta) \rightsquigarrow_e (G_{\delta'}, \delta')$ with $e = \{(v,j), (v',j')\}$ and (G_δ, δ) is defined over a constellation Φ. There are two cases.

- Assume $v \neq v'$. We have done an edge contraction on G_δ where two vertices merged into one vertex w such that $\Phi[\delta(w)] := \alpha_v \Phi[\delta(v)] \overset{j,j'}{\triangledown} \alpha_{v'} \Phi[\delta(v')]$. It terms of equations, what we have done is computing $\theta := \texttt{solution}\{\alpha_v \Phi[\delta(v)][j] \overset{?}{=} \alpha_{v'} \Phi[\delta(v')][j']\}$ and applying it on the merge of the two interacting stars $\delta(v)$ and $\delta(v')$. The only difference between (G_δ, δ) and $(G_{\delta'}, \delta')$ is this one equation $\texttt{eq}(e)$

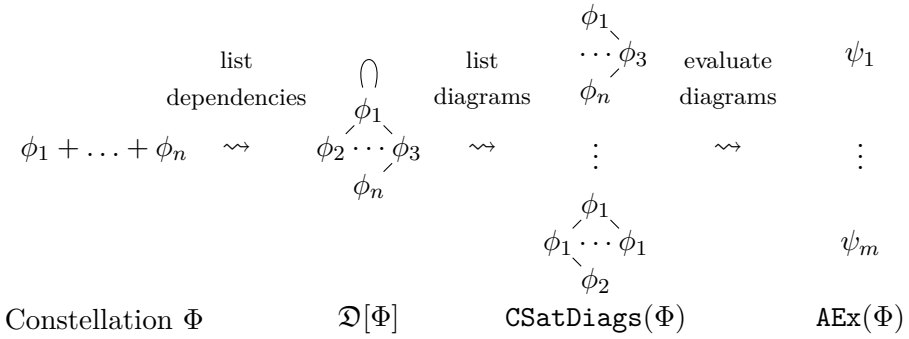


Figure 49.8: Illustration of the execution of a finite and strongly normalising constellation.

which has been solved. If we translate this situation in terms of equation solving, it corresponds to the transition between $P \cup \{\mathtt{eq}(e)\}$ and $P \cup P'$ such that $\overrightarrow{P'} = \theta$ where P contains the other equations of (G_δ, δ). In particular, by confluence of the unification algorithm (*cf.* Theorem B.2.4), equations can be solved in any order and we would end up on the same result in any case. It follows that actualisation can first solve $\mathtt{eq}(e)$ and the remaining equations are the ones of $(G_{\delta'}, \delta')$.

- Assume $v = v'$. The reasoning is the same and an equation is still solved except that we have a connexion between two rays of a same star instead of a fusion between two occurrences of stars.

□

§49.38 **Theorem** (Relation between diagram contraction and actualisation)**.** For all correct diagram (G_δ, δ) such that $G_\delta := (V_\delta, E_\delta, \mathtt{end}_\delta)$, we have $(G_\delta, \delta) \rightsquigarrow^{|E_\delta|} \Downarrow(G_\delta, \delta)$.

Proof. By induction on $n = |E_\delta|$. If $n = 0$ then we have 0 links and (G_δ, δ) reduces to itself in 0 step, which corresponds to $\Downarrow(G_\delta, \delta)$. For the inductive case, we have $n = n' + 1$ and we assume that $(G_{\delta'}, \delta') \rightsquigarrow^{n'} \Downarrow(G_{\delta'}, \delta')$ and show $(G_\delta, \delta) \rightsquigarrow^{n'+1} \Downarrow(G_\delta, \delta)$ or equivalently $(G_\delta, \delta) \rightsquigarrow (G_{\delta'}, \delta') \rightsquigarrow^{n'} \Downarrow(G_\delta, \delta)$ for some $(G_{\delta'}, \delta')$. By Lemma 49.37, if $(G_\delta, \delta) \rightsquigarrow (G_{\delta'}, \delta')$, we must have $\Downarrow(G_\delta, \delta) = \Downarrow(G_{\delta'}, \delta')$. It means that solving the equations for δ and δ' ultimately leads to the same result. Hence, by substitution on the induction hypothesis, we have $(G_{\delta'}, \delta') \rightsquigarrow^{n'} \Downarrow(G_\delta, \delta)$. Therefore, $(G_\delta, \delta) \rightsquigarrow^{n'+1} \Downarrow(G_\delta, \delta)$. □

§49.39 **Remark.** The actualisation of diagrams corresponds to a single star because diagrams are connected.

Execution of constellations

§49.40 The *(abstract) execution* of a constellation Φ consists in computing all the correct saturated diagrams of Φ and actualising them. This process is illustrated in Figure 49.8.

§49.41 **Notation** (Set of correct saturated diagrams)**.** We write $\texttt{SatDiags}_C(\Phi)$ for the *set of all saturated diagrams* obtained from $\mathfrak{D}[\Phi; C]$ for a constellation Φ and a set of colours $C \subseteq F$. We omit the set of colours and simply write $\texttt{SatDiags}(\Phi)$ when C is all colours of F.

We write $\texttt{CSatDiags}_C(\Phi)$ for the set of all diagrams in $\texttt{SatDiags}_C(\Phi)$ which are correct.

§49.42 **Definition** (Execution and normal form)**.** The *execution* or *normal form* of a constellation Φ *w.r.t.* a set of colours $C \subseteq F$ is defined by $\texttt{AEx}_C(\Phi) := \Downarrow \texttt{CSatDiags}_C(\Phi)$, where

$$\Downarrow \texttt{CSatDiags}_C(\Phi) := \{\Downarrow \delta \mid \delta \in \texttt{CSatDiags}_C(\Phi)\}.$$

We write $\texttt{AEx}(\Phi)$ when all colours in Φ participate in the execution.

§49.43 We define an operation of *concealing* which violently mutes the constellation by removing stars containing polarised rays, thus forbidding any communication with other stars.

§49.44 **Definition** (Concealing)**.** Let Φ be a constellation. The *concealing* of Φ is the constellation $\lightning \Phi$ defined by $I_{\lightning \Phi} := \{i \in I_\Phi \mid \phi := \Phi[i], \forall j \in I_\phi, \phi[j] \text{ is uncoloured}\}$.

§49.45 We define an operation of *noise filtering* of a constellation which removes the empty stars which are irrelevant since they cannot be connected. However, they still are valuable for quantitative analyses as we will see in the interpretation of MLL.

§49.46 **Definition** (Noise filtering)**.** Let Φ be a constellation. The *noise filtering* of Φ is the constellation $\flat\Phi := \{i \in I_\Phi \mid \Phi[i] \neq []\}$.

§49.47 The plurality of diagrams is related to the several parallel choices for branching rays (but branching rays do not necessarily yield several diagrams). The execution is defined as if we were considering all possible choices at once. It is not a problem because in logic programming for instance, inferences coming from a given query can lead to several answers and we are usually interested in all possible answers.

§49.48 It is possible to recover the notion of choice by simply picking a star of the normal form of a constellation. This is not very natural since we have to compute all possibilities before making the choice. It is however sufficient for abstract execution which is purely mathematical.

§49.49 **Definition** (Non-deterministic choice)**.** Let Φ be a constellation. A *non-deterministic choice* over Φ is defined by a constellation $\texttt{pick}_i(\Phi) := \Phi[i]$ where $i \in I_{\texttt{AEx}(\Phi)}$. We simply write $\texttt{pick}(\Phi)$ when we do not want to specify the index of the selected star.

The dynamics of subjective rays

§49.50 Something which has barely been mentioned is the case of internal polarities, the main feature making our model different from original resolution. There are two important characteristics of the introduction of internal polarities:

1. they introduce new polarised rays during execution. For instance, in

$$[-f(+g(X))] + [X, +f(X)],$$

   if we connect the two stars along $-f(+g(X))$ and $+f(X)$, we obtain $[+g(X)]$. We transformed an unpolarised ray X into a new polarised ray $+g(X)$;

2. they introduce mechanisms of *synchronisation*. In the previous example, if we want to interact/communicate with $+g(X)$ we must first making $+f$ and $-f$ interact. This is reminiscent of the use of *semaphores* in programming.

§49.51 Now, how does our abstract execution react in presence of internal polarities? The first point previously mentioned is problematic because diagrams are *fixed* structures. Once you construct a diagram connecting $+f$ with $-f$, you cannot see that there is an interaction available with $+g(X)$ providing we unlock it by first interacting with the colour $\pm f$. We could change definitions and make execution conscious of that but that would be complicated and probably unnecessary. What internal polarities do is that even after fully executing a constellation, there may be new connexions left. For instance in:

$$[-f(+g(X))] + [X, +f(X)] + [-g(X), +f(X), a]$$

we can make two diagrams over the colour $\pm f$ and the whole constellation has the normal form $[+g(X)] + [-g(+g(X)), a]$. This constellation can be executed again but its stars should be extended with stars of the initial constellation which has been lost during the first execution. We see that $[+g(X)]$ can be connected with $[-g(+g(X)), a]$, leading to $[a]$ but also with $[-g(X), +f(X), a]$ and in that case we obtain $[+f(X), a]$ which can be extended with $[-f(+g(X))]$, leading to $[a]$. We obtain $[a] + [a]$ which is the "real normal form". We can then handle internal polarities with repeated execution corresponding to several sequential layers in computation (as in imperative programming or any sequential model of computation).

§49.52 **Definition** (Iterated and hyper execution)**.** Let Φ and Ψ be constellations and $C \subseteq$

$F_+ \uplus F_-$ a set of colours. We define its *iterated* execution by the following constellation:

$$\mathtt{AEx}^0_{\Phi,C}(\Psi) = \Psi \qquad \mathtt{AEx}^1_{\Phi,C}(\Psi) = \mathtt{AEx}_C(\Psi)$$

$$\mathtt{AEx}^{n+1}_{\Phi,C}(\Psi) = \mathtt{AEx}_C(\Psi' \uplus \mathtt{AEx}^n_{\Phi,C}(\Psi)) \qquad \text{for } n > 1$$

where Ψ' is the constellation made of stars of Φ which are matchable with the stars of $\mathtt{AEx}^n_{\Phi,C}(\Psi)$.

The *hyper execution* of Φ is a constellation $\mathtt{AEx}^\infty_C(\Phi) := \mathtt{AEx}^k_{\Phi,C}(\Phi)$ which is defined when there exists some $k \in \mathbf{N}$ such that $\mathtt{AEx}^k_{\Phi,C}(\Phi) = \mathtt{AEx}^{k+1}_{\Phi,C}(\Phi)$.

§49.53 **Example.** For the constellation

$$\Phi := [-f(+g(X))] + [X, +f(X)] + [-g(X), +f(X), a],$$

we have $\mathtt{AEx}^1_{\Phi,C}(\Phi) = [+g(X)] + [-g(+g(X)), a]$ and $\mathtt{AEx}^\infty_C(\Phi) = \mathtt{AEx}^2_{\Phi,C}(\Phi) = [a] + [a]$.

§49.54 **Proposition** (Objective full evaluation)**.** Let (G_δ, δ) be a C-diagram for a constellation Φ such that all stars are objective. There is no pair of matchable rays in $\Downarrow(G_\delta, \delta)$.

Proof. First, $\mathtt{free}(\delta)$ contains no pair of matchable rays, by definition. The only way to produce pairs of matchable rays is to have an equation $\{X \mapsto s\}$ in $\mathtt{Prob}(\delta)$ replacing a variable X by some subjective ray s. However, since all stars are objective and equations do not consider head polarities by definition of $\lfloor\cdot\rfloor$, it follows that there is no such equation. Therefore, there cannot be pair of matchable rays in $\Downarrow(G_\delta, \delta)$. □

§49.55 **Corollary** (Idempotence of abstract execution)**.** For any constellation Φ and set of colours $C \subseteq F_+ \uplus F_-$, if Φ has only objective stars then $\mathtt{AEx}_C(\mathtt{AEx}_C(\Phi)) = \mathtt{AEx}_C(\Phi)$.

Proof. By Proposition 49.54, since abstraction execution is made of the actualisation of all saturated and correct diagrams and that those diagrams do not have pairs of matchable rays, each star of $\mathtt{AEx}_C(\Phi)$ has no pair of matchable rays. It remains to show that there are no pair of matchable rays between the distinct stars. If we had $\Phi[i][j] \bowtie \Phi[i'][j']$ with $i \neq i'$ then those two rays could have been connected and form a same diagram and are thus not free in (G_δ, δ). It follows that no diagram can be constructed from $\mathtt{AEx}_C(\Phi)$ if all stars are objective. Therefore, $\mathtt{AEx}_C(\mathtt{AEx}_C(\Phi)) = \mathtt{AEx}_C(\Phi)$. □

§49.56 **Proposition** (Idempotence barrier)**.** For any constellation Φ and set of colours $C \subseteq F_+ \uplus F_-$, if $\mathtt{AEx}^\infty_C(\Phi) = \mathtt{AEx}^k_C(\Phi)$ for some $k \in \mathbf{N}$ then for all $l \in \mathbf{N}$, $\mathtt{AEx}^{k+1+l}_C(\Phi) = \mathtt{AEx}^{k+l}_C(\Phi)$.

Proof. Assume we have $\mathtt{AEx}^\infty_C(\Phi) = \mathtt{AEx}^k_C(\Phi)$. We know by the definition of $\mathtt{AEx}^\infty_C(\Phi)$ that $\mathtt{AEx}^{k+1}_C(\Phi) \overset{H}{=} \mathtt{AEx}^k_C(\Phi)$. By induction on l.

◇ **Base case** If $l = 0$ then we have $\mathtt{AEx}_C^{k+1+0}(\Phi) = \mathtt{AEx}_C^{k+1}(\Phi) \overset{H}{=} \mathtt{AEx}_C^{k}(\Phi) = \mathtt{AEx}_C^{k+0}(\Phi)$.

◇ **Inductive case** Now assume $l = l' + 1$ and $\mathtt{AEx}_C^{k+1+l'}(\Phi) = \mathtt{AEx}_C^{k+l'}(\Phi)$ (induction hypothesis). By definition and by using the induction hypothesis, we have

$$\mathtt{AEx}_C^{k+1+l}(\Phi) = \mathtt{AEx}_C^{k+1+l'+1}(\Phi) = \mathtt{AEx}_C(\mathtt{AEx}_C^{k+1+l'}(\Phi)) \overset{IH}{=} \mathtt{AEx}_C(\mathtt{AEx}_C^{k+l'}(\Phi))$$

$$= \mathtt{AEx}_C^{k+l'+1}(\Phi) = \mathtt{AEx}_C^{k+l}(\Phi).$$

□

§49.57 Although execution is always idempotent for objective constellations, we have seen in the previous example that it can be lost in presence of subjective rays. If there is a point in repeated execution where we reach idempotence then hyperexecution is idempotent (it is similar to the nilpotency property in GoI).

§49.58 **Corollary** (Idempotence of hyper execution)**.** For any constellation Φ and set of colours $C \subseteq F_+ \uplus F_-$, if $\mathtt{AEx}_C^{\infty}(\Phi) = \mathtt{AEx}_C^{k}(\Phi)$ for some $k \in \mathbf{N}$ then $\mathtt{AEx}_C^{\infty}(\mathtt{AEx}_C^{\infty}(\Phi)) = \mathtt{AEx}_C^{\infty}(\Phi)$.

Proof. By hypothesis, we have $\mathtt{AEx}_C^{\infty}(\Phi) = \mathtt{AEx}_C^{k}(\Phi)$ for some $k \in \mathbf{N}$. Hence we have to prove $\mathtt{AEx}_C^{k}(\mathtt{AEx}_C^{k}(\Phi)) = \mathtt{AEx}_C^{k}(\Phi)$. By induction on k.

◇ **Base case** If $k = 0$ then $\mathtt{AEx}_C^{0}(\mathtt{AEx}_C^{0}(\Phi)) = \mathtt{AEx}_C^{0}(\Phi) = \Phi$.

◇ **Induction case** Assume $k = k'+1$ and $\mathtt{AEx}_C^{k'}(\mathtt{AEx}_C^{k'}(\Phi)) = \mathtt{AEx}_C^{k'}(\Phi)$ (induction hypothesis). We have $\mathtt{AEx}_C^{k}(\mathtt{AEx}_C^{k}(\Phi)) = \mathtt{AEx}_C^{2}(\mathtt{AEx}_C^{k'}(\mathtt{AEx}_C^{k'}(\Phi))) \overset{IH}{=} \mathtt{AEx}_C^{2}(\mathtt{AEx}_C^{k'}(\Phi)) = \mathtt{AEx}_C^{(}\mathtt{AEx}_C^{k'+1}(\Phi)) = \mathtt{AEx}_C^{(}\mathtt{AEx}_C^{k}(\Phi)) = \mathtt{AEx}_C^{k+1}(\Phi)$. By Proposition 49.56 with $l := 0$, it is equal to $\mathtt{AEx}_C^{k}(\Phi)$ which is what we were looking for.

□

§49.59 I leave a small open question: is there always a $k \in \mathbf{N}$ such that $\mathtt{AEx}^{\infty}(\Phi)$ for any constellation Φ or if there is a constellation always leaving new pairs of matchable rays after abstract execution (by using the dynamics of internal colours).

§49.60 Hyper execution is not necessarily defined. It is possible that after abstract execution, there are always pairs of matchable rays. Hence, it is possible to never reach a normal form.

§49.61 **Example.** Consider the following constellation:

$$\Phi := [X, +f(X)] + [-f(-g(X)), -g(X)] + [+g(X), +g(X), a].$$

The two stars $[X, +f(X)]$ and $[-f(-g(X)), -g(X)]$ (which can be connected) are

duplicated in order to satisfy the rays of $[+g(X), +g(X), a]$. After abstract execution, we obtain $[-g(X), -g(X), a] + [-g(X), -g(X), a]$. However, these stars can also be connected to $[X, +f(X)] + [-f(-g(X)), -g(X)]$ from Φ. Applying abstract execution again will create more occurrences of $[-g(X), -g(X), a]$.

50 Concrete execution

§50.1 In his first paper on TS, Girard already defined an effective way to execute constellations [Gir17, Section 2.3] by giving a method to construct diagrams. However, it was only valid for the deterministic case with tree-shaped diagrams, which is very simple. In the deterministic case, all rays are uniquely linked. The general idea is to find an *iterative* way to construct the saturated and correct diagrams of abstract execution. We extend Girard's idea to branching rays in which a ray can be dual to several other rays.

§50.2 **Definition** (Construction space)**.** A *construction space* is an expression

$$\Phi \vdash_C \Delta \mid \Psi$$

where $C \subseteq F_+ \uplus F_-$ is a set of colours, $i \in \mathbf{N}$, Φ and Ψ are constellations and Δ is a set of diagrams where two diagrams are identified by diagram equivalence (*cf.* Definition 49.20). We write $\vdash$ for $\vdash_{F_+ \uplus F_-}$.

§50.3 The idea is that $\Phi \vdash \Delta \mid \Psi$ represents a state during the process of the construction of all diagrams for Φ (which is the constellation we would like to execute). The set Δ is the set of all diagrams being constructed incrementally. Finally, the constellation Ψ will contain the normal form of Φ at the end. When diagrams are saturated, they are evaluated by actualisation and put in Ψ when correct or are simply discarded.

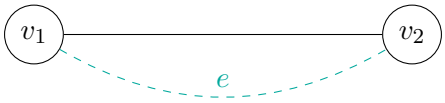
§50.4 **Definition** (Diagram extension)**.** Let (G, δ) be a C-diagram of a constellation Φ for a set of colours $C \subseteq F_+ \uplus F_-$ with $G := (V, E, \mathtt{end})$. We define two operations extending diagrams with a new edge (both illustrated in Figure 50.1):

◇ **Internal extension** $(G, \delta) \overset{(v,j)}{\oplus}_{\mathtt{in}} (v', j')$ with $v, v' \in V$ and $(\delta(v), j), (\delta(v'), j') \in \pm\mathtt{IdRays}(\Phi)$ only defined when

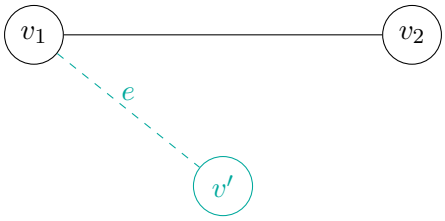
- $(\delta(v'), j') \in \mathtt{adj}^C_\Phi(\delta(v), j)$ and
- $(\delta(v), j) \in \mathtt{free}(G, \delta)$,

which defines a new diagram (G', δ') such that $G' := (V, E', \mathtt{end}')$ with:

- $E' := E \cup \{e\}$ for a fresh e;
- $\mathtt{end}'(e) = \{v, v'\}$ and $\mathtt{end}'(x) = \mathtt{end}(x)$ otherwise;
- $\delta'(e) = \{(\delta(v), j), (\delta(v'), j')\}$ and $\delta'(x) = \delta(x)$ otherwise.



(a) Internal extension with an edge e of a diagram with vertices v_1, v_2.



(b) External extension with an edge e of a diagram with vertices v_1, v_2. A new vertex v' is added for a new occurrence of star of the constellation.

Figure 50.1: Diagram extensions.

This operation connects two rays of a same diagram together.

◇ **External extension** $(G,\delta) \overset{(v,j)}{\oplus}_{\texttt{out}} (i',j')$ with $v, v' \in V$ and

$$(\delta(v), j), (i', j') \in \pm\texttt{IdRays}(\Phi)$$

only defined when

- $(i', j') \in \texttt{adj}^C_\Phi(\delta(v), j)$ and
- $(\delta(v), j) \in \texttt{free}(G, \delta)$,

which defines a new diagram (G', δ') such that $G' := (V', E', \texttt{end}')$ with:

- $V' := V \cup \{v'\}$ for a fresh v;
- $E' := E \cup \{e\}$ for a fresh e;
- $\texttt{end}'(e) = \{v, v'\}$ and $\texttt{end}'(x) = \texttt{end}(x)$ otherwise;
- $\delta(v') = i'$, $\delta'(e) = \{(\delta(v), j), (i', j')\}$ and $\delta'(x) = \delta(x)$ otherwise.

This operation connects a ray of the diagram (G, δ) with a new occurrence of star from the constellation Φ.

§50.5 **Definition** (Stellar construction)**.** Let $\Phi \vdash_C \Delta | \Psi$ be a construction space. A step of *stellar construction* takes all diagrams in Δ and extend it with:

$$(\Phi \vdash_C \Delta \mid \Psi) \rightsquigarrow (\Phi \vdash_C \Delta' \mid \Psi')$$

where:

- if $\texttt{SatDiags}(\Delta)$ is the set of saturated diagrams in Δ then

$$\Psi' := \Phi' \uplus \Downarrow \texttt{SatDiags}(\Delta);$$

- for all $(G, \delta) \in \Delta - \texttt{SatDiags}(\Delta)$ such that $G := (V, E, \texttt{end})$, we select non-deterministically a ray $(i, j) \in I_{\delta(v)} \cap \texttt{free}(G, \delta)$ for some $v \in V$;
- for all $(i', j') \in \texttt{adj}^C_\Phi(i, j)$, we have $(G, \delta) \overset{(i,j)}{\oplus}_{\texttt{out}} (i', j') \in \Delta'$;
- for all $(\delta(v'), j') \in \texttt{adj}^C_\Phi(i, j)$ such that $v \in V$ and $(\delta(v'), j) \in \texttt{free}(G, \delta)$, we have $(G, \delta) \overset{(i,j)}{\oplus}_{\texttt{in}} (\delta(v'), j') \in \Delta'$.

As usual, we write $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$.

§50.6 **Definition** (Concrete execution)**.** Let Φ be a constellation. Its concrete execution is defined by a constellation $\texttt{CEx}_C(\Phi)$ such that

$$(\Phi \vdash_C (G_1, \delta_1), ..., (G_n, \delta_n) \mid \emptyset) \rightsquigarrow^* (\Phi \vdash_C \emptyset \mid \texttt{CEx}_C(\Phi))$$

and:

1. for $i \in I_\Phi$ with $1 \leq i \leq n$, (G_i, δ_i) where $G_i := (V_i, E_i, \texttt{end}_i)$ is a diagram containing only the star $\Phi[i]$, *i.e.* $V := \{v_i\}$, $\delta(v_i) = i$ and $E := \emptyset$;
2. no step of stellar construction can be applied on $(\Phi \vdash_C \emptyset \mid \texttt{CEx}_C(\Phi))$ (we say that the construction space is in *normal form*).

§50.7 **Theorem** (Equivalence between abstract and concrete execution)**.** Let Φ be a constellation and $C \subseteq F_+ \uplus F_-$ be a set of colours. When $\texttt{CEx}_C(\Phi)$ is defined, we have $\texttt{AEx}_C(\Phi) = \texttt{CEx}_C(\Phi)$.

Proof. If $\texttt{CEx}_C(\Phi)$ is defined then $(\Phi \vdash_C (G_1, \delta_1), ..., (G_n, \delta_n) \mid \emptyset) \rightsquigarrow^* (\Phi \vdash_C \emptyset \mid \texttt{CEx}_C(\Phi))$ with $\Phi \vdash_C \emptyset \mid \texttt{CEx}_C(\Phi)$ in normal form. We show that for construction space $(\Phi \vdash_C \Delta' \mid \Psi')$ occurring during concrete execution, Δ' contains all diagrams of Φ with n edges, Ψ' all actualisation of saturated diagrams of Φ with $n - 1$ edges for n the number of steps done starting from the initial construction space. By induction on n.

◇ **Base case** Assume $n = 0$. We have the initial construction space. By definition, the diagrams of Δ' are all diagrams of 0 edge. There is no saturated diagram of size -1 hence we have $\Psi' := \emptyset$;

◇ **Induction case** Assume $n = n' + 1$. Assume we have $(\Phi \vdash_C \Delta' \mid \Psi')$ with Δ' containing diagrams of n' edges and Ψ' actualisation of saturated diagrams of $n' - 1$ edges. We have one more step and obtain $(\Phi \vdash_C \Delta'' \mid \Psi'')$. By definition of stellar construction, Δ'' extends diagrams of Δ' (which contains all diagrams of n' edges) by an edge (either by connecting two vertices/stars of the diagram or adding another

occurrence of star from Φ), hence Δ'' contains all diagrams of $n'+1$ edges. Again, by definition of stellar construction, a step actualise all saturated diagrams of Δ' and put them in Ψ'' which hence contains all actualisation of saturated diagrams of $n'-1+1=n'$ edges.

□

§50.8 **Remark.** Concrete execution is only defined when the construction of diagrams is terminating. Hence, concrete execution is weaker than abstract execution in some sense since abstract execution can compute infinite constellations and is always defined. Concrete execution only produces finite constellations.

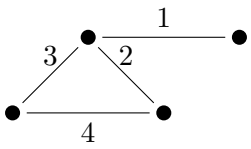
51 Interactive execution

§51.1 When Girard presented stellar resolution (which was simply called "stars and constellations"), his execution was something along the lines of concrete execution. Concrete execution (in the deterministic case) is thus *Girard's original notion of execution*. However, although concrete execution is rather intuitive and natural, it is only few months before my PhD defence that I defined it in my manuscript. When I first worked with Thomas Seiller on stellar resolution, we only worked with abstract execution because it was sufficient[2]. Since the beginning, I wanted to build something of a more computational fashion and I tried to implement it in Haskell[3]. This is what is presented in the section under the name of "interactive execution". Oddly enough, it is also few months before my defence that is has been properly and formally defined.

§51.2 The idea is not so different from concrete execution. Instead of constructing diagrams incrementally, we directly construct the actualisation of diagrams incrementally, by making stars interact by fusion "on the fly". The big difference is that we lose the structure of graph and in particular we are not able to distinguish two stars which came from the same diagram (as in concrete execution which identifies diagrams up to isomorphism). Interactive execution hence considers different *execution traces* in which diagrams are not incrementally constructed in the same order. This is a sort of *directed* version of concrete execution. The drawback is that we may end up with a lot of redundancies in the normal form but we can cope with that by selecting the right initial stars and not necessarily all stars of the constellation we would like to execute. Typically, if we implemented an automata, we would choose the subconstellation representing the initial state.

[2] It also seems that this choice was influenced by Thomas Seiller's mathematical taste. He liked the presentation of diagrams as graph homomorphism.

[3] I called the program "Large Star Collider". I'm currently working on another implementation in OCaml instead (because I wanted to practice OCaml).



(a) A particular diagram with 4 vertices corresponding to stars linked by 4 edges representing links between rays of those stars.

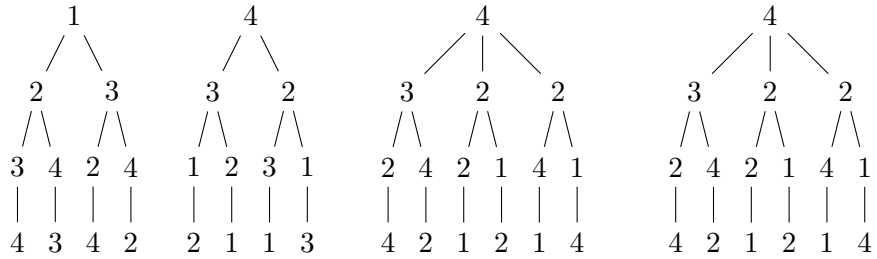


Figure 51.1: A diagram and 4 ways to construct it by iteratively adding edges.

§51.3 **Definition** (Interactive configuration)**.** An *interactive configuration* is an expression

$$\Phi \vdash_C \Psi$$

where Φ and Ψ are constellations, and $C \subseteq F_+ \uplus F_-$ is a set of colours. We write $\vdash$ instead of $\vdash_C$ when $C = F_+ \uplus F_-$.

The constellation Φ is called *reference constellation* and Ψ is called *interaction space*.

§51.4 In an interactive configuration $\Phi \vdash_C \Psi$, the reference constellation Φ corresponds to the constellation we would like to execute and the interaction space Ψ corresponds to a space where stars will interact in order to construct the result (corresponding to partial evaluations of diagrams). Interaction configurations can be transformed by some rules corresponding to the iterative construction of diagrams. Those transformations are dependent on a selected ray. The goal is to trigger interaction with other rays until no interaction is possible anymore in the interaction configuration (this corresponds to the property of saturation for diagrams).

§51.5 It may technically be possible to filter out the superfluous stars in the normal form by using arguments from combinatorics. We need to make all stars of the constellation initial (hence we start with $\Phi \vdash_C \Phi$) and for all possible diagrams we need to think about how many ways/order a diagram can be constructed from, by iteratively adding edges. An example of diagram with 4 ways to construct it is given in Figure 51.1.

§51.6 **Definition** (Set of matchable rays)**.** Let Φ be a constellation, $C \subseteq F_+ \uplus F_-$ a set of colours and r any ray. We define the *set of rays identifiers matchable with* r in Φ

w.r.t. C by:

$$\mathtt{mat}^C_\Phi(r) := \{(i,j) \in \pm\mathtt{IdRays}(\Phi) \mid r \bowtie \Phi[i][j], \mathtt{colours}(r) \cup \mathtt{colours}(\Phi[i][j]) \subseteq C\}.$$

As usual, we simply write $\mathtt{mat}_\Phi(r)$ when $C = F_+ \uplus F_-$.

§51.7 **Definition** (Self-interaction)**.** Let $\Phi[i]$ be a star of a constellation Φ with $i \in I_\Phi$. We define the self-interaction $\overset{j,j'}{\triangleright}\Phi[i]$ of $\Phi[i]$ along the rays j and j' as the star ϕ such that $I_\phi := \theta(I_{\Phi[i]} - \{j, j'\})$ with θ the solution of $\{\Phi[i][j] \overset{?}{=} \Phi[i][j']\}$.

§51.8 **Example.** Assume rays are indexed by natural numbers corresponding to their position in stars. We have that $\overset{0,1}{\triangleright}[+f(X), -f(g(X))]$ is undefined and

$$\overset{0,1}{\triangleright}[+c(X), -c(X), a] = [a].$$

§51.9 **Definition** (Stellar interaction)**.** We define *stellar interaction* for an interactive configuration $\Phi \vdash_C \Psi$ *w.r.t.* some set of colours $C \subseteq F_+ \uplus F_-$. Stellar interaction depends of some selected ray $(i,j) \in \pm\mathtt{IdRays}(\Psi)$. We set $\Psi := \Psi' + \Psi[i]$, hence the selected ray is $\Psi[i][j]$.

We have the following transition step:

$$\Phi \vdash_C \Psi' + \Psi[i]$$

$$\overset{(i,j)}{\rightsquigarrow} \quad \Phi \vdash \Psi' + \sum_{(i_k,j_k)\in\mathtt{mat}^C_\Phi(\Psi[i][j])} \Psi[i] \overset{j,j_k}{\triangledown} \Phi[i_k]$$

$$+ \sum_{(i,j_k)\in\mathtt{mat}^C_{\Psi[i]}(\Psi[i][j])} \overset{j,j_k}{\triangleright}\Psi[i]$$

where:

- the first sum corresponds to all possible ways to interact with a new occurrence of star $\Phi[i_k]$ from Φ along a free ray $\Psi[i][j]$ of $\Psi[i]$ which is matchable with some $\Phi[i_k][j_k]$;
- the second sum corresponds to all possible ways to connect two free matchable rays in the same star $\Psi[i]$ (representing a partial diagram being constructed).

When a unification error occurs for $\psi \overset{k,k'}{\triangledown} \psi'$ or $\overset{k,k'}{\triangleright}\psi$ then they disappears from the interaction space. We write $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$. We say that $\Phi \vdash_C \Psi$ is in *normal form* when no step of stellar interaction can be applied anymore.

§51.10 **Definition** (Interactive execution)**.** Let Φ a constellation and $C \subseteq F_+ \uplus F_-$ a set of colours. We define its *interactive execution w.r.t.* an initial subconstellation Ψ of Φ by a constellation $\texttt{IEx}_C(\Phi, \Psi)$ such that $\Phi \vdash_C \Psi \rightsquigarrow^* \texttt{IEx}_C(\Phi, \Psi)$ and $\texttt{IEx}_C(\Phi, \Psi)$ is in normal form. We write $\texttt{IEx}_C(\Phi)$ for $\texttt{IEx}_C(\Phi, \Phi)$.

§51.11 **Example** (Logic program for addition)**.** We have

$$\Phi_{\mathbf{N}}^{2+2} := [+add(\overline{0}, Y, Y)] + [-add(X, Y, Z), +add(s(X), Y, s(Z))] + [-add(\overline{2}, \overline{2}, R), \overset{(2,0)}{} \overset{(2,1)}{R}]$$

with stars and rays indexed the natural indexing with natural numbers corresponding to their position. We put indexes on the top of rays to make things clearer.

- We compute $\texttt{IEx}_C(\Phi_{\mathbf{N}}^{2+2}, [-add(\overset{(0,0)}{\overline{2}}, \overline{2}, R), \overset{(0,1)}{R}])$ by first selecting the ray $(0, 0)$ of the interaction space.

Step	$\Psi[i][j]$	Configuration	$\texttt{mat}_\Phi^C$	$\texttt{mat}_{\Psi[i]}^C$
0	$(0, 0)$	$\Phi_{\mathbf{N}}^{2+2} \vdash [-add(\overline{2}, \overline{2}, R), R]$	$\{(1, 1)\}$	$\emptyset$
1	$(0, 0)$	$\Phi_{\mathbf{N}}^{2+2} \vdash [-add(s(0), \overline{2}, Z), s(Z)]$	$\{(1, 1)\}$	$\emptyset$
2	$(0, 0)$	$\Phi_{\mathbf{N}}^{2+2} \vdash [-add(0, \overline{2}, Z), s(s(Z))]$	$\{(0, 0)\}$	$\emptyset$
3		$\Phi_{\mathbf{N}}^{2+2} \vdash [\overline{4}]$	$\emptyset$	$\emptyset$

  - Step 0 : we only have the query. It is matchable with a ray of the recursion star. We apply a step of fusion;
  - Step 1 : we can only use the recursion star again (only matchable ray is $(1, 1)$ from Φ);
  - Step 2 : the remaining star is only matchable with $(0, 0)$ in Φ, we apply a last step of fusion.

  Remark that after "peeling" the query, we actually remove the potential non-determinism. When we reach 0 as first argument in the query, there is only one matchable ray which is $[+add(0, Y, Y)]$.

- Interestingly, if we start from the wrong initial constellation (or consider $\Phi_{\mathbf{N}}^{2+2} \vdash \Phi_{\mathbf{N}}^{2+2}$), we may end up with an unwanted computational behaviour.

Step	$\Psi[i][j]$	Configuration	$\mathtt{mat}^C_\Phi$	$\mathtt{mat}^C_{\Psi[i]}$
0	$(0,0)$	$\Phi^{2+2}_{\mathbf{N}} \vdash [+add(\overline{0}, Y, Y)]$	$\{(1,0)\}$	$\emptyset$
1	$(0,0)$	$\Phi^{2+2}_{\mathbf{N}} \vdash [+add(s(0), Z, s(Z))]$	$\{(1,0)\}$	$\emptyset$
2	$(0,0)$	$\Phi^{2+2}_{\mathbf{N}} \vdash [+add(\overline{2}, Z, \overline{Z+2})]$	$\{(1,0),(2,0)\}$	$\emptyset$
3	$(0,0)$	$\Phi^{2+2}_{\mathbf{N}} \vdash [+add(\overline{3}, Z, \overline{Z+3})] + [\overline{4}]$	$\{(1,0)\}$	$\emptyset$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$K{+}1$	$(0,0)$	$\Phi^{2+2}_{\mathbf{N}} \vdash [+add(\overline{K}, Z, \overline{Z+K})] + [\overline{4}]$	$\{(1,0)\}$	$\emptyset$

This is not so surprising since there are infinitely many saturated diagrams which can be constructed but only one correct. Interactive execution has to compute all diagrams iteratively.

§51.12 **Observation** (Logic programs)**.** The observation is that in order to correctly execute programs, we need control over the interactive execution of stellar resolution. This corresponds to the fact that Prolog uses SLD-resolution which is a controlled version of Robinson's original resolution. In particular, the interactive configuration must only have the query in its interaction space.

§51.13 **Remark.** In an interactive configuration $\Phi \vdash_C \Psi$, the reference constellation Φ is *non-linear* (infinite supply of stars) and the interaction space Ψ is *linear* (stars are linearly consumed). This is reminiscent of intuitionistic sequents $!\Gamma \vdash_C \Delta$ in linear logic.

§51.14 **Example** (Basic sanity check)**.** In this example we look at non-trivial constellations to make sure that our interpretation works as expected.

- The constellation $\Phi := [\overset{(0,0)}{-a(X)}, \overset{(0,1)}{+a(X)}]$ has infinitely many saturated correct circular closed diagrams all actualising to $[]$.

Step	$\Phi[i][j]$	Configuration	$\mathtt{mat}^C_\Phi$	$\mathtt{mat}^C_{\Psi[i]}$
0	$(0,0)$	$\Phi \vdash [-a(X), +a(X)]$	$\{(0,1)\}$	$\{(0,1)\}$
1	$(0,0)$	$\Phi \vdash [-a(X), +a(X)] + []$	$\{(0,1)\}$	$\{(0,1)\}$
2	$(0,0)$	$\Phi \vdash [-a(X), +a(X)] + [] + []$	$\{(0,1)\}$	$\{(0,1)\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
K	$(0,0)$	$\Phi \vdash [-a(X), +a(X)] + [] + \overset{K}{\ldots} + []$	$\{(0,1)\}$	$\{(0,1)\}$

- We have

$$\Phi := [\overset{(0,0)}{X}, \overset{(0,1)}{+a(X)}] + [\overset{(1,0)}{-a(1)}, \overset{(1,1)}{+b(1)}] + [\overset{(2,0)}{-a(0)}, \overset{(2,1)}{+b(0)}] + [\overset{(3,0)}{-b(1)}, \overset{(3,1)}{-b(0)}]$$

  which should have for normal form $[0,1]+[1,0]$.

Step	$\Psi[i][j]$	Configuration	$\mathtt{mat}^C_\Phi$	$\mathtt{mat}^C_{\Psi[i]}$
0	$(0,1)$	$\Phi \vdash [X, +a(X)]$	$\{(1,0),(2,0)\}$	$\emptyset$
1	$(0,1)$	$\Phi \vdash [0, +b(0)] + [1, +b(1)]$	$\{(3,1)\}$	$\emptyset$
2	$(0,1)$	$\Phi \vdash [0, -b(1)] + [1, +b(1)]$	$\{(1,1)\}$	$\emptyset$
3	$(1,1)$	$\Phi \vdash [0, -a(1)] + [1, +b(1)]$	$\{(3,0)\}$	$\emptyset$
4	$(1,1)$	$\Phi \vdash [0, -a(1)] + [1, -b(0)]$	$\{(2,1)\}$	$\emptyset$
5	$(0,1)$	$\Phi \vdash [0, -a(1)] + [1, -a(0)]$	$\{(0,1)\}$	$\emptyset$
6	$(1,1)$	$\Phi \vdash [0,1] + [1, -a(0)]$	$\{(0,1)\}$	$\emptyset$
7		$\Phi \vdash [0,1] + [1,0]$	$\emptyset$	$\emptyset$

- If we reduce deterministic links in the previous example, we obtain the constellation $\Phi := [X, -a(X)] + [+a(0), +a(1)]$. What happens if we start from $\Phi[1][0]$?

Step	$\Psi[i][j]$	Configuration	$\mathtt{mat}^C_\Phi$	$\mathtt{mat}^C_{\Psi[i]}$
0	$(0,1)$	$\Phi \vdash [+a(0), +a(1)]$	$\{(0,1)\}$	$\emptyset$
1	$(0,0)$	$\Phi \vdash [+a(0), 1] + [0, +a(1)]$	$\{(0,0)\}$	$\emptyset$
2	$(1,1)$	$\Phi \vdash [0,1] + [0, +a(1)]$	$\{(0,0)\}$	$\emptyset$
3	$(1,1)$	$\Phi \vdash [0,1] + [0,1]$	$\emptyset$	$\emptyset$

- Consider the constellation $\overset{(0,0)}{[+a(0)]} + \overset{(1,0)}{[+a(1)]} + [\overset{(2,0)}{X}, \overset{(2,1)}{-a(X)}, \overset{(2,2)}{-b(X)}] + \overset{(3,0)}{[+b(0)]} + \overset{(4,0)}{[+b(1)]}$ with normal form $[0]+[1]$.

Step	$\Psi[i][j]$	Configuration	$\mathtt{mat}^C_\Phi$	$\mathtt{mat}^C_{\Psi[i]}$
0	$(0,0)$	$\Phi \vdash [+a(0)]$	$\{(2,2)\}$	$\emptyset$
1	$(0,1)$	$\Phi \vdash [0, -b(0)] + [1, -b(1)]$	$\{(3,0)\}$	$\emptyset$
2	$(1,1)$	$\Phi \vdash [0] + [1, -b(1)]$	$\{(4,0)\}$	$\emptyset$
3		$\Phi \vdash [0] + [1]$	$\emptyset$	$\emptyset$

§51.15 **Lemma** (Diagram extension by interaction)**.** Let (G, δ) be a C-diagram for a constellation Φ. We have:

- $\Downarrow \left((G,\delta) \overset{(v,j)}{\oplus}_{\texttt{in}} (v', j')\right) = \Downarrow (G,\delta) \overset{j,j'}{\triangledown} \Phi[\delta(v')]$;
- $\Downarrow \left((G,\delta) \overset{(v,j)}{\oplus}_{\texttt{out}} (i', j')\right) = \overset{j,j'}{\triangleright} \Downarrow (G,\delta)$.

Proof. By Theorem 49.38, a full contraction of a diagram can lead to the same result as its actualisation. We can then treat the evaluation of diagrams with diagram contraction. Consider the full contraction of the extended diagram. By the confluence of diagram contraction (*cf.* Corollary 49.36), it is possible to contract all edges except the edge e linking the ray index j and j' without any effect on the result. At the end we contract e. This exactly corresponds to the computation of $\Downarrow (G,\delta) \overset{j,j'}{\triangledown} \Phi[\delta(v')]$ for external extension and $\overset{j,j'}{\triangleright} \Downarrow (G,\delta)$ for internal extension. □

§51.16 **Theorem** (Relation between concrete and interactive execution)**.** Let Φ be a constellation and $C \subseteq F_+ \uplus F_-$ be a set of colours. If $\texttt{CEx}_C(\Phi)$ is defined, then

$$\texttt{IEx}_C(\Phi) \approx_\alpha \texttt{CEx}_C(\Phi) \uplus \Phi$$

where Φ' only contains occurrences (possibly none) of stars of $\texttt{CEx}_C(\Phi)$.

Proof. If $\texttt{CEx}_C(\Phi)$ is defined then $(\Phi \vdash_C (G_1,\delta_1), ..., (G_n,\delta_n) \mid \emptyset) \rightsquigarrow^n (\Phi \vdash_C \emptyset \mid \texttt{CEx}_C(\Phi))$ with $\Phi \vdash_C \emptyset \mid \texttt{CEx}_C(\Phi)$ in normal form for some $n \in \mathbf{N}$. By induction on n we show that for all construction spaces $(\Phi \vdash_C \Delta \mid \Psi)$ occurring during the concrete execution of Φ, we have a sequence of stellar interaction steps $(\Phi \vdash_C \Phi) \rightsquigarrow^* (\Phi \vdash_C \Downarrow \Delta \uplus \Psi)$.

◇ **Base case** Assume $n = 0$. We start from the initial construction space $(\Phi \vdash_C (G_1,\delta_1), ..., (G_n,\delta_n) \mid \emptyset)$. We do 0 step of stellar interaction and have $\Phi \vdash_C \Phi$. We indeed have $\Phi := \Downarrow \Delta \uplus \emptyset$ since Δ is the set of all diagrams of 0 edges (corresponding to the stars of Φ).

◇ **Induction case** Assume $n = n' + 1$ for some n'. We reached some construction space $\Phi \vdash_C \Delta_{n'} \mid \Psi_{n'}$ after n' steps and know that it can be simulated by a sequence of steps of stellar interaction $(\Phi \vdash_C \Phi) \rightsquigarrow^* (\Phi \vdash_C \Downarrow \Delta_{n'} \uplus \Psi_{n'})$. We add a new step of concrete execution and obtain $\Phi \vdash_C \Delta_{n'+1} \mid \Psi_{n'+1}$. We show that we can extend our interactive execution in order to obtain $\Phi \vdash_C \Downarrow \Delta_{n'+1} \uplus \Psi_{n'+1}$. By Lemma 51.15, what stellar interaction does on $(\Phi \vdash_C \Downarrow \Delta_{n'} \uplus \Psi_{n'})$ is basically computing the application of external and internal extensions of diagrams all in $\Delta_{n'}$, producing $\Delta_{n'+1}$. Since some stars cannot be extended because they correspond to the saturated diagram of $\Delta_{n'}$. When they are reunited with $\Psi_{n'}$ (corresponding to actualisations of saturated diagrams in concrete execution), we obtain $\Phi_{n'+1}$.

Since stellar interaction does not identify stars coming from equivalent diagrams (unlike concrete execution), we must have several duplicate stars corresponding to actualisation of equivalent diagrams, corresponding to the constellation Φ'. □

§51.17 **The case of internal polarities.** Interactive execution does not behave like abstract or concrete execution. Interactive execution construct diagrams *dynamically* and in an iterative way. Hence, we do not work with diagrams as *fixed and immutable structures* as with abstract execution. A consequence is that interactive polarities can work with subjective constellations without the need for repeated executions. Even if we create polarised rays by "unlocking" internal polarities, it is possible to make those new unlocked rays interact. However, the strategy we defined is not sufficient, as we will see in the next example.

§51.18 **Example.** We try to execute the constellation

$$\Phi := [-f(\overset{(0,0)}{+g(X)})] + [\overset{(1,0)}{X}, \overset{(1,1)}{+f(X)}] + [\overset{(2,0)}{-g(X)}, \overset{(2,1)}{+f(X)}, \overset{(2,2)}{a}]$$

which contains internal polarities.

Step	$\Psi[i][j]$	Configuration	$\mathtt{mat}^C_\Phi$	$\mathtt{mat}^C_{\Psi[i]}$
0	$(0,0)$	$\Phi \vdash [-f(+g(X))]$	$\{(1,1),(2,1)\}$	$\emptyset$
1	$(0,0)$	$\Phi \vdash [+g(X)] + [-g(+g(X)),a]$	$\{(2,0)\}$	$\emptyset$
2	$(0,0)$	$\Phi \vdash [+f(X),a] + [-g(+g(X)),a]$	$\{(0,0)\}$	$\emptyset$
3		$\Phi \vdash [a] + [-g(+g(X)),a]$	$\emptyset$	$\emptyset$

The problem appears at the second step. The ray $+g(X)$ has been extracted from $-f(+g(X))$ and can interact with the ray $-g(X)$. This was not possible with abstract and concrete execution. However, this does not faithfully simulate hyperexecution since we could not connect $[+g(X)]$ with $[-g(+g(X)),a]$, which corresponds to the reunion of two evaluated diagrams. In the previous example, we thus have a ray $[-g(+g(X)),a]$ left although it should have been eliminated. A simple solution is to allow the interaction between two stars of the interaction space. This shows that non-trivial considerations have to be made in order to handle internal polarities.

§51.19 **About the design of interactive execution.** When I first defined interactive execution, I actually tried to make it match exactly with abstract execution by adding information on rays such as a unique occurrence identifier and the origin of rays (from which star of the initial constellation it comes from). The point was to identify stars coming from the same diagram constructed in a different way. However, we could always find counter-example and in the end the best way was to... keep the whole structure of diagram in memory. This is actually what splitted interactive execution and what led to the definition of concrete execution which came later.

§51.20 **Execution trace in interactive execution.** If we compute $\Phi \vdash \Phi$ we may compute several equivalent diagrams but they will lead to several occurrence of a same star. The reason is that those diagrams are not constructed in the same order. Although it may be seen as a technical drawback, it can be seen as a feature: we can put "directionality" in computation by choosing initial stars to start with. The strategy is usually to do a wise selection of exactly one star and one ray to start with for each connected component of the constellation we would like to execute. Some choice will lead to a well-defined result and other may lead to infinite loops.

52 Difference with Girard's stars and constellations

§52.1 The stellar resolution defined in this chapter is not exactly Girard's original model of stars and constellations defined in his sixth paper on GoI [Gir13a] and first paper on TS [Gir17]. Thomas Seiller and I designed some changes to make it more general but also to solve some technical problems and errors which occurred in formal definitions of the interpretation of MLL (that Girard did not formally established).

§52.2 **Restriction to uniform stars.** In Girard's original definition, rays of stars have some specific restrictions:

- they must have exactly the same variables. For instance $[X, +c(X)]$ is a correct star but not $[f(X,Y), +c(X)]$. This requirement is sufficient for MLL and is much simpler to study but it is later relaxed in order to treat exponentials [Gir16b];
- they must be pairwise non-matchable in order to avoid loops (cycles of size > 1 are still allowed). It simplifies a lot of things but in order to have more expressivity (so that we can interpret logic programs and other things), we choose to remove this requirement.

§52.3 **Constellations as indexed families instead of sets.** In Girard's original definition, constellations are sets. Similarly to Wang tiles, it is actually sufficient to consider sets because stars of a given constellation can be duplicated in diagrams anyway. However, in our stellar resolution, we are not interested in diagrams themselves (which correspond to tilings) but in their evaluation. If two diagrams δ and δ' of a constellation Φ are both evaluated to a same star ϕ then we would like ϕ to appear in $\mathtt{AEx}(\Phi)$ to keep the quantitative information that two different diagrams produced ϕ. Another difference is that the several occurrences of a same star actually matter in a constellation because two distinct diagrams can be constructed from the dependency graph. We have $\mathtt{AEx}([X, +c(X)] + [-c(X)]) = [X]$ but $\mathtt{AEx}([X, +c(X)] + [-c(X)] + [-c(X)]) = [X] + [X]$. It is not clear whether this feature is wanted or not but we leave it in our model.

§52.4 **Removal of the empty star.** Girard does not consider valid the empty star [] because it cannot be connected to another star. However, we choose to allow it because it still gives information: it comes from a correct diagram which had no free rays. But can

still recover Girard's execution by using the operator $\flat$ (*cf.* Definition 49.46) on the normal form to remove the empty star. This is useful when we do not want superfluous information.

§52.5 **Restriction to tree-like diagrams.** Girard only considers tree-like diagrams because they are sufficient to interpret logic. However, we choose to allow cyclic diagrams since they allow to express tile systems (tilings can be circular). We will see that it is also necessary to correctly interpret MLL proofs in an effective way (and that Girard's original definition is not exactly sufficient).

§52.6 **Different handling of maximality of diagrams.** As in the GoI, we need to consider maximal paths. In Girard's original definition, all possible connected tree-like diagrams are considered (without requirement of saturation). In a second step, all stars containing coloured rays are erased because coloured rays in the normal forms are considered as signs of incomplete computation. This forces diagrams to add as many stars as possible to fill the holes opened by coloured rays, thus only evaluations of maximal diagrams are considered. For more generality and to link the model with Seiller's works on interaction graphs [Sei16b], we allow coloured rays in the normal form (in exchange of a notion of saturation). This allows to compose executed constellations. Girard's execution can still be recovered by using the operator $\lightning$ (*cf.* Definition 49.44).

53 Discussion: comparison with other notions

§53.1 A question that I received several times is "is not stellar resolution exactly logic programming?". Actually, if we remove some technical features such as internal colours or unpolarised rays, our model exactly corresponds to the semantics of first-order resolution with disjunctive clauses (*cf.* Section 23). But as far as I know, all models based on this semantics of term unification usually put *control* over it (SLD-resolution) or make the model subject to a logical interpretation whereas our model is purely computational, without any reference to any logic (Robinson's resolution makes the expressions coincide with classical predicate calculus). Stellar resolution is unique and I believe the reason is simple: who would need something so complicated and chaotic? We usually design a model of computation a practical need but in Girard's case, he was looking for a very flexible model in which he could express generalised proof-nets. We can still try to compare our model to other approaches in the field of logic programming and automated reasoning.

§53.2 **Original first-order resolution.** The closest model which comes to mind is simply Robinson's resolution logic. We use almost identical objects. We add unpolarised rays (but we can actually simulate it by using unused special predicate symbols). In resolution, we are usually interested in the reachability of the empty clause ([] in our case) representing a contradiction. In the stellar resolution, it does not have any meaning

and we use objects as query-free logic programs. Moreover, usual resolution is limited to tree derivations (corresponding to tree-like diagrams) whereas stellar resolution allows cyclic diagrams in order to simulate tile systems. There are graph-based models [Sic76, Kow75, EO91] which are very similar to stellar resolution but they are still different for the reasons mentioned above.

§53.3 **Horn clauses and logic programming.** In logic programming, we are usually interested in answering a query (or several queries) represented by a first-order atom (such as in Prolog, for instance). In order to answer the query, logic programming uses a backward reasoning by going up from the unique conclusion to the premises. Stellar resolution is naturally query-free (although queries can be simulated, there is no such distinguished objects). In particular, we can have several outputs and we do not distinguish between input and output. For instance, if we have a star representing an implication $A \Rightarrow B$, then we can connect a star to the output and only the input will survive. This does not make sense in logic programming because a direction from inputs to outputs is enforced in inference.

§53.4 **Stable model semantics.** There are several languages based on stable model semantics such as disjunctive logic programming [Min94, LRM91] itself based on a subset of Prolog called Datalog. The notion of stable model is also the basis of answer set programming (ASP) [Gel08, EIK09]. In these languages, a primitive handling of logical negation is used whereas we want our model to be purely computational, without any reference to logic.

§53.5 **Resolution operator.** In appearance, execution is very similar to the resolution operator [Lei12, Chapter 3] which is analogous to the consequence operator [DEGV01, Section 2.2] of logic programming. This operator computes a *full inference* from a given set of clauses. The difference is that we allow cyclic diagrams which makes our model closer to the construction of tilings in tile systems. As we later show when interpreting logic programs, allowing cyclic diagrams still preserves the interpretation of Horn clauses since cyclic diagrams are often wrong for logic programs: for the constellation $\Phi_{\mathbf{N}}^{n+m}$ of Example 48.18, a loop can be constructed with

$$[-add(X,Y,Z), +add(s(X),Y,s(Z))],$$

leading to the equation $X \stackrel{?}{=} s(X)$ which has no solution.

§53.6 **Lafont's interaction nets.** A remark I received is that stellar resolution may be equivalent to Lafont's interaction nets [Laf89] which are known to generalise proof-nets (and even proof-structures) [Laf95] and which can be simplified into a minimal set of interaction combinators [Laf97]. I admit that stellar resolution is indeed very close to interaction nets. I personally believe that stellar resolution can nicely encode interaction nets[4], however I have never been able to figure out how close these models were. If we

[4]This is something I investigated a little bit with Julien Marquet but we never reached a conclusion.

consider stars with internal colours then as far as I know there is no equivalent. Moreover, everything can interact in interaction nets whereas stellar resolution features unpolarised rays which cannot interact. But if we consider the objective polarised fragment of stellar resolution, it seems to me that it already contains non-trivial elements which cannot be represented by interaction nets which are purely based on structural graph rewriting. Typically, in stellar resolution, we have equation solving which passes simple messages based on first-order terms. Anyway, I think it can be interesting to connect our model to Lafont's nets.

§53.7 **Generalised token machines.** There exists several extensions of the IAM for GoI (*cf.* Section 36). For instance, Dal Lago, Tanaka and Yoshimizu defined a token machine for Mazza's multiport interaction combinators (a variant of the interaction combinators mentioned above) [DLTY17]. We can also mention Castellan and Clairambault's "Multi-token Geometry of Interaction" [CC23]. Another recent work which can be mentioned is Chardonnet, Valiron and Vilmart work on ZX-calculus, a graph-based model for quantum computing [CVV21]. Their machine can be related to the concrete or interactive execution of stellar resolution but I currently do not know how similar these models are. It is likely that ZX-calculus can be expressed purely from term unification over more sophisticated equations with complex coefficients (instead of graph rewriting or a token machine) but I do not know why one would do that.

Chapter 8

Illustrating stellar resolution

§53.8 It is breaktime. In this chapter, we play with stars and constellations in order to encode several models of computation. It demonstrates the expressivity and computational power of stellar resolution. Most models of computation only need objective stars.

§53.9 The encodings show that stellar resolution can naturally express the main classes of models of computation given in Chapter 2:

- state machines (including Turing machines);
- λ-calculi (from their translation to proof-nets) – this will come later, during the interpretation of linear logic;
- tile systems (Wang tiles but also the abstract tile assembly model appearing in Paragraph 17.7);
- logic programs (with Horn clauses);
- generalised circuits (subsuming boolean and arithmetic circuits).

§53.10 In this chapter we focus on *interactive execution* instead of abstract or concrete execution because concrete execution is closer to the real behaviour of classical models of computation. In some cases, we use abstract (or concrete) execution because it is more convenient (for instance for tile systems).

54 Flows, wirings and graphs

§54.1 We start with the encoding of flows and wirings (defined in Section 37) since they are very close to stellar resolution: stars are n-ary symmetric flows.

§54.2 The main problem with the encoding of flows is that flows are asymmetric objects. The composition of two flows $t \leftharpoonup u$ and $v \leftharpoonup w$ only connects u and v. In the meantime, even though we can use binary stars with polarities representing input and output, stars are natively symmetric. In the case of stars, t can be connected to u or even v. In order to encode the composition of flows, we have to restrict the possible connexions by distinguishing input and output.

§54.3 We fix a polarised signature $\mathbb{F} = (V, F, \mathtt{ar}, \frown, \lfloor\cdot\rfloor)$ for which we require the existence of colours $-S, +S, -T, +T \in C$ and function symbols S, T. The symbol S stands for *source* and T for *target*.

§54.4 **Definition** (Translation of flows and wirings)**.** Let $t \leftharpoonup u$ be a flow. Its translation into a star is defined by $(t \leftharpoonup u)^\bigstar := [S(t), T(u)]$.

The translation extends to wirings F by $F^\bigstar := \sum_{f\in F} f^\bigstar$.

§54.5 We could now treat $[S(t), T(u)]$ as a flow $t \leftharpoonup u$ and define composition exactly like the composition of flows. However, this is not very interesting. We are simply giving an alias to flows, an alternative notation but do not even use the mechanisms of stellar resolution. Another solution is to use the execution of stellar resolution to reproduce the composition of flows. However, because stars are symmetric and flows asymmetric, we need to play with polarities with external definitions (this is all we can do).

§54.6 **Definition** (Lateralisation)**.** We define two colouring functions μ_l and μ_r called *lateralisations*, defined by:

$$\mu^l_{\mathtt{lat}}[S(t), T(u)] := [-S(t), T(u)] \quad \text{and} \quad \mu^r_{\mathtt{lat}}[S(t), T(u)] := [S(t), +T(u)].$$

§54.7 Now remark that even if we lateralise $(t \leftharpoonup u)^\star$ and $(v \leftharpoonup w)^\star$ respectively to the right and to the left, we would obtain $[S(t), +T(u)]$ and $[-S(v), T(w)]$ which means that only $+T(u)$ and $-S(v)$ can interact. However, since they do not share the same colour, we need an intermediate star $[-T(X), +S(X)]$ linking the two star. This star behaves like a cut in linear logic and execution becomes cut-elimination.

§54.8 **Theorem** (Simulation of composition and product)**.** Let f and g be flows. We have:

1. $(fg)^\bigstar = \mathtt{AEx}(\mu^r_{\mathtt{lat}}(f^\bigstar) + \mu^l_{\mathtt{lat}}(g^\bigstar) + [-T(X), +S(X)])$ and
2. $(FG)^\bigstar = \mathtt{AEx}(\mu^l_{\mathtt{lat}}(F^\bigstar) \uplus \mu^r_{\mathtt{lat}}(G^\bigstar) + [-T(X), +S(X)])$.

Proof. By definition, if we have $f = t \leftharpoonup u$ and $g = v \leftharpoonup w$, then $fg = \theta t \leftharpoonup \theta w$ where $\theta := \mathtt{solution}\{u \stackrel{?}{=} v\}$. By the translation of flows, we have $(fg)^\bigstar = (\theta t \leftharpoonup \theta w)^\bigstar = [S(\theta t), T(\theta w)]$. As for the right-hand side of the equality, we obtain

$$\mathtt{AEx}\big(\mu^r_{\mathtt{lat}}(f^\bigstar) + \mu^l_{\mathtt{lat}}(g^\bigstar) + [-T(X), +S(X)]\big)$$

$$= \mathtt{AEx}([S(t), +T(u)] + [-S(V), T(W)] + [-T(X), +S(X)]).$$

The third star $[-T(X), +S(X)]$ will connect the two other stars and we obtain the normal form $[S(\theta t), T(\theta w)]$. As for the product of wirings, we have $(FG)^\bigstar = \{(fg)^\bigstar \mid f \in F, g \in G, fg \text{ is defined}\}$ by definition and

$$\mathtt{AEx}(\mu^l_{\mathtt{lat}}(F^\bigstar) \uplus \mu^r_{\mathtt{lat}}(G^\bigstar) + [-T(X), +S(X)])$$

$$= \{\mathtt{AEx}\big(\mu^r_{\mathtt{lat}}(f^\bigstar) + \mu^l_{\mathtt{lat}}(g^\bigstar) + [-T(X), +S(X)]\big) \mid f \in F, g \in G, fg \text{ is defined}\}.$$

We conclude with the simulation of flow composition proved above. □

§54.9 **Example.** We have $\big((X \leftharpoonup f(X))(Y \leftharpoonup h(Y))\big)^\bigstar = [S(X), T(h(f(X))]$ which is also equal to $\mathtt{AEx}\big([S(X), +T(f(X))] + [-S(Y), T(h(Y))] + [-T(X), +S(X)]\big)$.

§54.10 When interpreting proof-nets or encoding bidirectional automata as in Aubert and Bagnol's works [AB14], "biflows" $t \leftrightharpoons u := (t \leftrightharpoons u) + (u \leftrightharpoons t)$ are used instead in order to encode undirected edges in a graph (simple flows represent directed edges). Such biflows are translated into a constellation $(t \leftrightharpoons u)^\bigstar = [S(t), T(u)] + [S(u), T(t)]$ (by the definition above). In case of a path $\big((t \leftrightharpoons u)(t' \leftrightharpoons u')\big)(t'' \leftrightharpoons u'')$ such that u in only matchable with t', $\theta u'$ only with t'' where $\theta := \mathtt{solution}\{u \stackrel{?}{=} t'\}$, the composition will be computed in stellar resolution as follows:

$$\begin{gathered}\mathtt{AEx}\big([S(t), +T(u)] + [S(u), +T(t)] + [-S(t'), T(u')] + [-S(u'), T(t')]\big)\\ = [S(\theta t), T(\theta u')] + [S(\theta u'), T(\theta t)] = (\theta t \leftrightharpoons \theta u')^\bigstar\\ \mathtt{AEx}\big([S(\theta t), +T(\theta u')] + [S(\theta u'), +T(\theta t)] + [-S(t''), T(u'')] + [-S(u''), T(t'')]\big)\\ = [S(\psi\theta u'), T(\psi t'')] + [S(\psi t''), T(\psi\theta u')] = (\psi\theta u' \leftrightharpoons \psi t'')^\bigstar\end{gathered}$$

where $\psi := \mathtt{solution}\{\theta u' \stackrel{?}{=} t''\}$. Actually, with such use of flows, it is sufficient to use a simpler translation where biflows are translated into binary stars because of the introduction of symmetry.

§54.11 **Definition** (Encoding of biflows)**.** Let $t \leftrightharpoons u$ be a biflow. Its encoding is defined by $(t \leftrightharpoons u)^{\bigstar\!\!\!\!\!\!\bigstar} := [+F(t), +F(u)]$.

§54.12 We need to use a star $[-F(x), -F(x)]$ as a bridge between biflows. The previous computation becomes:

$$\begin{gathered}\mathtt{AEx}\big([+F(t), +F(u)] + [+F(u'), +F(t')] + [+F(u''), +F(t'')] + [-F(X), -F(X)]\big)\\ = [+F(\psi\theta u'), +F(\psi t'')] = \Big(\big((t \leftrightharpoons u)(t' \leftrightharpoons u')\big)(t'' \leftrightharpoons u'')\Big)^{\bigstar\!\!\!\!\!\!\bigstar}.\end{gathered}$$

By using these ideas, it is easy to derive an encoding of graphs where edges are represented by binary stars since this was the initial purpose of flows.

55 Encoding of logic programs with Horn clauses

§55.1 A natural illustration of the computational power of the stellar resolution is the encoding of logic programs since the stellar resolution directly generalises Robinson's first-order resolution which corresponds to the core of logic programming.

§55.2 First, it is possible to do programming with predicate calculus [Kow74]. It is then known that formulas of predicate calculus can be normalised so that formulas are represented only by conjunctions of disjunctions (called clauses) with only universal quantifiers appearing as prefix [Hed04, Section 3.2]. Formulas are then of the shape $\forall x_1, ..., x_n.(A_1^1 \vee ... \vee A_n^1) \wedge ... \wedge (A_1^m \vee ... \vee A_k^m)$ where every A_y^x is an atomic formula and where quantification is usually hidden. We use those normalised formulas of predicate calculus with at most one positive (without negation) atom in each clause. Such normalised formulas called *Horn clauses* represent sequents $\Gamma \vdash A$ for a set of hypotheses Γ (*cf.* Section 23).

§55.3 A *fact* is a closed (variable-free) first-order formula. Several facts form a *knowledge base*. We have *rules* which can be used to infer new facts from the available ones and thus expend the knowledge base. Rules are often represented as implications $A_1, ..., A_n \vdash B$ called *Horn clauses* [Hor51, Tär77]. A *query* asks if it is possible to infer a given fact from the knowledge base and is itself represented as a fact symbolising a goal. A *logic program* is a multiset of rules and facts.

§55.4 The translation is direct. We use the use the polarities to distinguish between hypothesis and conclusion (or input and output). The translation of a fact is defined by

$$P(t_1, ..., t_n)^\bigstar := [+P(t_1, ..., t_n)].$$

For a rule, the translation is defined by:

$$\big(\wedge_{i=1}^m P_1(t_1^i, ..., t_n^i) \vdash Q(u_1, ..., u_k) \big)^\bigstar := (\bigcup_{i=1}^m \{-P_1(t_1^i, ..., t_n^i)\}) \cup \{+Q(u_1, ..., u_k)\}.$$

Finally, for a query, we have:

$$(?P(t_1, ..., t_n))^\bigstar := [-P(t_1, ..., t_n), R_1, ..., R_m]$$

with $\{R_1, ..., R_m\} = \bigcup_{i=1}^n \texttt{vars}(t_i)$ which represents the information we would like to make visible in the output (see Example 48.18 where $[-add(\overline{n}, \overline{m}, r), r]$ is the query). A logic program $P := \biguplus_{i=1}^n \{C_i\}$ becomes $P^\bigstar := \bigcup_{i=1}^n \{C_i^\bigstar\}$.

§55.5 The set of answers for a query q on a program P is defined by a set of substitutions $A_P^q = \{\theta_1, ..., \theta_k\}$ such that for all $\theta \in A_P^q$, we have θq logically satisfied by P, written $P \vDash \theta q$. The answers are usually computed by iteratively applying the resolution rule between q and all possible $C \in P$ until either no variables remain in q or the resolution

rule is no more applicable. We refer to definitions of the SLD-resolution itself derived from Kowalski's SL-resolution [Kow74, KK71] for more details about the computation of answers.

§55.6 **Theorem** (Simulation of logic programs)**.** Let P be a logic program with query q and $P^\bigstar$ and $q^\bigstar$ be their translation. For $n > 0$ and $|P| > 0$, if $\lightning\flat\texttt{IEx}(P^\bigstar, q^\bigstar) = \theta_1\phi_1 + ... + \theta_n\phi_n$ for some θ_i is the answer of the query q then $P \vDash \theta_i q$.

Proof. The proof relies on the fact that

$$\lightning\flat\texttt{IEx}(P^\bigstar, q^\bigstar)$$

exactly implements SLD-resolution [Kow74, KK71] which is known to be sound and complete for Horn clauses (*cf.* Section 23).

1. SLD-resolution selects a clause to be resolved. It corresponds to the query. This clause is fixed, meaning that we can mark it with a label "to be resolved". The execution $\lightning\flat\texttt{IEx}(P^\bigstar, q^\bigstar)$ computes a normal form for $P^\bigstar \vdash q^\bigstar$ and in particular all fusion must applies to $q^\bigstar$ with corresponds to the selected clause;
2. SLD-resolution make the marked clause linearly interact with clauses of P. Similarly, interactive execution make the only polarised ray of $q^\bigstar$ interact with matchable rays of stars in $P^\bigstar$. The same rule of interaction is used: resolution corresponds to fusion. The obtained clause is still marked as query;
3. In case of non-determinism, the marked clause is duplicated and interact with several clauses which are all marked as new queries and we have $P^\bigstar \vdash q_1^\bigstar + ... + q_n^\bigstar$;
4. We repeat the procedure on each obtained marked clause q_i until we reach a fact. In this case, we end up with a substitution for the variables of the marked clause corresponding to the solution. If a marked clause cannot interact anymore without being resolved, it is erased. This is done by our operator $\lightning$.

Additionally, we have to ensure that our relaxation to cyclic diagrams do not cause problems. In logic programming, we usually require that the rays of a star have exactly the same variables (all variables are bound). Because of this restriction, cycles in dependencies graphs of logic programs, when reduced to a loop on a single star, either involve equations of the shape $t \stackrel{?}{=} t$ or equations of the type $X \stackrel{?}{=} f(X)$. In the former case, if the associated rule is binary, *i.e.* of the shape $A \vdash B$, we obtain the empty star [] which is irrelevant in the computation and removed by the operator $\flat$. If the rule is not binary, *e.g.* of the shape $A_1, A_2, ..., A_n \vdash B$, the equation $t \stackrel{?}{=} t$ associated with the loop is erased because it has no effect on the computation. In the latter case of the ill-behaving equation $X \stackrel{?}{=} f(X)$, the whole diagram is incorrect and ignored in the output. □

§55.7 **Remark** (Absence of solution)**.** The previous theorem assumed the existence of at least one solution for the query. If there are no solution at all for the query, then we obtain $\lightning\flat\texttt{IEx}(P^\bigstar, q^\bigstar) = \emptyset$. However we cannot say that we have $P \not\models \theta_i q$. In logic programming, we usually work with the "closed-world assumption", meaning that the absence of knowledge (P does not satisfy the query) entails the falsity of a clause.

§55.8 **Remark.** There are additional features which could be added. We could also add rays $x \cdot X$ where x is a constant representing the variable X in order to keep the name of variable in the output. We would finally obtain a normal form made of stars $\phi_i = \bigcup_{i=1}^{k} \{x_i \cdot r_i\}$ such that ϕ_i corresponds to some $\theta \in A_P^q$. By doing so, which value corresponds to which variable in the solution (as in Prolog).

56 State machines

§56.1 Definitions of automata and Turing machines are taken from Sipser's introduction to the theory of computation [Sip06].

Non-deterministic finite automata (NFA)

Let Σ be an alphabet and $w \in \Sigma^*$ a word on some alphabet Σ. A non-deterministic automaton (NFA) on Σ is a tuple $A = (\Sigma, Q, Q_0, \Delta, F)$ where Q is the set of states, Q_0 and $F \subseteq Q$ are respectively the set of initial and final states, and $\Delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function where $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$. The purpose of automata is to recognise languages by applying the transition function on the initial state and reaching either a final or non-final state.

A *partial run* in A is a sequence of states $(q_0, ..., q_n)$ such that $q_0 \in Q_0$ and $q_{i+1} \in \Delta(q_i, c_{i+1})$ for $0 \leq i \leq n-1$. A partial run $(q_0, ..., q_n)$ is a *full run* (or simply *run*) when we also have $q_n \in Q_F$.

We say that A accepts $w \in \Sigma^*$ from a state q, written $A(w, q) = 1$, when it has a full run starting with q. Otherwise, it is rejected and $A(w, q) = 0$. We define $A(w) := A(w, q_0)$ for q_0 the initial state of A and say that A *accepts* or *rejects* w. The language of A is defined by $\mathcal{L}(A) = \{w \mid A(w) = 1\}$.

Links between tile systems and automata have already been studied [Tho91] but stellar resolution provides a framework where it is especially natural to express various classes of automata by simulating a linear run in a graph. Notice that it is possible to encode directed graphs by translating edges (e, e') by binary stars $[-g(e), +g(e')]$. It is then possible to encode an NFA by first encoding its state graph then extending the rays so that the fusion triggers a flow of information. The final state will contain a dummy unpolarised ray `accept` so that the existence of a visible output in the normal form will

correspond to the acceptation a word. A similar approach has been studied with the model of flows [AB14, ABS16] where pointer machines are encoded.

§56.2 **Definition** (Encoding of words)**.** If $w = c_1...c_n$ is a word (*cf.* Appendix A.3) over some alphabet Σ then $w^\bigstar = [+i(c_1 \cdot ... \cdot c_n \cdot \varepsilon)]$ with a constant ε and the binary function symbol $\cdot$ which is considered right-associative, *i.e.* $a \cdot b \cdot c = a \cdot (b \cdot c)$.

§56.3 **Definition** (Encoding of non-deterministic automata)**.** Let $A = (\Sigma, Q, Q_0, \Delta, F)$ be an NFA. Its encoding is defined by $A^\bigstar$ such that:

- for each $q_0 \in Q_0$, we have $[-i(W), +a(W, q_0)]$;
- for each $q_f \in F$, we have $[-a(\varepsilon, q_f), \texttt{accept}]$;
- for each $q \in Q$ and if $c \in \Sigma_\varepsilon$ and for each $q' \in \Delta(q, c)$,
  - if $c = \varepsilon$, we have $[-a(W, q), +a(W, q')]$ (also known as *epsilon transition*);
  - otherwise, we have $[-a(W, q), +a(W, q')]$.

§56.4 **Lemma** (Simulation of transitions)**.** Let w be a word and A be an NFA. If $\Delta(c, q) = \{q_1, ..., q_n\}$ then $A^\bigstar \vdash [+a(c \cdot w, q)] \rightsquigarrow A^\bigstar \vdash \sum_{i=1}^n [+a(w, q_i)]$.

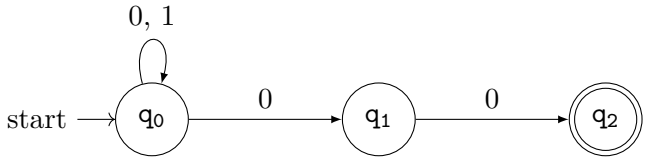
Proof. Assume that we have $\Delta(c, q) = \{q_1, ..., q_n\}$ and $A^\bigstar \vdash [+a(c \cdot w, q)]$. By the encoding of NFA, we must have stars $\sum_{i=1}^n [-a(c \cdot W, q), +a(W, q_i)]$ in $A^\bigstar$ representing the transitions $q_i \in \Delta(c, q)$. These are the only stars which are matchable with $[+a(c \cdot w, q)]$. By using a step of interactive execution, we obtain $A^\bigstar \vdash [+a(c \cdot w, q)] \rightsquigarrow A^\bigstar \vdash \sum_{i=1}^n [+a(w, q_i)]$ since $[-a(c \cdot W, q), +a(W, q_i)] \triangledown [+a(c \cdot w, q)] = [+a(w, q_i)]$. □

§56.5 **Theorem** (Simulation of non-deterministic finite automata)**.** Let w be a word and A be an NFA. We have:

1. if $A(w) = 1$, then $[\texttt{accept}] \in \lightning \texttt{IEx}(A^\bigstar, w^\bigstar)$;
2. if $A(w) = 0$, then $[\texttt{accept}] \notin \lightning \texttt{IEx}(A^\bigstar, w^\bigstar)$;

Proof. We show the two statements.

1. Assume $A(w) = 1$. It means that there is a run $\rho := (q_0, ..., q_n)$ from the initial state q_0 to $q_n \in F$. Since there is bijection between states of ρ and symbols of w, we have $w = w_0...w_n$. We have to compute $\lightning \texttt{IEx}(A^\bigstar, w^\bigstar)$. We start from $A^\bigstar \vdash [+i(w)]$. By interacting with the initial star $[-i(w), +a(w, q_0)]$, we obtain $A^\bigstar \vdash [+a(w, q_0)]$. By using n times Lemma 56.4, for each transition $q_{i+1} \in \Delta(w_i, q_i)$, interactive execution will produce $A^\bigstar \vdash [+a(w_{i+1} \cdot ... \cdot w_n \cdot \varepsilon, q_{i+1})]$. By using the last transition $q_n \in \Delta(c_{n-1}, q_{n-1})$, we obtain $A^\bigstar \vdash [+a(\varepsilon, q_n)]$ which can interact with the final star $[-a(\varepsilon, q_n), \texttt{accept}]$. We then have an occurrence of $[\texttt{accept}]$ on the right of $\vdash$ and other stars which did not reached the final state are erased by $\lightning$. The star



$$A^\star = [-i(W), +a(W, \mathtt{q_0})] + [-a(\epsilon, \mathtt{q_2}), \mathtt{accept}] + [-a(0 \cdot W, \mathtt{q_0}), +a(W, \mathtt{q_0})] +$$

$$[-a(1 \cdot W, \mathtt{q_0}), +a(W, \mathtt{q_0})] + [-a(0 \cdot W, \mathtt{q_0}), +a(W, \mathtt{q_1})] + [-a(0 \cdot W, \mathtt{q_1}), +a(W, \mathtt{q_2})]$$

Figure 56.1: An NFA accepting words finishing by 00 and its translation in stellar resolution. We have $\lightning\, \mathtt{IEx}(A^\star, [+i(0 \cdot 0 \cdot 0 \cdot \varepsilon)]) = [\mathtt{accept}]$.

[accept] cannot interact because its only ray is unpolarised. It follows that it must be part of the normal form and $[\mathtt{accept}] \in \lightning\, \mathtt{IEx}(A^\star, w^\star)$.

We have to check that cyclic diagrams do not cause problems. Transitions are encoded as binary stars, hence a cyclic diagram is necessarily closed and will reduce into a loop on a binary star. This kind of loop involves an equation of the form $c \cdot W \stackrel{?}{=} W$ (because of the linear consumption of the input), which is impossible to solve. Such diagrams will be excluded in the computation of the normal form.

2. Assume $A(w) = 0$. It means that all runs are partial runs $\rho := (q_0, ..., q_n)$ where $q_n \notin F$. Since there is bijection between states of ρ and symbols of w, we have $w = w_0 ... w_n$. By Lemma 56.4, these runs correspond to transitions $A^\star \vdash [+a(w, q_0)] \rightsquigarrow^* A^\star \vdash [+a(w, q_n)]$. Because of the operator $\lightning$, such polarised stars are erased. Hence, the normal form must be empty, *i.e.* $\lightning\, \mathtt{IEx}(A^\star, w^\star) = \emptyset$ and in particular $[\mathtt{accept}] \notin \lightning\, \mathtt{IEx}(A^\star, w^\star)$.

□

§56.6 **Remark** (Incomplete path)**.** Remark that in the case $A(w) = 0$ of Lemma 56.4, partial runs lead to polarised stars which cannot interact anymore (or diagrams with polarised free rays). We could keep those incomplete paths but we choose to make them invisible by using the operator $\lightning$.

§56.7 Different encodings of words are also possible. For instance, in Aubert and Bagnol's works [AB14, ABS16], characters are encoded with flows forming a cyclic chain of α-unifiable terms which interact with the encoding of the state graph of an automaton. This defines a representation of logspace computation where the input is explored with pointers.

§56.8 **Definition** (Logspace words [AB14])**.** Let $w = c_1 ..., c_n$ be a word together with distinct constants $\mathtt{p_0}, \mathtt{p_1}, ..., \mathtt{p_n}$ where $\mathtt{p_i}$ represents a position in w. The encoding of w is

defined by:

$$w^{\mathrm{LOG}\bigstar} := (\star \bullet \mathtt{r} \bullet X \bullet (\mathtt{p_0} \bullet Y) \leftrightharpoons c_1 \bullet \mathtt{l} \bullet X \bullet (\mathtt{p_1} \bullet Y))^{\bigstar}$$

$$+ \sum_{i=1}^{n-1} (c_i \bullet \mathtt{r} \bullet X \bullet (\mathtt{p_i} \bullet Y) \leftrightharpoons c_{i+1} \bullet \mathtt{l} \bullet X \bullet (\mathtt{p_{i+1}} \bullet Y))^{\bigstar}$$

$$+ (c_n \bullet \mathtt{r} \bullet X \bullet (\mathtt{p_n} \bullet Y) \leftrightharpoons \star \bullet \mathtt{l} \bullet X \bullet (\mathtt{p_0} \bullet Y))^{\bigstar}$$

where $\mathtt{l}, \mathtt{r}$ are constants and $\bullet$ is a binary symbol considered right-associative, *i.e.* $t \bullet u \bullet v := t \bullet (u \bullet v)$.

Non-deterministic pushdown automata (NPDA)

§56.9 A non-deterministic pushdown automaton is an NFA with a stack defined as a tuple $P = (Q, \Sigma, \Gamma, \Delta, Q_0, \$, F)$ where Q is the set of states, Σ the alphabet of the input, Γ the alpabet of the stack, $Q_0 \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, $\$ \in \Gamma$ is the initial stack symbol and $\Delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function.

§56.10 We say that P accepts $w = c_1 ... c_n \in \Sigma^*$, written $P(w) = 1$, when there exists $q_0, ..., q_n \in Q$ and $s_0, ..., s_n \in \Gamma^*$ such that $q_0 \in Q_0$, $s_0 = \varepsilon$, $q_n \in F$ and for $0 \leq i \leq n-1$, we have $(q_{i+1}, b) \in \Delta(q_i, c_{i+1}, a)$ where $s_i = a \cdot t$ and $s_{i+1} = b \cdot t$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$. In particular, if $a = \varepsilon$, then b is just added on the stack and if $b = \epsilon$ then we simply pop a. Otherwise, we replace the symbol a by b on the top of the stack. Otherwise, w is rejected by P, which is written $P(w) = 0$.

§56.11 **Definition** (Encoding of non-deterministic pushdown automata)**.** The encoding of a NPDA $P = (Q, \Sigma, \Gamma, \Delta, Q_0, \$, F)$ is defined by $P^{\bigstar}$ such that:

- for each $q_0 \in Q_0$, we have $[-i(W), +p(W, q_0, \$)]$;
- for each $q_f \in F$, we have $[-p(\varepsilon, q_f, S), \mathtt{accept}]$;
- for each $q \in Q, c \in \Sigma_\varepsilon$ and for each $(q', b) \in \Delta(q, c, a)$ such that $a, b \in \Gamma_\varepsilon$, we have:
  - $[-p(c \cdot W, q, S), +p(W, q', S)]$ when $a = \varepsilon$ and $b = \varepsilon$;
  - $[-p(c \cdot W, q, S), +p(W, q', b \cdot S)]$ when $a = \varepsilon$ and $b \neq \varepsilon$;
  - $[-p(c \cdot W, q, a \cdot S), +p(w, q', S)]$ when $a \neq \varepsilon$ and $b = \varepsilon$;
  - $[-p(c \cdot W, q, a \cdot S), +p(w, q', b \cdot S)]$ when $a \neq \varepsilon$ and $b \neq \varepsilon$.

  If $c = \varepsilon$, as for NFA, it disappears from the interpretation and we have simply W instead of $\varepsilon \cdot W$ in the above four cases.

§56.12 **Lemma** (Stack management)**.** Let P be a NPDA and w be a word. If $\Delta(q, c, a) = \{(q_1, b_1), ..., (q_n, b_n)\}$, then for any term s representing a stack:

1. $P^\star \vdash [+p(q, c \cdot w, s)] \rightsquigarrow P^\star \vdash \sum_{i=1}^n [+p(q_i, w, s)]$ when $a = \varepsilon$ and $b = \varepsilon$;
2. $P^\star \vdash [+p(q, c \cdot w, b \cdot s)] \rightsquigarrow P^\star \vdash \sum_{i=1}^n [+p(q_i, w, s)]$ when $a = \varepsilon$ and $b \neq \varepsilon$;
3. $P^\star \vdash [+p(q, c \cdot w, a \cdot s)] \rightsquigarrow P^\star \vdash \sum_{i=1}^n [+p(q_i, w, s)]$ when $a \neq \varepsilon$ and $b = \varepsilon$;
4. $P^\star \vdash [+p(q, c \cdot w, a \cdot s)] \rightsquigarrow P^\star \vdash \sum_{i=1}^n [+p(q_i, w, b \cdot s)]$ when $a \neq \varepsilon$ and $b \neq \varepsilon$.

Proof. The proof follows from matchability and case analysis of the encoding of NPDA. We only look at the last case and the other ones are similar. Assume $a \neq \varepsilon$ and $b \neq \varepsilon$. We have $P^\star \vdash [+p(q, c \cdot w, a \cdot s)]$. It can interact with stars $[-p(q, c \cdot W, a \cdot S), +p(q_i, W, b \cdot S)]$ encoding the transitions $(q_i, b_i) \in \Delta(q, c, a)$. By fusion, we have:

$$[+p(q, c \cdot w, a \cdot s)] \triangledown [-p(q, c \cdot W, a \cdot S), +p(q_i, W, b \cdot S)] = +p(q_i, w, b \cdot s).$$

Hence, we obtain $P^\star \vdash \sum_{i=1}^n [+p(q_i, w, b \cdot s)]$, as expected. □

§56.13 **Theorem** (Simulation of non-deterministic pushdown automata)**.** Let P be a NPDA and w be a word. We have:

1. if $P(w) = 1$, then $[\texttt{accept}] \in \lightning \, \texttt{IEx}(P^\star, w^\star)$;
2. if $P(w) = 0$, then $[\texttt{accept}] \notin \lightning \, \texttt{IEx}(P^\star, w^\star)$.

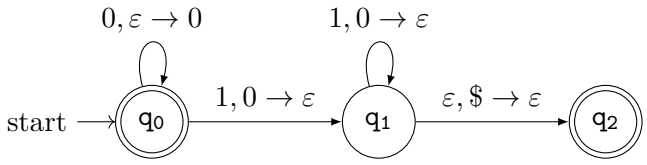
Proof. The proof follows exactly the same idea as in Theorem 56.5. NPDA are NFA extended with a stack, hence it only remains to show that the treatment of the stack is sound. This follows from Lemma 56.12. □

§56.14 An example of pushdown automata recognising the language $\{0^n 1^n \mid n \geq 0\}$ is illustrated in Figure 56.2.

Finite sequential transducers (NFST)

§56.15 Transducers are automata which can output symbols at each transition. When reaching a final state, a whole word is produced. The idea of the encoding is simple: we use a stack as in the encoding of NPDA and simply output this stack in the final state with an unpolarised variable replacing `accept`.

§56.16 A finite sequential transducer is defined as a tuple $T = (\Sigma, \Gamma, Q, Q_0, \Delta, F)$ corresponding to an NFA $(\Sigma, Q, Q_0, \Delta, F)$ extended with a language of output Γ and a transition function $\Delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$.



$$P^{\bigstar} = [-i(W), +p(W, \mathtt{q_0}, \$)] + [-p(\epsilon, \mathtt{q_3}, \$), \mathtt{accept}] +$$
$$[-p(0 \cdot W, \mathtt{q_0}, S), +p(W, \mathtt{q_0}, 0 \cdot S)] + [-p(1 \cdot W, \mathtt{q_0}, 0 \cdot S), +p(W, \mathtt{q_1}, S)] +$$
$$[-p(1 \cdot W, \mathtt{q_1}, 0 \cdot S), +p(W, \mathtt{q_1}, S)] + [-p(W, \mathtt{q_1}, \$), +p(W, \mathtt{q_2}, \$)]$$

Figure 56.2: A transition $a, b \to c$ means that we read a and replace b on the stack by c. The idea is to add 0 on the stack for each 0 read on the input then decrease by the same number for each 1 read. If we reach the initial stack symbol \$, then we necessarily have as many 0s as 1s.

§56.17 **Definition** (Encoding of non-deterministic finite sequential transducers)**.** Let $T = (\Sigma, \Gamma, Q, Q_0, \Delta, F)$ be an NFST. Its encoding is defined by $A^{\bigstar}$ such that:

- for each $q_0 \in Q_0$, we have $[-i(W), +f(W, q_0, \varepsilon)]$;
- for each $q_f \in F$, we have $[-f(\varepsilon, q_f, S), S]$;
- for each $q \in Q$ and if $c \in \Sigma_\varepsilon$ and $c' \in \Gamma_\varepsilon$ and for each $(q', c') \in \Delta(q, c)$, we have:

$$[-f(c \cdot W, q, S), +f(W, q', c' \cdot S)].$$

If $c = \varepsilon$, the transition is translated into $[-f(W, q, S), +f(W, q', c' \cdot S)]$. If $c' = \varepsilon$ then we have S instead of $\varepsilon \cdot S$.

§56.18 It is possible to extend even more the encoding of transducers, by using two stacks which explores the input by going to the left or to the right, thus obtaining 2-way transducers [BC18]. But instead of exploring this idea, we will use these two stacks for the interpretation of Turing machines in order to assert the Turing completeness of stellar resolution.

Non-deterministic Turing machines (NTM)

§56.19 Non-deterministic Turing machines (NTM) have been defined in Paragraph 16.5. For the encoding, we use the facts that Turing machines can be represented with two stacks in order to represent the left and right part of a tape. A move of the head will be represented as a manipulation of stack.

§56.20 We use terms $m(L, Q, X, R)$ where L and R are the left and right part of the tape relatively to the current position of the head. The variables Q and X respectively

represent the current state and symbol read by the head. We implicitly consider the symbol $\bullet$ as left-associative (hence $a \bullet b \bullet c = (a \bullet b) \bullet c$) and $\circ$ right-associative (hence $a \circ b \circ c = a \circ (b \circ c)$) so that it looks like we are traversing a tape.

§56.21 **Convention** (New encoding of words)**.** For technical reasons, Turing machines use the same encoding of words as before but with $\cdot$ replaced by $\circ$ (but the priority does not change). This is because at the beginning of the machine, the word is loaded on the right-hand tape.

§56.22 **Definition** (Encoding of non-deterministic Turing machines)**.** The encoding of an NTM $M = (Q, \Gamma, \Delta, q_0, q_a, q_r)$ is defined by a constellation $M^\bigstar$ such that:

- q_0 is translated into $[-i(C \circ W), +m(\sqcup, q_0, C, W)] + [-i(\sqcup), +m(\sqcup, q_0, \sqcup, \sqcup)]$;
- q_a is translated into $[-m(L, q_a, X, R), \texttt{accept}]$;
- q_r is translated into $[-m(L, q_r, X, R), \texttt{reject}]$;
- for each $q \in Q$ and $c \in \Gamma_\sqcup$ such that $(q', c', d) \in \Delta(q, c)$:
  - if $d = \texttt{l}$ (going left) then we have $[-m(L \bullet X, q, c, R), +m(L, q', X, c' \circ R)]$;
  - if $d = \texttt{r}$ (going right) then we have $[-m(L, q, c, X \circ R), +m(L \bullet c', q', X, R)]$;
  - if $d = \texttt{s}$ (staying still) then we have $[-m(L, q, c, R), +m(L, q', c', R)]$;
- we add two additional "memory allocation stars":

$$[-m(\sqcup, Q, C, R), +m(\sqcup \bullet \sqcup, Q, C, R)] + [-m(L, Q, C, \sqcup), +m(L, Q, C, \sqcup \circ \sqcup)].$$

§56.23 The two last stars are used to dynamically allocate space on the tape when necessary (similarly to `malloc()` in the C language). Instead of considering Turing machines as word acceptors, it is also possible to output the content of the tape and hence compute functions by translating q_a into $[-m(L, q_a, X, R), \texttt{accept}(L, X, R)]$.

§56.24 **Lemma** (Simulation of transitions)**.** Let M be an NTM and w be a word. If

$$(q', c', d) \in \Delta(q, c),$$

then:

- if $d = \texttt{l}$, then $M^\bigstar \vdash [-m(l \bullet a, q, c, r)] \rightsquigarrow M^\bigstar \vdash [+m(l, q', a, c' \circ r)]$;
- if $d = \texttt{r}$, then $M^\bigstar \vdash [-m(l, q, c, a \circ r)] \rightsquigarrow M^\bigstar \vdash [+m(l \bullet c', q', a, r)]$;
- if $d = \texttt{s}$, then $M^\bigstar \vdash [-m(l, q, c, r)] \rightsquigarrow M^\bigstar \vdash [+m(l, q', c', r)]$.

Proof. As for Theorem 56.13, this follows from a case analysis and from term unification. We only do the first case and the other cases are similar. Assume that we have $d = \texttt{l}$ and

$M^\star \vdash [-m(l \bullet a, q, c, r)]$. It can interact with $[-m(L \bullet X, q, c, R), +m(L, q', X, c' \circ R)]$. By fusion, we obtain $[+m(l, q', a, c' \circ r)]$. □

§56.25 **Theorem** (Simulation of non-deterministic Turing machines)**.** Let M be an NTM and w be a word. We have:

1. if $M(w) = 1$, then $M^\star \vdash w^\star \rightsquigarrow^* M^\star \vdash \Psi$ with $[\texttt{accept}] \in \Psi$;
2. if $M(w) = 0$, then $M^\star \vdash w^\star \rightsquigarrow^* M^\star \vdash \Psi$ with $[\texttt{reject}] \in \Psi$;
3. if $M(w) = \infty$ and $M^\star \vdash w^\star \rightsquigarrow^* M^\star \vdash \Psi$ then $[\texttt{accept}], [\texttt{reject}] \notin \Psi$.

Proof. The proof is similar to the proof of Theorem 56.5. We show the two statements.

1. Assume $M(w) = 1$. We must have a sequence of configurations $\rho := (C_0, ..., C_n)$ such that $C_n = (l, q_0, r)$ and $C_n = (l', q_a, r')$ for some l, l', r and r'. We start from $M^\star \vdash w^\star$. The star $w^\star$ can interact with the initial star and we obtain either $[+m(\sqcup, q_0, c, w)]$ for some word $c \circ w$ or $[+m(\sqcup, q_0, \sqcup, \sqcup)]$ depending on if $w \neq \varepsilon$ or $w = \varepsilon$. By Lemma 56.24, we can simulate the transitions of M. Each time there is a memory shortage, *i.e.* when we have $M^\star \vdash [+m(\sqcup, q', c', r')]$ and the next transition goes right, or $M^\star \vdash [+m(l', q', c', \sqcup)]$ and the next transition goes left, we need to allocate memory with the special memory allocation stars of the encoding. Without loss of generality, we only look at the case of right direction. We have $[+m(\sqcup, q', c', r')]$ which is matchable with $[-m(\sqcup, Q, C, R), +m(\sqcup \bullet \sqcup, Q, C, R)]$. By fusion, we obtain $[+m(\sqcup \bullet \sqcup, q', c', r')]$. We can then interact with the star encoding the next transition as in Lemma 56.24. By repeating this operation for all configurations of ρ, we will introduce duplications for non-deterministic choices. Eventually, since $M(w) = 1$, we should reach $[\texttt{accept}]$ which cannot interact, as in the proof of Theorem 56.5.
2. Assume $M(w) = 0$. This case is similar to the previous case since both are different class of halting states.
3. Assume $M(w) = \infty$. The reasoning is the same as for Theorem 56.5. All sequences of configuration do not reach neither q_a nor q_r. By simulating sequences of configurations with interactive execution, we can never reach $[\texttt{accept}]$ nor $[\texttt{reject}]$.

We have to check that cyclic diagrams cause no problems. Since Turing machines only use binary polarised stars, all cyclic diagrams must be closed. Consider such a cyclic diagram. If it is incorrect, then it does not appear in the normal form. In case it is correct, it represents a trivially infinite loop during the execution of the machine such as $[-m(L, q, c, R), +m(L, q, c, R)]$. This diagram has no effect on the presence or absence of $[\texttt{accept}]$ or $[\texttt{reject}]$ in the normal form. □

§56.26 **Remark** (Bounded memory allocation)**.** Remark that it is impossible to allocate too much memory because the allocation stars require that we have a tape equal to $\sqcup$.

> Otherwise, we would have infinitely many diagrams for all the possible amount of space allocation and no encoding of Turing machine would be strongly normalising.

§56.27 An example of Turing machine and its encoding is given in Figure 56.3. Although we have shown an example of *deterministic* Turing machine, it is easy to see how the non-deterministic case works. We can have several choices so that a same transition can match with several transitions. This will necessarily yield several diagrams corresponding to different runs. The whole machine accepts the input when at least one run accepts the word.

57 More advanced machines

Alternating Turing machines (ATM)

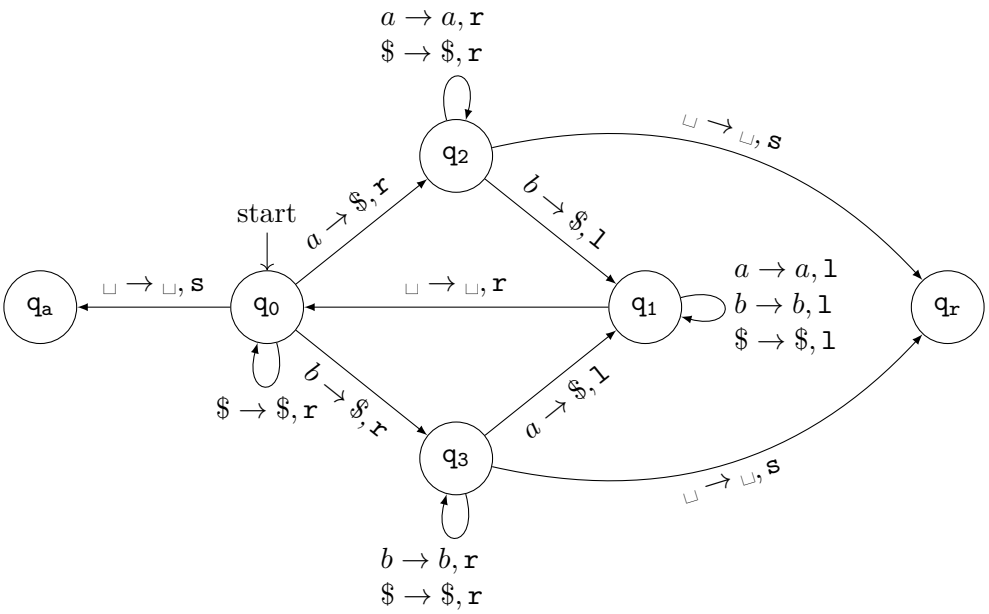
§57.1 Alternating Turing machines are Turing machines with two classes of states: existential ($\vee$) and universal ($\wedge$) states. We use a function $\texttt{class} : Q \to \{\vee, \wedge\}$ in order to associate a class with states.

§57.2 The acceptation is slightly modified: in order for an ATM M to accept a word w (with the usual notation $M(w) \in \{1, 0, \infty\}$), we require that in the computation tree of M, all children of $\wedge$ nodes are accepting paths and at least one of the children of $\vee$ paths is accepting. The idea is that universal states compute in parallel. More formally, by using the notation used for automata, $M(w, q) = 1$ when:

- if $\texttt{class}(q) = \vee$, then there exists a sequence of configurations $(C_1, ..., C_n)$ with $C_1 = (c, q, w)$ for some c which is accepting (as for NTM);
- if $\texttt{class}(q) = \wedge$, then *all* sequences of configurations $(C_1, ..., C_n)$ with $C_1 = (c, q, w)$ for some c are accepting.

We have $M(w) = 1$ when for all state q of M, we have $M(w, q) = 1$. Rejection is similar but with q_r instead of q_a. The case of NTM is a special case where all states are of type $\vee$ (because non-determinism only requires one path to succeed).

§57.3 In order to reproduce the computation of an ATM M on a word w, we need a trick for the treatment of $\wedge$ states. Instead of using the non-determinism of constellations which would induce several parallel diagrams, a $\wedge$ state $q \in Q$ of $n > 1$ outputs should be represented by a star of n outputs (polarity $+$) so that all paths from q are part of the same diagram. Since in our definition of NTM, all paths terminate either on `accept` or `reject`, it is sufficient to require that if $M(w) = 1$, then $\texttt{Ex}(M^\bigstar + w^\bigstar)$ contains at least one star $[\texttt{accept}, \overset{k}{...}, \texttt{accept}]$ for $k \geq 1$. Otherwise, if there is at least one `reject`, it means that some paths from a $\wedge$ state has been rejected or that no paths of $\vee$ states are accepting.



$$M^\bigstar = [-i(C \circ W), +m(\sqcup, q_0, C, W)] + [-i(\sqcup), +m(\sqcup, q_0, \sqcup, \sqcup)]+$$
$$[-m(L, q_0, \sqcup, R), +m(L, q_a, \sqcup, R)] + [-m(L, q_2, \sqcup, R), +m(L, q_r, \sqcup, R)]+$$
$$[-m(L, q_0, \$, C \circ R), +m(L \bullet \$, q_0, C, R)] + [-m(L, q_2, \$, C \circ R), +m(L \bullet \$, q_2, C, R)]+$$
$$[-m(L, q_0, a, C \circ R), +m(L \bullet \$, q_2, C, R)] + [-m(L, q_2, a, C \circ R), +m(L \bullet a, q_2, C, R)]+$$
$$[-m(L, q_0, b, C \circ R), +m(L \bullet \$, q_3, C, R)] + [-m(L \bullet C, q_2, b, R), +m(L, q_1, C, \$ \circ R)]+$$
$$[-m(L, q_1, \sqcup, C \circ R), +m(L \bullet \sqcup, q_0, C, R)] + [-m(L, q_3, \sqcup, R), +m(L, q_r, \sqcup, R)]+$$
$$[-m(L \bullet C, q_1, \$, R), +m(L, q_1, C, \$ \circ R)] + [-m(L, q_3, \$, C \circ R), +m(L \bullet \$, q_3, C, R)]+$$
$$[-m(L \bullet C, q_1, a, R), +m(L, q_1, C, a \circ R)] + [-m(L \bullet C, q_3, a, R), +m(L, q_1, C, \$ \circ R)]+$$
$$[-m(L \bullet C, q_1, b, R), +m(L, q_1, C, b \circ R)] + [-m(L, q_3, b, C \circ R), +m(L \bullet b, q_3, C, R)]+$$
$$[-m(L, q_a, X, R), \mathtt{accept}] + [-m(L, q_r, X, R), \mathtt{reject}]+$$
$$[-m(\sqcup, Q, C, R), +m(\sqcup \bullet \sqcup, Q, C, R)] + [-m(L, Q, C, \sqcup), +m(L, Q, C, \sqcup \circ \sqcup)]$$

Figure 56.3: A Turing machine accepting words containing as many symbols `a` as symbols `b` where $a \to b, d$ from a state q to q' corresponds to a transition $\Delta(q, a) = (q', b, d)$. When computing $\mathtt{IEx}(M^\bigstar, a^\bigstar)$, we plug the input with the correct initial star and obtain $[+m(\sqcup, \mathtt{q_0}, a, \sqcup)]$. No star can be connected, hence we have to connect to the right allocation star and obtain $[+m(\sqcup, \mathtt{q_0}, a, \sqcup \circ \sqcup)]$. We can use the star corresponding to $a \to \$, \mathtt{r}$ and obtain $[+m(\sqcup\bullet\$, \mathtt{q_2}, \sqcup, \sqcup)]$. Since we read $\sqcup$, we the use star corresponding to the transition $\sqcup \to \sqcup, \mathtt{s}$ and obtain $[+m(\sqcup \bullet \$, \mathtt{q_r}, \sqcup, \sqcup)]$. We can only use the star corresponding to q_r and obtain $[\mathtt{reject}]$. If we had a character b next to a, we would reach $[\mathtt{accept}]$.

§57.4 **Definition** (Encoding of alternating Turing machines)**.** The encoding of an ATM $M = (Q, \Gamma, \Delta, q_0, q_a, q_r, \texttt{class})$ is defined by a constellation $M^\bigstar$ such that:

- q_0, q_a and q_r are translated as in Definition 56.22.
- for each $q \in Q$ and $c \in \Gamma_\varepsilon$ such that $(q', c', d) \in \Delta(q, c)$:
  - if $\texttt{class}(q') = \vee$, we have the star given in Definition 56.22;
  - if $\texttt{class}(q') = \wedge$, then for q_i, c_i, d_i such that $(q_i, c_i, d_i) \in \Delta(q', c')$ for all c' with $1 \leq i \leq k$, we have:
    - $[-m(L \bullet X, q', c', R), +m(L, q_1, X, c_1 \circ R), \overset{k}{\ldots}, +m(L, q_k, X, c_k \circ R)]$ when $d_i = \texttt{1}$;
    - $[-m(L, q', c', X \circ R), +m(L \bullet c_1, q_1, X, R), \overset{k}{\ldots}, +m(L \bullet c_k, q_k, X, R)]$ when $d_i = \texttt{1}$.

  If $c = \varepsilon$ or $c' = \varepsilon$, then we do not write them in the stack as for Turing machines.
- we add two additional stars for memory allocation:

$$[-m(\sqcup, q, c, R), +m(\sqcup \bullet \sqcup, q, c, R)] + [-m(L, q, c, \sqcup), +m(L, q, c, \sqcup \circ \sqcup)].$$

§57.5 **Theorem** (Simulation of alternating Turing machines)**.** Let M be an ATM and w a word. For $k > 0$, we have:

1. if $M(w) = 1$, then $M^\bigstar \vdash w^\bigstar \rightsquigarrow^* M^\bigstar \vdash \Psi$ with $[\texttt{accept}, \overset{k}{\ldots}, \texttt{accept}] \in \Psi$;
2. if $M(w) = 0$, then $M^\bigstar \vdash w^\bigstar \rightsquigarrow^* M^\bigstar \vdash \Psi$ with $[\texttt{reject}, \overset{k}{\ldots}, \texttt{reject}] \in \Psi$.

Proof. The proof follows the same idea as Theorem 56.25 since ATM are extensions of NTM. It remains to show that the behaviour associated with the types of states is correctly simulated. The case of $\vee$ states is identical to the behaviour of regular states in NTM. We show the first statement for $M(w) = 1$ with $w = c \circ w'$ for some c and w' (the case with $w = \varepsilon$ is similar to the one with Turing machines in Theorem 56.25). The case of $M(w) = 0$ is similar since these are both halting cases. Assume $M(w) = 1$. We show that for any universal state, interactive execution produces a star $[\texttt{accept}, \overset{n}{\ldots}, \texttt{accept}]$ for some n.

We start from the interactive configuration $M^\bigstar \vdash w^\bigstar$. It will interact with the initial state and we obtain $M^\bigstar \vdash w^\bigstar \rightsquigarrow M^\bigstar \vdash [+m(\sqcup, q_0, c, w)]$. Assume we have a non-deterministic transition to states $q_1, ..., q_n, s_1, ..., s_m$ such that $\texttt{class}(q_i) = \vee$ and $\texttt{class}(s_i) = \wedge$.

- For all $(q_i, c_i, d_i) \in \Delta(q_0, c)$, a copy of $[+m(\sqcup, q_0, c, w)]$ interacts with the star associated with the transition, as in Theorem 56.25. We obtain $\sum_{i=1}^{n} [+m(l_i, q_i, c_i', r_i)]$ in the normal form with a choice of l_i, c_i' and r_i consistent with the translation of Turing machines into constellations.

- Without loss of generality, we consider the left case for directions $d_i = \mathtt{l}$. For all $(s_i, c_i, d_i) \in \Delta(q_0, c)$, a copy of $[+m(\sqcup, q_0, c, w)]$ interacts with the star $[-m(L \bullet X, q_0, c, R), +m(L, s_1, c_1, X \circ R), \overset{m}{\ldots}, +m(L, s_m, c_m, X \circ R)]$. We have

$$\phi := [+m(\sqcup, s_1, c_1, c \circ w), \overset{k}{\ldots}, +m(\sqcup, q_m, c_k, c \circ w)]$$

  in the normal form.

We finally obtain the stellar interaction:

$$M^\bigstar \vdash [+m(\sqcup, q_0, c, w)] \rightsquigarrow M^\bigstar \vdash \sum_{i=1}^{n} [+m(l_i, q_i, c'_i, r_i)] + \phi$$

We can apply more transitions (either to existential or universal states) by trigering interaction between a selected ray on the right of $\vdash$ and stars associated with next transitions. During interactive execution we have interaction spaces $M^\bigstar \vdash \sum_{i=1}^{n} \phi_i$ such that the ϕ_i are either stars $[+m(l', q', c', r')]$ for some l', q', c' and r' or stars

$$[+m(l'_1, q'_1, c'_1, r'_1), \overset{k}{\ldots}, +m(l'_n, q'_n, c'_n, r'_n)]$$

for some l'_i, q'_i, c'_i and r'_i. By the proof of Theorem 56.25, each time a state reach the accepting state q_a, its corresponding ray is replaced by the unpolarised ray `accept`. If we obtain a star $[\mathtt{accept}, \overset{n}{\ldots}, \mathtt{accept}]$ it means that all children of an universal state are accepting for one possible non-deterministic path. □

§57.6 Remark that ATM are, structurally speaking, not so far from regular constellations. In stellar resolution, we have non-deterministic choices with several rays matchable with one single ray and branching with multiple rays in a star matchable which rays of other stars. The first situation generalises $\vee$ states and the second one $\wedge$ states. This may be coherent with known results regarding the linear relation between the complexity of logic programs and ATM [Sha84].

Non-deterministic finite tree automata (NFTA)

§57.7 The previous automata we described recognise words. In this section, we consider automata which recognise trees [CDG+97]. By recognising trees, we can work on complex data types such as boolean formulas or arithmetic expressions for instance. In the literature, we usually have two types of NFTA: bottom-up NFTA which read the leaves and infer new states by going up and top-down NFTA which starts from the root and associate states with children. Since constellations deal with terms, we can naturally encode trees by using function symbols for nodes, which leads to an encoding of top-down NFTA.

$$[-ta(X, or(not(1), 1)), \texttt{accept}] + [+ta(q_0, 0)] + [+ta(q_1, 1)]+$$

$$[-ta(q_a, T_a), +ta(q_{\texttt{max}(a,b)}, or(T_a, T_b))] + [-ta(q_a, T_a), +ta(q_{1-a}, not(T_a))]$$

Figure 57.1: Constellation recognising trees correspond to true boolean formulas. In this example, it takes an instance of excluded middle as input.

§57.8 Formally, a top-down NFTA is a tuple $T = (Q, F, \texttt{ar}, Q_0, \Delta)$ where Q is the set of states, F is a set of function symbols, $\texttt{ar} : F \to \mathbf{N}$ associates an arity with function symbols, $Q_0 \subseteq Q$ is the set of initial states and Δ is a set of transition rules of the form $q(f(x_1, ..., x_n)) \to f(q_1(x_1), ..., q_n(x_n))$ for $f \in F$, $\texttt{ar}(f) = n$, $q, q_1, ..., q_n \in Q$ and $x_1, ..., x_n$ variables. An input tree t is accepted, written $T(t) = 1$, when it can be fully traversed by the transition rules by starting from a state $q \in Q_0$. Otherwise, it is rejected and we write $T(t) = 0$.

§57.9 **Definition** (Syntactic encoding of trees)**.** Let t be a tree. We define its encoding $t^\bigstar$ inductively by:

- X when t is a leaf x;
- $f(t_1^\bigstar, ..., t_n^\bigstar)$ when t is a node containing a symbol f with n children $t_1, ..., t_n$.

§57.10 **Definition** (Encoding of top-down tree automata)**.** Let $T = (Q, F, \texttt{ar}, Q_0, \Delta)$ be a top-down NFTA. The encoding of T is a constellation $T^\bigstar$ defined as follows:

- for each $q_0 \in Q_0$, we have $[-i(T), +ta(q_0, T)]$;
- for each $q(f(x_1, ..., x_n)) \to f(q_1(x_1), ..., q_n(x_n)) \in \Delta$, we have a constellation

  $$[-ta(q, f(X_1, ..., X_n)), +ta(q_1, X_1)] + ... + [-ta(q, f(X_1, ..., X_n)), +ta(q_n, X_n)];$$

  and for each leaf x, we add the following star: $[-ta(q, X), \texttt{accept}]$.

§57.11 **Theorem** (Simulation of top-down tree automata)**.** Let t be a tree and T be a top-down NFTA. We have $\lightning \texttt{Ex}(T^\bigstar + t^\bigstar) \neq \emptyset$ if and only if $T(w) = 1$.

Proof. The simulation follows the same idea as for automata (*cf.* Theorem 56.5). Instead of a consumption of word, we have a decomposition of tree which has the same role. □

§57.12 Interestingly, bottom-up NFTA are encoded in the same way but with reversed polarities: we start from leaves and go up to the root. Moreover, since we start with all the leaves, we need final states instead of initial ones. Instead of deconstructing trees, we are constructing trees. An example of what would be constellation corresponding

to a bottom-up tree automaton recognising true booleans formulas is illustrated in Figure 57.1. By using these ideas, it is also possible to extend the previous models to trees instead of words.

Krivine Abstract Machine (KAM) with call/cc

§57.13 The Krivine Abstract Machine (KAM) has already been defined in Section 22. I randomly had the idea to encode it into constellations but I am not very confident about the details. In this section, I only give the definitions without establishing any simulation result.

§57.14 The encoding of the λ_c-terms corresponds to an encoding of their syntax tree. We use the binding of variables already present in stellar resolution in order to create bound variables in λ-terms. We obtain a sort of λ-calculus with independent explicit substitutions [ACCL91]. The idea is that stars representing terms will be of shape $[-i(n_1, X_1), ..., -i(n_m, X_m), t]$ where t is the encoding of a λ_c-term, n_i are identifier of variables and X_i are variables of stellar resolution appearing in t.

§57.15 To make things simple, we consider that variables under some scope of a λ do not appear under another scope: there is no possible capture of variables. We can obtain this situation by simply renaming bound variables in a term. I tried to take scopes into account so that it would work for any λ-term but it was horrible to define and too technical to be understandable.

§57.16 **Definition** (Encoding of processes)**.** We write V_λ for the set of variables of λ_c-calculus and define a bijection $\sigma : V_\lambda \to \mathbf{N}$ between variables of λ_c-calculus and natural numbers and a bijection $\nu : V_\lambda \to V$ between variables of λ_c-calculus and variables of stellar resolution. First, we associate a term with λ_c-terms and stacks:

$$x^\bullet := \nu(x) \qquad (\lambda x.M)^\bullet := l(\overline{\sigma(x)}, M^\bullet) \qquad (MN)^\bullet := a(M^\bullet, N^\bullet) \qquad \mathtt{cc}^\bullet := \mathtt{cc}$$

$$\mathbf{k}_\pi^\bullet := s(\pi^\bullet) \qquad \varepsilon^\bullet := \varepsilon \qquad (M \cdot \pi)^\bullet := M^\bullet \cdot \pi^\bullet$$

The encoding of λ_c-terms and stacks is defined as follows with an index $\alpha \in \mathbf{N}$ associated with terms in order to construct their syntax tree:

$$x_\alpha^\star := [-i(\overline{\sigma(x)}, \nu(x)), +T_\alpha(x^\bullet)] \qquad (Y \neq \nu(x))$$

$$(\lambda x.M)_\alpha^\star := M_{\alpha\cdot 0}^\star + [-T_{\alpha\cdot 0}(X), +T_\alpha(l(\overline{\sigma(x)}, X))]$$

$$(MN)_\alpha^\star := M_{\alpha\cdot 0}^\star + N_{\alpha\cdot 1}^\star + [-T_{\alpha\cdot 0}(X), -T_{\alpha\cdot 1}(Y), +T_\alpha(a(X, Y))]$$

$$\mathtt{cc}_\alpha^\star := [+T_\alpha(\mathtt{cc}^\bullet)] \qquad (\mathbf{k}_\pi)_\alpha^\star := [+T_\alpha(\mathbf{k}_\pi^\bullet)] \qquad \pi^\star := [+S(\pi^\bullet)]$$

Finally, processes $M \star \pi$ are constructed with a star

$$(M \star \pi)^{\bigstar} := \mathtt{Ex}\big(M_{\varepsilon}^{\bigstar} + \pi^{\bigstar} + [-T_{\varepsilon}(X), -S(Y), +P(X \star Y)]\big)$$

connecting terms and stacks where $\star$ is a binary infix symbol and $\varepsilon \cdot \alpha = \alpha$ and ε is not written when alone. Since it will never be composed, we do not define priorities for $\star$.

§57.17 **Example.** We have $(\lambda x.x)^{\bigstar} = [-i(0,X), +T_0(X)] + [-T_0(x), +T(l(0,x))]$ which normalises into $[-i(0,X), +T(l(0,X))]$. Notice that the ray $-i(0,X)$ acts as an explicit substitution bound to X. When providing $[+i(0,u)]$ for some term u, the two stars merge and we obtain $[+T(l(0,u)]$ which indeed corresponds to a substitution. The process of β-reduction can be simulated with a star

$$[-T_{\alpha}(a(l(n,X),u)), +T_{\alpha}(X)] + [+i(n,u)]$$

which destructs a pattern of redex and produce a new term with the body of the function in which all occurrences of variables of identifier n are replaced by u thanks to the substitution star $[+i(n,u)]$.

§57.18 **Lemma.** Let t be a λ_c-term. We have:

$$\mathtt{Ex}(M_{\alpha}^{\bigstar}) = [-i(\overline{\sigma(x_1)}, \nu(x_1)), ..., -i(\overline{\sigma(x_n)}, \nu(x_n)), +T_{\alpha}(M^{\bullet})]$$

for $x_1, ..., x_n$ the variables in M.

Proof. The translation of λ_c-term is designed so to gives the encoding of their syntax tree. The only remaining free rays are the rays $-i(n,x)$ (for each variables appearing in the term) which corresponds to explicit substitutions $[x := \cdot]$ waiting for a value, and a ray $+T(M^{\bullet})$ where $M^{\bullet}$ is a representation of the λ_c-term being constructed. □

§57.19 **Definition** (Encoding of Krivine Abstract Machines)**.** The KAM is encoded as the constellation $K^{\bigstar}$ made of the following stars:

$$\begin{array}{ll}
[-P(a(M,N) \star \pi), +P(M \star N \cdot \pi)] & \text{(Push)} \\
[-P(l(X,M) \star N \cdot \pi), +P(M \star \pi), +i(X,N)] & \text{(Grab)} \\
[-P(\mathtt{cc} \star M \cdot \pi), +P(M \star \mathbf{k}_{\pi} \cdot \pi)] & \text{(Save)} \\
[-P(\mathbf{k}_{\pi} \star M \cdot \pi'), +P(M \star \pi)] & \text{(Restore)}
\end{array}$$

§57.20 **Example.** We would to evaluate $(\lambda xy.x)z$. We start from the construction of the λ_c-term given by Lemma 57.18. It produces the star:

$$[-i(0,X), -i(1,Y), -i(2,Z), +T(a(l(0,l(1,X)),Z))].$$

When put together with an empty stack to form a process and we obtain:

$$[-i(0,X),-i(1,Y),-i(2,Z),+P(a(l(0,l(1,X)),Z)\star\varepsilon)].$$

By fusion with the push star then the grab star, we obtain:

$$\begin{aligned}&\overset{push}{\rightsquigarrow} && [-i(0,X),-i(1,Y),-i(2,Z),+P(l(0,l(1,X))\star Z\cdot\varepsilon)]\\ &&\overset{grab}{\rightsquigarrow} && [-i(1,Y),-i(2,Z),+P(l(1,Z)\star\varepsilon)]\\ &&&= [-i(1,Y),-i(2,Z),+P((\lambda y.z)^\bullet\star\varepsilon)]\end{aligned}$$

58 Generalised circuits

§58.1 Boolean circuits have been introduced in Section 18. However, in this section, we do not directly encode boolean circuits but suggest a notion of *generalised circuits*. These circuits are acyclic and connected hypergraph structures in which information flows from inputs to a unique output and in which hyperedges are able to perform function calls from a semantics called *module*. This subsumes boolean circuits, arithmetic circuits but also opens the model of atypical circuits (or other variants with several outputs or other weird models we can imagine).

§58.2 **Definition** (Module)**.** A *module* is a tuple $(X, L, \mathtt{ar}, [\![\cdot]\!])$ where X is a set of *values*, L is a set of labels, $\mathtt{ar} : L \to \mathbf{N}\times\mathbf{N}$ a function associating an input-output arity to labels and $[\![\cdot]\!]$ associates with any label $l \in L$ a computable total function $X^n \to X^m$ such that $\mathtt{ar}(l) = (n, m)$.

§58.3 **Example.** Boolean circuits are constructed with the module $(X, L, \mathtt{ar}, [\![\cdot]\!])$ with:

- $X := \{0, 1\}$ and $V := \{x_i, i \in \mathbf{N}\}$;
- $L := \{0, 1, x_i \in V, \mathtt{s}, \neg, \wedge, \vee, \mathtt{c}\}$;
- $\mathtt{ar}(b) = (0, 1)$ for $b \in \{0, 1\}$ and $[\![b]\!] = b$;
- $\mathtt{ar}(\mathtt{s}) = (1, 2)$ and $[\![\mathtt{s}]\!](x) = (x, x)$;
- $\mathtt{ar}(\neg) = (1, 1)$ and $[\![\neg]\!](x) = 1 - x$;
- $\mathtt{ar}(\wedge) = (2, 1)$ and $[\![\wedge]\!](x, y) = \mathtt{min}(x, y)$;
- $\mathtt{ar}(\vee) = (2, 1)$ and $[\![\vee]\!](x, y) = \mathtt{max}(x, y)$;
- $\mathtt{ar}(\mathtt{c}) = (1, 0)$ and $[\![\mathtt{c}]\!](x)$ is undefined.

§58.4 **Definition** (Generalised circuit)**.** A *generalised circuit* is a tuple (C, M, κ) of:

- an acyclic and connected hypergraph $C = (V, E, \texttt{in}, \texttt{out})$ where elements of V are called *ports* and elements of E are called *gates*;
- a module $M = (X, L, \texttt{ar}, [\![\cdot]\!])$;
- the total function $\kappa : E \to L$ associates a label with each gate $e \in E$. We require that $|\texttt{in}(e)| = n$ and $|\texttt{out}(e)| = m$ when $\texttt{ar}(\kappa(e)) = (n, m)$.

There exists some special gates:

- *output* gates e such that $\texttt{out}(e) = \emptyset$;
- *input* gates e such that $\texttt{in}(e) = \emptyset$.

We require that a generalised circuit has at least one input gate and a *unique* output gate.

§58.5 **Definition** (Evaluation of generalised circuits)**.** Let (C, M, κ) be a generalised circuit with $C = (V, E, \texttt{in}, \texttt{out})$ and $M = (X, L, \texttt{ar}, [\![\cdot]\!])$. Its evaluation is given by a function $\texttt{val}$ such that $\texttt{val}(C) \in X$. It is defined as follows, where o is the only output gate of C:

- $\texttt{val}(C) := \texttt{val}(o)$;
- if $e \in E$ is an input gate then $\texttt{val}(e) = [\![e]\!]$;
- for all e such that $\texttt{in}(e) = \{i_1, ..., i_n\}$ with parents $\{e_k \in E \mid i_k \in \texttt{out}(e_k)\}$, we have $\texttt{val}(e) = [\![\kappa(e)]\!](\texttt{val}(e_1), ..., \texttt{val}(e_n))$.

§58.6 **Example.** A boolean circuit is any generalised circuit using constructors of Example 58.3. It is easy to extend the interpretation to arithmetic circuits by changing the module and replacing boolean functions by computable arithmetic functions. Conjunction is replaced by multiplication and disjunction by addition. Theoretically, we are not limited to computable functions but using computable functions guarantees that the circuit can be realised in stellar resolution.

§58.7 **Imperative programs.** If we allow loops and define ways to handle them, it might be possible to also represent imperative programs by considering the structure of the execution flow graph of a program and then passing states containing a memory (representing as a matrix over $\{0, 1\}$ for instance). We are also able to do function calls since functions are attached to constructors. I do not investigate this idea here.

§58.8 **Definition** (Encoding of generalised circuits and modules)**.** Let (C, M, κ) be a generalised circuit with $C = (V, E, \texttt{in}, \texttt{out})$ and $M = (X, L, \texttt{ar}, [\![\cdot]\!])$. We translate C into a constellation $C^\bigstar := \sum_{e \in E} e^\bigstar$. Assume that $\texttt{in}(e) = \{i_1, ..., i_n\}$ and $\texttt{out}(e) =$

$\{o_1, ..., o_m\}$. The translation of e is defined by a star

$$e^\bigstar := \begin{bmatrix} -i_1(X_1), & ..., & -i_n(X_n), \\ & -\kappa(e)(X_1, ..., X_n, Y_1, ..., Y_m), & \\ +o_1(Y_1), & ..., & +o_m(Y_m) \end{bmatrix}.$$

If e is a conclusion gate, then we add unpolarised rays $Y_1, ..., Y_m$ to output the result and remove the function call $-\kappa(e)(X_1, ..., X_n, Y_1, ..., Y_m)$.

For all label $l \in L$ such that $\mathtt{ar}(l) = (n, m)$, we define a star $l^\bigstar$ by any star such that $\mathtt{IEx}(l^\bigstar, [-l(a_1, ..., a_n, Y_1, ..., Y_m), Y_1, ..., Y_m]) = [b_1, ..., b_m]$ and $[\![l]\!](a_1, ..., a_n) = (b_1, ..., b_m)$.

The module M is translated into the constellation $M^\bigstar := \sum_{l \in L} l^\bigstar$.

§58.9 **Example.** It is possible to design a circuit for excluded middle with 1 as input:

$$[-1(X), +c_0(X)] + [-c_0(X), -\mathtt{s}(X, Y, Z), +c_1(Y), +c_2(Z)]$$

$$+[-c_1(X), -\neg(X, R), +c_3(R)] + [-c_2(X), -c_3(Y), -\wedge(X, Y, R), +c_4(R)] + [-c_4(R), R].$$

We need the constellation module:

$$[+1(1)] + [+\mathtt{s}(X, X, X)] + [+\neg(1, 0)] + [+\neg(0, 1)]$$

$$+[+\wedge(1, X, X)] + [+\wedge(0, X, 0)]$$

Remark that unused parts of the module can remain in the normal form except if we use the operator $\lightning$, as for automata.

§58.10 Notice that generalised circuits are purely structural waiting for some "function definitions". We have to plug its constellation with a module giving the semantics of gates. It shows that stellar resolution computes by internalising external procedures. In some sense, the syntax interacts with the semantics by expliciting hidden procedures.

§58.11 **Variables.** It is possible to hijack the encoding of generalised circuits in stellar resolution in order to express satisfiability of boolean expressions. If we do not use constants, then some variables will wait for a value. For any input ray $-c_x(X)$ which is free in the dependency graph of the constellation, we can add a star $\phi_x := [-i(X) + c_x(X)]$ for variables. We can add stars $\sum_{b \in X}[+i(b)]$ so that ϕ_x will face a non-deterministic choice for all values The execution of the whole constellation will produce several stars for each combinations of inputs. It is also possible to add a variant of conclusion star $[-c_x(1), ok]$ with an explicit constant in order to ask for satisfiability of the circuit. If the output is 1 then we obtain the unpolarised star $[ok]$. By using such constellations it is possible to express the NP-complete satisfiability problem.

§58.12 **Example.** It is possible to design a circuit for excluded middle with unknown boolean inputs:

$$[-i(X), +c_0(X)] + [-c_0(X), -\mathtt{s}(X,Y,Z), +c_1(Y), +c_2(Z)]$$

$$+[-c_1(X), -\neg(X,R), +c_3(R)]+[-c_2(X), -c_3(Y), -\wedge(X,Y,R), +c_4(R)]+[-c_4(R), R].$$

We then add $[+i(1)] + [+i(0)]$ in the module and these two stars can be linked with $[-i(X), +c_0(X)]$. We will then obtain one result for all possible combinations of value. The presence of one star $[ok]$ indicates that the corresponding formula is satisfiable. For n inputs, the presence of 2^n occurrences of $[ok]$ indicates that we have a tautology.

§58.13 When considering such circuits with input variables, we can plug value stars as in the previous example to fill values. We then obtain several circuits without input variables. It is then sufficient to consider generalised circuits without input variables.

§58.14 **Theorem** (Simulation of generalised circuits)**.** Let (C, M, κ) be a generalised circuit without input variables. We have $\lightning \mathtt{AEx}(C^\bigstar \uplus M^\bigstar) = [\mathtt{val}(C)]$.

Proof. Assume $C := (V, E, \mathtt{in}, \mathtt{out})$ and that we have a hyperedge e which is incident to another hyperedge e' through a vertex v. This situation is translated by stars $\phi \cup \{+c_x(X_1, ..., X_n)\}$ and $\phi' \cup \{-c_y(Y_1, ..., Y_n)\}$ for variables X_i and Y_i. By applying fusion, we have $\theta\phi \cup \theta\phi'$ with θ the solution of the equation induced by the rays we connected. We can apply execution on all stars of $C^\bigstar$ first (this is later justified with the partial pre-execution lemma of Lemma 64.9) and we obtain a star $[-f_1(...), ..., -f_k(...), r]$ where f_i corresponds to unsolved function call waiting for an interaction with $M^\bigstar$, and r is the only unpolarised ray corresponding to the only conclusion of C. Now, if implementations of functions in $M^\bigstar$ are indeed correct (consistently with Definition 58.8), then when the remaining star interacts with $M^\bigstar$, we must obtain $[\varphi(C)]$. All unused stars of $M^\bigstar$ are erased by the operator $\lightning$. □

59 Tile systems with the abstract tile assembly model

§59.1 In this section, I choose to only encode the abstract tile assembly model (aTAM) mentioned and defined in Paragraph 17.7. Other tile systems can easily be encoded. For instance, Wang tiles' encoding can be directly inferred from the encoding of aTAM and flexible tiles correspond to stars with polarised ray of nullary colour (hence a polarised constant).

§59.2 We suggest an encoding of the aTAM in $\mathbf{N}^2$ instead of $\mathbf{Z}^2$ which is more natural but not less powerful since it is known that $\mathbf{N}^2 \simeq \mathbf{Z}^2$ and also because we are able to compute any computable function (since we previously encoded Turing machines).

§59.3 **Definition** (Encoding of tiles)**.** Tile types $t_i = (g^i_{\mathtt{w}}, g^i_{\mathtt{e}}, g^i_{\mathtt{s}}, g^i_{\mathtt{n}})$ are encoded by a star $t_i^{\bigstar}$:

$$\begin{bmatrix} -\overset{\bullet}{h}(\mathtt{gl}(g^i_{\mathtt{w}})(X), X, Y), & -\overset{\bullet}{v}(\mathtt{gl}(g^i_{\mathtt{s}})(Y), X, Y), \\ +\overset{\circ}{h}(\mathtt{gl}(g^i_{\mathtt{e}})(s(X)), s(X), Y), & +\overset{\circ}{v}(\mathtt{gl}(g^i_{\mathtt{n}})(s(U)), X, s(Y)) \end{bmatrix}$$

where $gl(g)(X) := g(X) \cdot \overline{str(g)}$ for $str(g) \in \mathbf{N}$.

We define the translation of a set of tile types T as the constellation $T^{\bigstar} := \sum_{t_i \in T} t_i^{\bigstar}$.

§59.4 The symbols h (horizontal) and v (vertical) represent axis of connexion. The key point of the encoding is that because of the dots $\bullet$ and $\circ$, tiles have to use an intermediate star to check whether a connexion is allowed.

§59.5 **Definition** (Environment constellation)**.** The *environment constellation* for a temperature $\tau \in \mathbf{N}\backslash\{0\}$ is defined by $\Phi^{\tau}_{env} :=$

$$[+temp(\overline{\tau})] + \begin{bmatrix} +\overset{\bullet}{v}(g_1(X_1) \cdot N_1, X_1, Y_1), & -\overset{\circ}{v}(g_2(X_3) \cdot N_2, X_3, Y_3), \\ +\overset{\bullet}{h}(g_3(X_5) \cdot N_3, X_5, Y_5), & -\overset{\circ}{h}(g_4(X_7) \cdot N_4, X_7, Y_7), \\ -\overset{\circ}{v}(g_1(X_2) \cdot N_1, X_2, Y_2), & +\overset{\bullet}{v}(g_2(X_4) \cdot N_2, X_4, Y_4), \\ -\overset{\circ}{h}(g_3(X_6) \cdot N_3, X_6, Y_6), & +\overset{\bullet}{h}(g_4(X_8) \cdot N_4, X_8, Y_8), \\ -add(N_1, N_2, R_1), & -add(N_3, N_4, R_2), \\ \multicolumn{2}{c}{-add(R_1, R_2, R), -geq(R, T, \overline{1}), -temp(T)} \end{bmatrix}$$

$$+[-\overset{\bullet}{v}(g(X)\cdot\overline{0}, X, Y)] + [+\overset{\circ}{v}(g(X)\cdot\overline{0}, X, Y)] + [-\overset{\bullet}{h}(g(X)\cdot\overline{0}, X, Y)] + [+\overset{\circ}{h}(g(X)\cdot\overline{0}, X, Y)]$$

$$+[+\overset{\circ}{v}(g(X)\cdot\overline{0}, X, Y)] + [-\overset{\bullet}{v}(g(X)\cdot\overline{0}, X, Y)] + [+\overset{\circ}{h}(g(X)\cdot\overline{0}, X, Y)] + [-\overset{\bullet}{h}(g(X)\cdot\overline{0}, X, Y)]$$

$$+[+geq(\overline{0}, \overline{0}, \overline{1})] + [+geq(s(X), s(Y), R), -geq(X, Y, R)] + [+geq(s(X), \overline{0}, \overline{0})] +$$

$$[+geq(\overline{0}, s(Y), \overline{0})] + [+add(\overline{0}, Y, Y)] + [-add(X, Y, Z), +add(s(X), Y, s(Z))]$$

§59.6 **Theorem** (Simulation of the aTAM)**.** Let $\mathcal{T} = (T, \tau)$ be a TAS. We have

$$\mathtt{CSatDiags}(T^{\bigstar} + \Phi^{\tau}_{env}) \simeq \mathcal{A}_{\square}[\mathcal{T}].$$

Proof. It is sufficient to show that the computation of $\mathtt{CSatDiags}(T^{\bigstar} + \Phi^{\tau}_{env})$ behaves like the construction of tilings in aTAM.

Direct connexions between tiles without using Φ^{τ}_{env} is forbidden because of the symbols $\circ$ and $\bullet$. Notice that the colours v and h force the connexions to be on the same axis in order to follow the geometric restriction of tiling in a plane. The tiles are designed so that a plugging increment a coordinate X or Y depending on the position/axis of the side. The purpose of this feature is to simulate a shifting of tile on a plane so that two tiles cannot connect on two sides at the same time.

Because of the symbols $\circ$ and $\bullet$, we have to use the constellation Φ^{τ}_{env} as an intermediate for the connexion of two tile sides. We consider a tile $t_i \in \texttt{dom}(\alpha)$. We start with $t_i^{\bigstar}$. Assume t_i can be connected to k other tiles in $\texttt{dom}(\alpha)$. They can only be connected through Φ^{τ}_{env} by their connectable sides. Their glue type and strength for the connected sides have to match because of the shared variables for opposite sides in Φ^{τ}_{env}. All other unused sides of the connector star will be plugged by the unary stars used as fillers. By using principles of logic programming, the diagram can only be correct and saturated if the sum of connected sides of t_i is greater or equal to τ (note that the filled unused sides add 0 to the sum). The stars sing symbols *add* and *geq* are common logic programs, hence their correctness is assumed.

Since all $t_i \in \texttt{dom}(\alpha)$ satisfy the above property, the two operations have the same dynamics. Moreover, each tile corresponds exactly to a star and each of its sides corresponds to a ray and we have a structural isomorphism between tiles and their translation. It follows that we have a bijection between the set of non-empty finite assemblies constructible from T at temperature τ and $\texttt{CSatDiags}(T^{\bigstar} + \Phi^{\tau}_{env})$. □

§59.7 **Encoding of Wang tiles.** Wang tiles can be recovered by considering the case of non-cooperative tiles with temperature $\tau = 1$ [MW17], considering all strengths to be 0 and removing the environment constellation. Glue types then correspond to colours of Wang tiles. This works because like Wang tiles, tiles of aTAM are also bricks placed on a plane.

60 Discussion: a common language for classical computation?

§60.1 In this chapter, I presented the encoding of a lot of different models of computation. However, when I talk about it to other people, they often say "but your model is Turing-complete so it's not very surprising. Any Turing-complete model can do the same". But do all encodings have the same value? Intuition tells us that no: encoding λ-calculus into proof-nets seems more interesting that encoding λ-calculus with Turing machines. But why? I am not sure about that... How to differentiate an encoding which *reveals more* about the computational mechanisms of an object and another one which is only superfluous bureaucracy? Even though I do not have answer, I would like to write about several remarkable points appearing in the encodings I defined. I believe that it can potentially raise interesting questions about classical computation.

§60.2 **What models of computation share.** As discussed in Section 19, it seems to me that the main models of computation are all about basic information flowing inside a discrete structure (typically a graph or a hypergraph). This is the elementary mechanism that stellar resolution implements. Constellations are about basic information (expressed with terms) moving between points in a network.

- With Horn clauses, a path of inference trying to justify a query then several statement traverse the knowledge base of a logic program.
- In automata theory, we have information about states transferred from one state to another. Typical data exchanged are the different stacks of information such as information about how many characters are remaining in the input or information needed to start the next transition.
- In boolean and arithmetic circuits, data such as numbers or booleans moves from inputs to the unique output. During the traversal of the circuit, these data are altered by some functions (*e.g.* boolean gates).
- In tile systems it is more subtle but geometric information about position and the structure of space (coordinates in a plane) are exchanged between tiles.
- Finally, with the GoI (*cf.* Chapter 5), we have seen with the token machine and the long trip criterion that we could travel inside a proof-structure by following some rules in order to express cut-elimination and logical correctness.

This idea of basic information flowing inside a structure is very primitive. It seems to generalise most classical models of computation without serious rise of complexity or unnatural hacks. A similar idea has been developed with *(labelled) transition systems* used in model checking [BK08, Chapter 2]. Compared to other similar models, stellar resolution has the advantage of being self-sufficient, without the need for external procedures. It only uses term unification which is triggered when making two stars interact.

§60.3 **Automata as constellations with stacks.** I remarked a small but funny thing about the encoding of automata in stellar resolution. Notice that our encoding of words is simply a stack of characters and that consuming a character (by using a star $[-a(c \cdot w), +a(w)]$) is the dual operation of adding a character (by using a star $[-a(w), +a(c \cdot w)]$). In stellar resolution, these operations only differ by the polarity of terms and the direction of computation (given by the initial and final state). We obtain three types of stacks: *strictly decreasing* (SD), *strictly increasing* (SI) and *arbitrary* (A) which can be both increasing or decreasing. The difference between NFA and NPDA are the number of stacks they use and how these stack are used: SD-stack for NFA and both an SD-stack and an A-stack for NPDA.

§60.4 Using an additional SI-stack and replacing `accept` by the content of this new stack can actually naturally lead to transducers which have the additional feature of writing symbols. By assembling stacks of different types of stacks over the encoding of a graph, we obtain the most common state machines (Figure 60.1). It seems to me that it is also possible to manipulate collapsible pushdown automata [HMOS08] which manipulate higher order stacks (stacks of stacks). Since stacks in stellar resolution are terms (made by composition function symbols to stack symbols), they can be manipulated as any other terms.

Automata	**Stacks to add**		
	Input	**Output**	**Auxiliary stack**
NFA	SD		
NPDA	SD		A
NFST		SI	
2-way NFST	$2 \times$ A	SI	
NTM	$2 \times$ A		
n-tape NTM	$2 \times$ A		$2(n-1) \times$ A
NTM with output	$2 \times$ A		

Figure 60.1: Characterisation of automata by number and types of stacks.

§60.5 We have a very general characterisation of the different sorts of automata in the stellar resolution. However, in order to consider more complex extensions of automata such as alternating Turing machines, tree automata or other features, adding stacks is not sufficient any more: we need more sophisticated changes.

§60.6 **Seiller's graphings as another computational basis.** Seiller has similar ideas with his graphings which are able to represent automata by actions on state spaces [Sei18]. The idea seems so general that Seiller uses graphings to represent models of computation in a more mathematical and generic way. From this idea, he hopes to give a mathematical definition to the notion of *algorithm* which would correspond to a specification for graphings. However, the notion is still unclear and no works in that direction have been published yet, at the time of this thesis.

§60.7 **Are models of computation logical.** The encodings presented in this chapter are formatting of constellations. We are constraining constellations by giving some paths that execution must take. The shapes of terms also limit the potential of execution. To take an elementary example, what would we do to characterise constellations representing finite automata? Automata must contain the star $[+i(W), +a(W, q_0)]$ (exactly once), the star $[-a(\varepsilon), \texttt{accept}]$ (exactly once) then only stars of the shape $[-a(c \cdot W, q), +a(W, q')]$ representing transitions with some symbol c and states q, q'. We then require that the corresponding dependency graph is connected. Given a constellation Φ in the wild, how do we determine if it is an automata? Probably by using *tests* (*cf.* Chapter 6), as with correctness criteria for linear logic (although no tests for automata have been defined in this thesis). I believe that this makes *all* models of computation as "logical" as mathematical proofs. In particular, it is not so surprising that computer science and proof theory converged in the CHL correspondence Section 21 since they share similar practices of mixing the computational (object) and the logical (subject).

§60.8 **Regulation of computation.** I claim that models of computation are logical because their shape are characterised by some chosen tests. However, this is not even sufficient. A "human intervention" seems to be needed in a lot of models of computation. Circuits, for instance, usually compute from inputs to the output. However, constellations seem

to operate an asynchronous and parallel computation. There is no *explicit direction of computation* nor initial point (inputs). If we are not interested in implementations (for instance by using abstract execution), then there is no impact on the result. A solution that I implemented in this chapter is to use interactive execution by only selecting stars representing initial states on the right of $\vdash$ in interaction spaces. This is sufficient for automata but not for circuits. Imagine that we are starting from the conclusion instead of inputs in a circuit. When reaching the term associated with a function call, for instance $- \vee (X, Y, R)$, we need to non-deterministically test all choices of rays $+ \vee (..., ..., ...)$. However, if we start from the inputs, it is possible to fix some values and limit the possible choices (erasure of non-determinism). Actually, if we would like to perfectly reproduce the computation of circuits, we need *synchronisation* and a computation layer by layer. All this shows that *strategies* must be considered for real-world computation (as in λ-calculus in which a lot of strategies are studied). I now claim that:

$$\text{Model of computation} = \text{Stellar resolution} + \text{Logic} + \text{Strategy}$$
$$\text{Computational object} = \text{Constellation} + \text{Correctness tests} + \text{Regulated computation}$$

A lot of models of computation already assume a strategy which affects the complexity of their operations. A possible solution which I did not explored is to use internal polarities to simulate synchronisation of execution.

§60.9 **Anarchy and authority.** We have seen that an external control is needed so that we faithfully reproduce the behaviour of existing models of computation. However, this is not the case of all models. This corresponds to Girard's apodictic and epidictic described in Section 44.

- Apodictic models of computation are typically tile systems which are sometimes qualified as *natural computation*. Computation does not need any external control or human intervention by choosing where to start and how to proceed. Interestingly, the aTAM (*cf.* Section 59) features self-cooperation without the need of external control. We have an "intermediate-free" or "permissionless"[1] computation[2]. Computation is autonomous in these models and freed from any query to external procedures, structures of systems.
- Epidictic models contain circuits or logic programs in which computation has to be done in a certain way. We have to "educate" computation. However, what is the nature of those strategies? Can strategies always be implemented by terms or constellations so that we internalise this seemingly external control (as for aTAM)? For instance, those strategies would be checked by tests and we would have that "strategies=logic". It seems possible in some cases by finding alternative and more sophisticated encodings but I do not believe it is always possible. For automata

[1] Similarly to decentralised blockchain technologies, whereas banking systems are centralised with intermediates. Both have their advantages and drawbacks.

[2] Which can be considered a requirement for Girard's ideal of analytic space (*cf.* Section 42).

with interactive execution for instance, the arbitrary choice of initial states (although it may be simulated by using internal polarities).

I currently have no idea about how to implement strategies in a satisfying and generic way but I believe this is a serious and interesting problem which is in consistent with the idea that TS reveals implicit operations. Moreover, these considerations has an effect on computational complexity.

Chapter 9

Properties of execution for objective constellations

In this section, we present few things which can be said about constellations, their structure and their behaviour. By convention, we `Ex` is written, we can mean either `AEx` or `IEx`.

We focus on objective constellations since they are the ones which are studied and used the most in this thesis.

61 Computability of stellar resolution

§61.1 In this section, a few results about execution are detailed. First, our model is Turing-complete, which is not too surprising since it is very close to logic programming which is itself known to be Turing-complete (especially by considering Horn clauses [Hor51, Tär77]) but also able to simulate the aTAM which is also Turing-complete [Win98, Section 3.2.5][Woo15, Section 2].

§61.2 **Proposition** (Turing-completeness)**.** Stellar resolution is Turing-complete.

Proof. Consequence of Theorem 56.25. Although we can encode Turing machines, stellar resolution (with abstract execution) is actually "stronger" but for wrong reasons: the ability to compute infinite normal forms. In particular, it is possible to construct infinite non-uniform families of boolean circuits (*cf.* Section 58) which are known to be theoretically able to decide any language but without concrete implementation of how such families work (for that reason, we usually require families to be *uniform*, *i.e.* that they can be generated by a Turing machine). This is not a problem since we are usually interested in finite constellations and finite normal forms. Moreover, stellar resolution with concrete execution is not affected by this problem. □

§61.3 It is possible to construct decider-constellations corresponding to a computable function $f : \mathbf{N}^k \to \mathbf{N}$ on natural numbers (or boolean deciders $f : \mathbf{N}^k \to \{0, 1\}$). We already have an encoding of natural numbers in Definition 48.17. Let Φ be a constellation. We say

that Φ computes a function $f : \mathbf{N}^k \to \mathbf{N}$ when $\mathtt{Ex}(\Phi + [+i(\overline{n_1}, ..., \overline{n_k})]) = [\overline{f(n_1, ..., f_k)}]$. In case of a function f returning a boolean (or a natural number of $\{0, 1\}$), we say that Φ *accepts* $[+i(\overline{n_1}, ..., \overline{n_k})]$ in case $\mathtt{Ex}(\Phi + [+i(\overline{n_1}, ..., \overline{n_k})]) = [\overline{1}]$ and we say that it *rejects* when $\mathtt{Ex}(\Phi + [+i(\overline{n_1}, ..., \overline{n_k})]) = [\overline{0}]$ (as in automata theory). We say that Φ *recognises* the set of constellations:

$$\mathcal{L}(\Phi) := \{[+i(\overline{n_1}, ..., \overline{n_k})] \mid \Phi \text{ recognises } [+i(\overline{n_1}, ..., \overline{n_k})]\}.$$

As explained in Paragraph 16.9, we can encode most interesting classical datatypes with natural numbers.

62 Classes of constellations

(Non-)Terminating constellations

§62.1 The most remarkable behaviour of constellations is (non-)termination of execution. There exists actually two notions of termination which are distinguished by concrete execution.

§62.2 **Definition** (Visible normalisation)**.** A constellation Φ is *visibly normalising w.r.t.* to a set of colours $C \subseteq F$ when $\mathtt{AEx}_C(\Phi)$ is finite, written $|\mathtt{AEx}_C(\Phi)| < \infty$ or sometimes $|\mathtt{CSatDiags}_C(\Phi)| < \infty$.

§62.3 **Definition** (Strong normalisation)**.** A constellation Φ is *strongly normalising*[a] *w.r.t.* to a set of colours $C \subseteq F_+ \uplus F_-$ when $\mathtt{IEx}_C(\Phi)$ terminates.

[a] The term comes from λ-calculus in which strongly normalisation is the termination of all possible paths of reduction.

§62.4 The difference is clear in the constellation $[-c(X), +c(f(X))]$. With abstract execution, we will have $\mathtt{AEx}([-c(X), +c(f(X))]) = \emptyset$ because it is impossible to construct a saturated diagram. All diagrams will be infinite chains linking $-c(X)$ with $+c(f(X))$. Any cyclic diagram would connect $-c(X)$ with $+c(f(X))$ where the two occurrences of X refer to the same variable but such diagram are incorrect because $\{X \stackrel{?}{=} f(X)\}$ has not solution. The visible output is finite. However, with concrete execution, we would have an infinite loop always introducing new occurrences of the star. There is no way to terminate since it is impossible to tell when to stop. Strong normalisation subsumes visible normalisation.

§62.5 **Black holes.** The previous example can be generalised to what I call a *black hole*. The feature we are looking for with black holes is their ability to erase a given connected constellation (or connected component of a constellation) Φ. If ϕ is a black hole then we want to obtain $\mathtt{Ex}(\Phi + \phi) = \emptyset$. However, there is no generic black hole: they necessarily depends on the shape of Φ.

- With abstract execution, a typical black hole is to choose a star

$$[r, -w(X), +w(f(X))]$$

  where $r \bowtie r'$ for r' a ray of Φ. All diagram of Φ connected to $[r, -w(X), +w(f(X))]$ will never be saturated because it is always to possible to add an occurrence of $[r, -w(X), +w(f(X))]$ and extend the diagram as we wish.

- With concrete execution, the previous black hole yields an infinite loop but not really the erasure we expect. There are several solutions such as embedding execution with an operator such as $\lightning$ as for the interpretation of automata (*cf.* Section 56). We will introduce specific solutions when needed (for the interpretation of weakening in linear logic).

§62.6 What distinguishes terminating constellations from other constellations? It looks like cycles in dependency graphs are related to loops in concrete execution. Cyclic constellations can be non-terminating but also terminating (the logic program for unary addition in Example 48.18). A first intuition comes from rewriting theory and the theory of functional languages [BN98, Chapter 5]. A rewriting system is terminating when all sequences of applications of rewriting rules are decreasing *w.r.t.* some (usually polynomial) measure of the initial term. In other words, termination occurs in converging cycles *consuming* terms of a star.

§62.7 I give a typical example inspired from the consumption of words with finite automata. We have a star $[+a(0\cdot 0\cdot 0\cdot \varepsilon)]$ meant to be consumed by a loop $[-a(0\cdot W), +a(W)]$. The loop, by itself, is not terminating because we can repeat it as many times as we want. It can be connected with itself to produce $[-a(0\cdot 0\cdot W), +a(W)]$, $[-a(0\cdot 0\cdot 0\cdot W), +a(W)]$ and so on. It is has not way to know when to stop, as in black holes. However, it is productive: the negative ray can be *cancelled* by a star. In particular, $[-a(0\cdot 0\cdot 0\cdot W), +a(W)]$ can terminate by interaction with $[+a(0\cdot 0\cdot 0\cdot \varepsilon)]$ and produce $+a(\varepsilon)$. Remark that the constant ε forbids any possible additional looping (as if we had linearised the term[1]). This idea of loop stopped by a ray is the same idea as in recursion/induction where a base case in needed.

Graph-structural classification of constellations

§62.8 The shape of $\mathfrak{D}[\Phi; C]$ for a constellation Φ contains a lot of information about $\mathtt{Ex}_C(\Phi)$. By observing the shape of constellations (as in Chapter 8 for instance), we observe that only cycles make iteration/loops possible and that the plurality of diagrams is caused by branching rays. Since the relationship between the structure of $\mathfrak{D}[\Phi; C]$ and the computational behaviour of $\mathtt{Ex}_C(\Phi)$ is a bit complex, we suggest a few structural classes of constellations and establish theorems which will be useful to reason with constellations.

[1] This is what is actually used for the interpretation of dereliction in linear logic.

§62.9 **Definition** (Properties of constellation)**.** Let Φ be a constellation,

$$\mathfrak{D}[\Phi; C] = (V, E, \mathtt{end})$$

be its dependency graph over a set of colours $C \subseteq F_+ \uplus F_-$. We say that Φ is:

- *finite* if I_Φ is finite (and so are V and E) and it is *infinite* otherwise;
- *exact* if for all $e \in E$ we have $e = \{(i,j),(i',j')\}$ such that $\lfloor \Phi[i][j] \rfloor$ is α-equivalent to $\lfloor \Phi[i'][j'] \rfloor$;
- *acyclic* when $\mathfrak{D}[\Phi; C]$ is acyclic and otherwise it is *cyclic*;
- *connected* when $\mathfrak{D}[\Phi; C]$ is connected and otherwise it is *disconnected*;
- *branching* if it has some branching rays. There are two sub-classes of branching constellations. We say that Φ is:
  - *non-deterministic* if $|\mathtt{CSatDiags}_C(\Phi)| > 1$;
  - *replicating* if $|\mathtt{CSatDiags}_C(\Phi)| = 1$.
- *deterministic* is there is no branching rays.

All the definitions can be naturally parametrised with a set of colours $C \subseteq F_+ \uplus F_-$.

§62.10 **A few words on replication and non-determinism.** Replication is a duplication occurring in a same diagram. Non-determinism is *also* a duplication but splitting diagrams. The two are not so different because they come from the same operation of duplication. The only difference is that the two diagrams can be joined in the case of replication but not in the case of non-determinism. We could talk about *linkable* and *non-linkable* duplication instead. Replication is like having two choices but realising that one choice links to the other: it is then an *illusionary choice*[2].

§62.11 **Example.** We illustrate the properties defined above.

- The constellation $\Phi_{\mathbf{N}}^{n+m}$ of Example 48.18 is connected, cyclic and branching (non-deterministic). The middle star handles recursion but the construction of diagrams can either continue or exit the loop.
- $[+a(X), +a(X)] + [-a(X), -a(X), X]$ is exact, connected, cyclic and branching (non-deterministic).
- $[X, -c(X)] + [+c(f(Y))] + [+c(g(Y))]$ is acyclic, connected and branching (non-deterministic).
- $[+a(\mathtt{l}), +a(\mathtt{r})] + [+b(\mathtt{l}), +b(\mathtt{r})] + [-a(X), -b(X)]$ is connected, cyclic and branching (replicating but not non-deterministic). Two choices are possible for the

[2] I got this idea from a discussion with Valentin Maestracci.

> negative rays but all the stars can appear in the same diagram by duplicating $[-a(X), -b(X)]$ and connecting the `l` (*resp.* `r`) together.
>
> All these constellations are finite.

§62.12 **Proposition.** If a constellation Φ with $\mathfrak{D}[\Phi; C] := (V_{\mathfrak{D}}, E_{\mathfrak{D}}, \mathtt{end}_{\mathfrak{D}})$ is finite and acyclic *w.r.t.* $C \subseteq F_+ \uplus F_-$ then for all diagram (G_δ, δ) such that $G_\delta := (V_\delta, E_\delta, \mathtt{end}_\delta)$ and $\delta : D_\delta \to \mathfrak{D}[\Phi; C]$, we have that δ is injective.

Proof. Assume that for $v, v' \in D_\delta$, $\delta(v) = \delta(v')$. We have to show $v = v'$, meaning that v and v' do not correspond to two occurrences of a same star in $\mathfrak{D}[\Phi; C]$. Assume by contradiction that $v \neq v'$. The only way to have $v \neq v'$ while preserving the adjacency relation (which is required by the definition of diagram, which is a multigraph homomorphism) is to have a path ρ from v to v' such that $\delta(\rho)$ is a path from $\delta(v) = \delta(v')$ to itself. However, this contradicts the acyclicity of Φ. □

§62.13 **Corollary** (Termination of acyclic constellations)**.** If a constellation Φ with

$$\mathfrak{D}[\Phi; C] := (V_{\mathfrak{D}}, E_{\mathfrak{D}}, \mathtt{end}_{\mathfrak{D}})$$

is finite and acyclic *w.r.t.* $C \subseteq F_+ \uplus F_-$ then $|\mathtt{CSatDiags}_C(\Phi)| < \infty$.

Proof. Assume Φ is finite and acyclic. By Proposition 62.12, vertices of diagrams are uniquely taken from $V_{\mathfrak{D}}$ and since stars have finitely many rays which must be uniquely connected, there are finitely many edges. There are only finitely many graphs we can construct with finitely many vertices and edges and in particular $\mathtt{CSatDiags}_C(\Phi)$ is finite. □

§62.14 **Lemma** (Exactness)**.** Let Φ be a constellation which is exact *w.r.t.* a set of colours $C \subseteq F_+ \uplus F_-$. We have $\mathtt{SatDiags}_C(\Phi) = \mathtt{CSatDiags}_C(\Phi)$.

Proof. Let Φ be an exact constellation and $(G_\delta, \delta) \in \mathtt{SatDiags}_C(\Phi)$ one of its diagrams. We already have $\mathtt{CSatDiags}_C(\Phi) \subseteq \mathtt{SatDiags}_C(\Phi)$ by definition. It remains to show that we have $\mathtt{SatDiags}_C(\Phi) \subseteq \mathtt{CSatDiags}_C(\Phi)$ by showing that (G_δ, δ) is correct. Since Φ is correct, all the n equations associated with edges of $\mathfrak{D}[\Phi; C]$ are equations $t_i \stackrel{?}{=} u_i$ for $1 \leq i \leq n$ such that t_i is α-equivalent to u_i, meaning that any solution of $\{t_i \stackrel{?}{=} u_i\}$ must a renaming α. We have to show that $\{t_1 \stackrel{?}{=} u_1, ..., t_n \stackrel{?}{=} u_n\}$ has a solution. If we solve all equations independently, we obtain subproblems with only variables. The only rule which applies is "replace rule" (*cf.* Theorem B.2.4). We finally show that this rule never fails when we only have variables. Assume we have a unification problem P with only variables. If we replace X by Y, either $X \notin P$ and nothing happens or $X \in P$ and all occurrences of X are replaced by Y. We obtain either equations $Y \stackrel{?}{=} Y$ which can

be removed by the "clear rule" or equations $X \stackrel{?}{=} Z$ with $X \neq Z$ and in this case there is no problem. □

§62.15 **Definition.** Let Φ be a constellation. It is called *perfect* when it is finite, connected, exact, acyclic and deterministic.

§62.16 **Lemma** (Perfection)**.** If a constellation Φ is perfect then it has a unique saturated diagram which is correct.

Proof. By Corollary 62.13, we know that Φ has finitely many saturated correct diagrams. By induction on the number n of edges in $\mathfrak{D}[\Phi]$.

- ◇ **Base case** If $n = 0$ then Φ must be made of a unique star $\Phi[i]$, otherwise it would not be connected. We have a unique diagram (G_δ, δ) with $G_\delta := (V, E, \mathtt{end})$, $V := \{v\}$, $E := \emptyset$ and $\delta(v) = i$. This diagram is correct and we have $\Downarrow(G_\delta, \delta) = \Phi[i]$;
- ◇ **Induction** Assume we have a perfect constellation Φ' with dependency graph $\mathfrak{D}[\Phi'] := (V', E', \mathtt{end}')$ with $|E'| = n'$ a unique diagram (G'_δ, δ'). We now consider a bigger constellation Φ with $\mathfrak{D}[\Phi] := (V, E, \mathtt{end})$ and $|E| = |E'| + 1$. We necessarily added a new star $\Phi[j]$, hence $\Phi := \Phi' + \Phi[j]$. Since Φ' is connected, j is connected to the unique connected component of $\mathfrak{D}[\Phi']$. Since Φ is deterministic then $\Phi[j]$ has been linked to a free polarised ray of Φ'. The unique diagram (G_δ, δ) of Φ' (given by induction hypothesis) can be extended with $\Phi[j]$ and by fusion we still obtain a unique bigger saturated diagram (G_δ, δ). Since Φ' and Φ are both exact, it guarantees the correctness of (G_δ, δ).

□

§62.17 **Lemma** (Independence of connected components)**.** Let Φ be a constellation and $\mathfrak{D}[\Phi; C]$ be its dependency graph for a set of colours $C \subseteq F_+ \uplus F_-$. We define K as the set of connected components of $\mathfrak{D}[\Phi; C]$ and for all $k \in K$, we define Φ_k as the restriction of Φ to the stars in K. We have $\mathtt{AEx}_C(\Phi) = \biguplus_{k \in K} \mathtt{AEx}_C(\Phi_k)$.

Proof. Consider a diagram (G_δ, δ) such that $G_\delta := (V_\delta, E_\delta, \mathtt{end}_\delta)$ and $\delta : D_\delta \to \mathfrak{D}[\Phi; C]$. In the proof of Proposition 62.12, we have seen that diagrams preserve reachability of vertices because they preserve the adjacency relation of multigraphs. It follows that for all $v, v' \in V_\delta$, if there is no path from $\delta(v)$ to $\delta(v')$, then there cannot be a path from v to v'. Since diagrams are connected, they must have a connected component of $\mathfrak{D}[\Phi; C]$ as image. Therefore, $\mathtt{CSatDiags}_C(\Phi) = \bigcup_{k \in K} \mathtt{CSatDiags}_C(\Phi_k)$. Since execution is the actualisation of all correct saturated diagrams, it follows that $\mathtt{AEx}_C(\Phi) = \biguplus_{k \in K} \mathtt{AEx}_C(\Phi_k)$. □

§62.18 **Corollary** (Independence of isolated stars)**.** Let Φ be a constellation and $\mathfrak{D}[\Phi; C]$ be its dependency graph for a set of colours $C \subseteq F_+ \uplus F_-$. For each star index $i \in I_\Phi$ such that i is isolated (no adjacent vertex) in $\mathfrak{D}[\Phi; C]$, there is $i' \in I_{\mathtt{AEx}_C(\Phi)}$ such that

$\mathtt{AEx}_C(\Phi)[i'] = \Phi[i']$.

Proof. By Lemma 62.17, in the particular case where we have a connected component Φ_l (with $l \in K$) made of a unique isolated star $\Phi[i]$, if $\Phi = \Phi' + \Phi[i]$ then we would have $\mathtt{AEx}_C(\Phi' + \Phi[i]) = \big(\biguplus_{k\in K-\{l\}} \mathtt{AEx}_C(\Phi_k)\big) + \mathtt{AEx}_C(\Phi_l) = \big(\biguplus_{k\in K-\{l\}} \mathtt{AEx}_C(\Phi_k)\big) + \mathtt{AEx}_C(\Phi[i]) = \big(\biguplus_{k\in K-\{l\}} \mathtt{AEx}_C(\Phi_k)\big) + \Phi[i]$ because the execution of a star is a star (no diagram can be constructed). In particular, if i' is the index of $\Phi[i]$ in the normal form, we have $\mathtt{AEx}_C(\Phi)[i'] = \Phi[i']$. □

63 Stellar transformations

§63.1 From the definition of concrete execution (*cf.* Section 50), we can remark that it is possible to reduce all deterministic links without any impact on the normal form. This corresponds to a sort of pre-normalisation of constellations. It is then possible to only consider constellations without such links. We would only lose convenience or clarity because constellations would be more compact.

§63.2 **Definition** (Stellar compression and compact constellations)**.** Let Φ be a constellation and $C \subseteq F_+ \uplus F_-$ be a set of colours. Its compression $\mathtt{compress}(\Phi)$ is the constellation obtained by the following procedure:

- identify deterministic rays $\Phi[i][j]$ in $\mathfrak{D}[\Phi; C]$ such that $\mathtt{adj}^C_\Phi(i,j) = \{(i',j')\}$ and $\Phi[i][j]$ is also deterministic;
- for each such ray r, we transform $\Phi := \Phi' + \Phi[i]$ into $\Phi' + \Phi[i] \overset{j,j'}{\triangledown} \Phi[i']$.

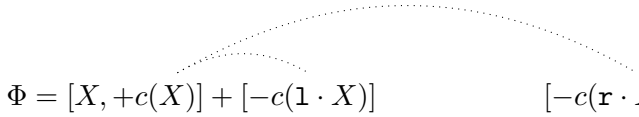
We say that a constellation Φ is *compact* when $\mathtt{compress}(\Phi) = \Phi$.

§63.3 **Example.** For the constellation

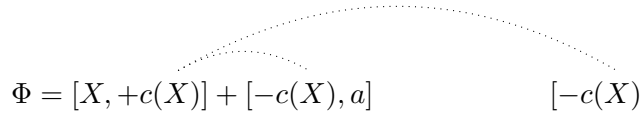
$$\Phi := [-1(X), +4(X)] + [-2(X)] + [-4(X), -3(X), +5(X)] + [-5(X), 5(X)],$$

we have $\mathtt{compress}(\Phi) = [-1(X), -3(X), 5(X)] + [-2(X)]$.

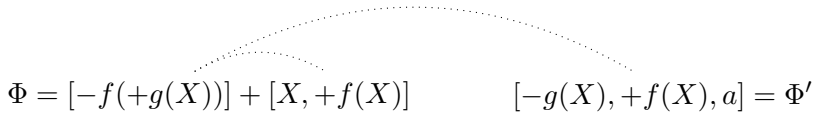
§63.4 There are other operations which we could think of but that we will not be interested in here. For instance, we could think about splitting non-deterministic constellations into several deterministic ones (we make choices explicit with several slices of a same constellation). We could also think about unfolding loops. For instance, for the logic program $\Phi^{2+2}_{\mathbf{N}}$, we could explicitly create a chain of stars unfolding the query so that the constellation is no longer cyclic. This is similar to how we would unfold loops in imperative programming.



(a) We have $\mathtt{Ex}_{\{c\}}(\Phi) = [\mathtt{l} \cdot X]$ and $\mathtt{Ex}_{\{c\}}(\mathtt{Ex}_{\{c\}}(\Phi) \uplus \Phi') = [-c(\mathtt{r} \cdot X)] + [\mathtt{l} \cdot X]$, but $\mathtt{Ex}_{\{c\}}(\Phi \uplus \Phi') = [\mathtt{l} \cdot X] + [\mathtt{r} \cdot X]$ which is different. Notice that both $[-c(\mathtt{l} \cdot X)]$ and $[-c(\mathtt{r} \cdot X)]$ need $[X, +c(X)]$ but when executing Φ, Φ' cannot be connected to it anymore.



(b) We have $\mathtt{Ex}_{\{c\}}(\Phi) = [X, a]$ and $\mathtt{Ex}_{\{c\}}(\mathtt{Ex}_{\{c\}}(\Phi) \uplus \Phi') = [-c(X), b] + [X, a]$, but $\mathtt{Ex}_{\{c\}}(\Phi \uplus \Phi') = [X, a] + [X, b]$ which is different.



(c) We have $\mathtt{Ex}_{\{f\}}(\Phi) = [+g(X)]$ and $\mathtt{Ex}_{\{g\}}(\mathtt{Ex}_{\{f\}}(\Phi) \uplus \Phi') = [+f(X), a]$, but $\mathtt{Ex}_{\{f,g\}}(\Phi \uplus \Phi') = []$ (no saturated diagrams) which is different.

Figure 64.1: Counter-examples for partial pre-execution.

64 Partial pre-execution and confluence

§64.1 An important result is the possibility of partial execution by only reducing rays of some colours on some constellations first then the others without any effect on the normal form, *i.e.* that $\mathtt{Ex}_{C \cup D}(\mathtt{Ex}_D(\Phi) \uplus \Phi') = \mathtt{Ex}_{C \cup D}(\Phi \uplus \Phi')$ for some set of colours C and D. However, this is not valid in general as presented in Figure 64.1. The problem is that stars from two distinct constellations want to access a same ray and an execution of a constellation may erase some potential connexions which were present and required by another constellation. This problem is reminiscent of the idea of *mutual exclusion* in concurrent programming [Dij01]. Also notice that in Figure 64.1c, internal colours add even more complexity: some rays may indirectly access internal colours. Direct access occurs once the internal colour has been "unlocked" by a star extracting it. I choose to omit the case of subjective constellations in this thesis.

§64.2 This property of partial pre-execution is necessary in order to obtain a result of confluence and associativity of execution which are necessary for the definition of linear logic as explained at the end of this chapter. We need to design a precondition for which these properties are valid and from which it is possible to express linear logic.

§64.3 There are several possible choices depending on how demanding we want to be. A simple choice is to reason on the accessibility of rays in a dependency graph. We do not want a ray to be accessible from two different constellations such that one is pre-executed before

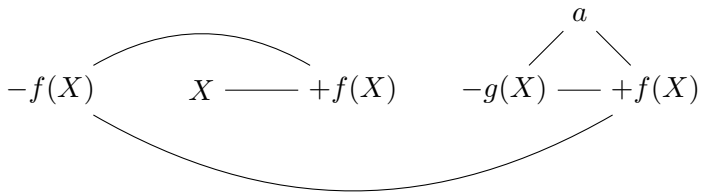


Figure 64.2: Example of interaction map for the constellation $\Phi \uplus \Phi'$ of Figure 64.1c where the internal ray is replaced by a variable X.

the other. For instance, in Figure 64.1, the ray $[+c(X)]$ is accessible both from Φ and Φ'.

§64.4 **Definition** (Interaction map)**.** Let $C \subseteq F_+ \uplus F_-$ be a set of colours. The *interaction map* $\texttt{IM}_C(\Phi)$ of a constellation Φ over C is a graph (V, E) defined with:

- $V := \{(i,j) \in \pm\texttt{IdRays}(\Phi) \mid \texttt{colours}(\Phi[i][j]) \subseteq C\}$;
- $\{(i,j),(i',j')\} \in E$ when:
  - $\Phi[i][j]$ and $\Phi[i][j']$ are linked in $\mathfrak{D}[\Phi; C]$;
  - or $i = i'$ (in order to travel from one star to another).

§64.5 An example of interaction map is given in Figure 64.2.

§64.6 **Observation** (Idea for the case of subjective rays)**.** In order to consider subjective rays, an idea I had was to consider matchability between subrays in the interaction map in order to anticipate the possibility of connexion with internal colours.

§64.7 **Definition** (Shared rays)**.** We define the *set of rays shared* the constellations

$$\Phi_1, ..., \Phi_n$$

w.r.t. a set of colours $C \subseteq F_+ \uplus F_-$ as the set:

$$\Cap_C(\Phi_1, ..., \Phi_n) := \{(\sum_{i=1}^{n} \Phi_i)[i][j] \mid$$

$$(\sum_{i=1}^{n} \Phi_i)[i][j] \text{ is reachable from some } \Phi_i[k][l] \text{for all } 1 \leq i \leq n \text{ in } \texttt{IM}_C(\Phi)$$

$$\text{with a non-empty path}\}.$$

We omit the set of colour when we consider all colours.

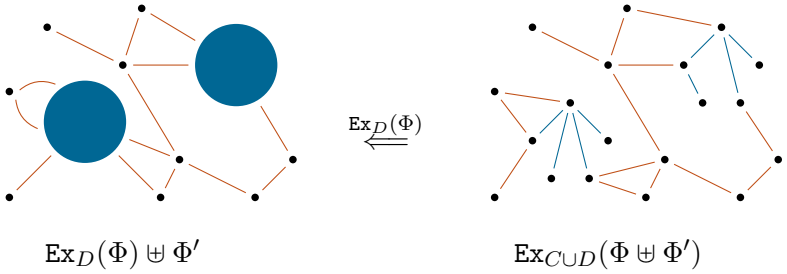


Figure 64.3: Partial execution acts as a partial diagram contraction, which is only possible when Φ and Φ' do not act on a same variable. The "blow-up" obtained by inverting execution preserves the connexions between rays.

§64.8 **Example.** In Figure 64.1, we have $\Cap(\Phi, \Phi') = \{+c(X)\}$ in the two first examples and $\Cap(\Phi, \Phi') = \{-f(X)\}$ in the third one.

§64.9 **Lemma** (Partial pre-execution)**.** Let Φ and Φ' be constellations and $C, D \subseteq F_+ \uplus F_-$ be sets of colours such that $\Cap_{C\cup D}(\Phi, \Phi') = \emptyset$. We have $\mathtt{Ex}_{C\cup D}(\mathtt{Ex}_D(\Phi) \uplus \Phi') = \mathtt{Ex}_{C\cup D}(\Phi \uplus \Phi')$.

Proof. We show the existence of an isomorphism φ between $\mathtt{CSatDiags}_{C\cup D}(\mathtt{Ex}_D(\Phi)\uplus\Phi')$ and $\mathtt{CSatDiags}_{C\cup D}(\Phi \uplus \Phi')$. Assume we have a diagram

$$(G^{PE}, \delta^{PE}) \in \mathtt{CSatDiags}_{C\cup D}(\mathtt{Ex}_D(\Phi) \uplus \Phi')$$

with $G^{PE} := (V^{PE}, E^{PE}, \mathtt{end}^{PE})$, where PE stands for "partial execution" because a subconstellation is executed first. The goal is to associate (G^{PE}, δ^{PE}) with a diagram $\varphi(G^{PE}, \delta^{PE}) \in \mathtt{CSatDiags}_{C\cup D}(\Phi \uplus \Phi')$ and to show that φ is an isomorphism.

The idea, illustrated in Figure 64.3, is that (G^{PE}, δ^{PE}) (on the left) can be decomposed into two parts:

- vertices $v \in V^{PE}$ such that $\delta^{C\cup D}(v) = k$, representing stars $\mathtt{Ex}_D(\Phi)[k]$ (in blue);
- vertices $v \in V^{PE}$ such that $\delta^{C\cup D}(v) = j$, representing stars $\Phi'[j]$ with:
  - edges $E_{\Phi'} \subseteq E^{PE}$ linking stars of Φ' together (in red);
  - edges $E_{\Phi,\Phi'} \subseteq E^{PE}$ linking stars of Φ' with stars $\mathtt{Ex}_D(\Phi)[k]$ (in red);

The stars $\mathtt{Ex}_D(\Phi)[k]$ come from diagrams $(G^D_k, \delta^D_k) \in \mathtt{CSatDiags}_D(\Phi)$, *i.e.* there are diagrams $(G^D_k, \delta^D_k) \in \mathtt{CSatDiags}_D(\Phi)$ such that $\Downarrow(G^D_k, \delta^D_k) = \mathtt{Ex}_D(\Phi)[k]$, because these stars are part of the execution of Φ, by definition of the abstract execution $\mathtt{Ex}_D := \mathtt{AEx}_D$. We can perform a "star expansion" (right diagram in Figure 64.3) on $\mathtt{Ex}_D(\Phi)[k]$ by replacing it with its corresponding diagram (G^D_k, δ^D_k). In some sense, we are reversing execution. The isomorphism φ associates (G^{PE}, δ^{PE}) with a new diagram $\varphi(G^{PE}, \delta^{PE})$

corresponding to (G^{PE}, δ^{PE}) where this operation of star expansion has been applied on stars $\texttt{Ex}_D(\Phi)[k]$ for all k. This operation works as follows:

1. we start from our diagram (G^{PE}, δ^{PE}) and apply the same procedure on all stars $\texttt{Ex}_D(\Phi)[k] = [r_1, ..., r_n]$ (blue circles on the left);
2. by definition of abstract execution, the star $\texttt{Ex}_D(\Phi)[k]$ must come from some actualised diagram $\Downarrow(G^D_k, \delta^D_k)$ such that each ray r_i corresponds to a ray $\delta(v)[i_k]$;
3. we can construct a new diagram $\varphi(G^{PE}, \delta^{PE})$ (on the right) by taking (G^{PE}, δ^{PE}), removing the vertex u such that $\delta(u) = \texttt{Ex}_D(\Phi)[k]$ and adding the subgraph G^D_k. We then extend δ^{PE} with δ^D_k.
4. The links between rays r_i and rays of Φ' with colour in C (on the left) are then replaced by links between rays r_i and rays $\delta(v)[i_k]$ (on the right).
5. By our hypothesis $\Cap_{C\cup D}(\Phi, \Phi') = \emptyset$, the stars of Φ and of Φ' cannot interfere by acting on a same ray of a same star. It follows that the actualisation of diagrams (G^D_k, δ^D_k) preserve the links between rays r_i and rays $\delta(v)[i_k]$. In other words, the execution of blue diagrams are independent from the rest of the diagram $\varphi(G^{PE}, \delta^{PE})$. Without this precondition, some connexions could disappear (as in Figure 64.1) and we would not preserve all connexions allowing this inversion of execution.

We obtain diagrams $\varphi(G_{\delta^D_k}, \delta^D_k)$ corresponding to diagrams $(G_{\delta^D_k}, \delta^D_k)$ extended with stars of Φ' in exactly the same way as how ϕ^D_k can be connected with Φ'. We have $\varphi(\delta^D_k) \in \texttt{CSatDiags}_{C\cup D}(\Phi \uplus \Phi')$ since it connects stars of both Φ and Φ'.

It remains to show that φ is invertible so that we have an isomorphism between

$$\texttt{CSatDiags}_{C\cup D}(\texttt{Ex}_D(\Phi) \uplus \Phi')$$

and

$$\texttt{CSatDiags}_{C\cup D}(\Phi \uplus \Phi').$$

Assume we have

$$(G_{\delta'}, \delta') \in \texttt{CSatDiags}_{C\cup D}(\Phi \uplus \Phi').$$

We would like to define $\varphi^{-1}(G_{\delta'}, \delta')$. By the confluence of diagram reduction (*cf.* Corollary 49.36), we can apply fusion first on the stars coming from Φ using colours in D and then on the colours in C. By doing so, we obtain a diagram $\varphi^{-1}(G_{\delta'}, \delta') \in \texttt{CSatDiags}_{C\cup D}(\texttt{Ex}_D(\Phi) \uplus \Phi')$. We have $\varphi(\varphi^{-1}(\delta)) = \delta$ and $\varphi^{-1}(\varphi(\delta)) = \delta$ because of the relationship between stars $\texttt{Ex}_D(\Phi)[k]$ and the diagrams $(G_{\delta^D_k}, \delta^D_k)$ they come from. Actualising then expanding or the converse is equivalent to the identity homomorphism $\varphi_i d(G, \delta) = (G, \delta)$, by design. □

§64.10 **Theorem** (Confluence)**.** For any constellation Φ, and $C, D \subseteq F_+ \uplus F_-$ two disjoint sets of colours, we have $\mathtt{Ex}_D(\mathtt{Ex}_C(\Phi)) = \mathtt{Ex}_{C\cup D}(\Phi) = \mathtt{Ex}_C(\mathtt{Ex}_D(\Phi))$.

Proof. By Lemma 64.9 with $\Phi' := \emptyset$ (in this case we trivially have $\Cap_{C\cup D}(\Phi, \emptyset) = \emptyset$ which is the required precondition) we have $\mathtt{Ex}_{C\cup D}(\mathtt{Ex}_D(\Phi)) = \mathtt{Ex}_{C\cup D}(\Phi)$. Since $\mathtt{Ex}_D(\Phi)$ already applies fusion on all pairs of rays of colour in D and there is nothing left of colour D in $\mathtt{Ex}_{C\cup D}$, we have $\mathtt{Ex}_{C\cup D}(\mathtt{Ex}_D(\Phi)) = \mathtt{Ex}_C(\mathtt{Ex}_D(\Phi))$, hence $\mathtt{Ex}_C(\mathtt{Ex}_D(\Phi)) = \mathtt{Ex}_{C\cup D}(\Phi)$. Since $C \cup D = D \cup C$, we also have $\mathtt{Ex}_{C\cup D}(\Phi) = \mathtt{Ex}_{D\cup C}(\Phi)$. By using again Lemma 64.9, we finally obtain $\mathtt{Ex}_{D\cup C}(\Phi) = \mathtt{Ex}_D(\mathtt{Ex}_C(\Phi))$. □

65 Discussion: the sufficient conditions for logical emergence

§65.1 Properties which are necessary to define (linear) logic as we know it do not naturally appear in stellar resolution which is very chaotic. It is then necessary to consider only constellations in some specific cases. Again, this characterisation of "logical constellations" may be characterised by tests (represented by constellations), consistently with the philosophy of transcendental syntax, as we will see later.

§65.2 The first necessary condition for (linear) logic is the idea of mutual exclusion previously defined. This condition led to confluence results. Why are they *necessary*? We will later see that confluence leads to associativity of execution, itself leading to a logical property called *adjunction*.

§65.3 **Adjunction.** Although already mentioned in Girard's writing [Gir11a, Section 14.2.1] and made explicit by Seiller [Sei12a, Section 2], the *adjunction property* is a necessary condition for the reconstruction of linear logic. The adjunction property is related to adjunctions in category theory. This property defines *linear implication*. In Seiller's work it is used as a way to prove that linear implication $\mathbf{A} \multimap \mathbf{B} := \mathbf{A}^\perp \parr \mathbf{B}$ defined by realisability interpretation has a functional behaviour, *i.e.* in the case of stellar resolution, that:

$$\mathbf{A} \multimap \mathbf{B} = \{\Phi \mid \Psi \in \mathbf{A}, \mathtt{Ex}(\Phi \uplus \Psi) \in \mathbf{B}\}.$$

§65.4 **Trefoil property.** Seiller shows that the adjunction could be generalised by a property called *trefoil property* [Sei16a, Section 2.2]. This property is more general but sufficient for the definition of linear logic. For that reason, it will be used in the interpretation of linear logic in this thesis. This trefoil property is often the consequence of associativity of binary execution which comes itself from confluence. Both only hold under some conditions as explained above.

§65.5 The understanding of these necessary conditions is part of transcendental syntax and has yet to be explored. In particular, in his paper *"La syntaxe Transcendantale, manifeste"* [Gir11c, Section 2.4.3], Girard explicitly mentions that epidictic architectures must be closed by adjunction (*cf.* Paragraph 44.17).

Chapter 10

Stellar interpretation of multiplicative linear logic

In this chapter, we interpret linear logic with stellar resolution to provide a technical illustration of transcendental syntax. We start with multiplicative linear logic (MLL).

66 Proofs as constellations

§66.1 The representation of MLL proofs is not very surprising: it directly follows the GoI interpretation of proofs as finite permutations but with terms instead. Actually, it is simply a variant of the interpretation with flows. However, a difference with flows is that we have a little trick with the representation of cuts and that we are able to express Danos-Regnier correctness.

§66.2 **Convention** (Basis of representation)**.** In order to encode proof-structures, we fix a *basis of representation* which is a polarised signature $\mathcal{B} := (V, F, \mathtt{ar}, \supset, \lfloor\cdot\rfloor)$ with any set of variables such that $X \in V$, function symbols $F := F_0 \uplus F_+ \uplus F_-$ with $F := \{\mathtt{l}, \mathtt{r}, \cdot\} \cup \bigcup_{u\in U}\{u\}$, $F_+ := \bigcup_{u\in U}\{+u\}$ and $F_- := \bigcup_{u\in U}\{-u\}$ for U a set of elements which be used to represent vertices of proof-structures (we can set $U := \mathbf{N}$). We have $\mathtt{ar}(u) = 1$ for all $u \in U$, $\mathtt{ar}(\cdot) = 2$ and $\mathtt{ar}(\mathtt{l}) = \mathtt{ar}(\mathtt{r}) = 0$. The symbol $\cdot$ is considered right-associative, *i.e.* $t \cdot u \cdot v := t \cdot (u \cdot v)$.

§66.3 Similarly to unlabelled proof-structures, constellations are purely "locative": only "physical locations" appearing in a proof-structure $\mathcal{S}$ are translated, without giving any serious meaning to labels. We would like to associate a unique address in $\mathtt{Term}(\mathbb{B})$ with atoms $v \in \mathtt{Atoms}(\mathcal{S})$ of a proof-structure $\mathcal{S}$. The address of v will be a term $v'(t)$ where t is a path encoded as a sequence of $\mathtt{l}$ (left) and $\mathtt{r}$ (right) symbols representing the direction to follow in $\mathcal{S}$ to get from the conclusion $v' \in \mathtt{Concl}(\mathcal{S})$ to the atom v. In other words, we are hard-coding the structure of proof-structures directly in axioms.

§66.4 For convenience, we suggest an inductive definition of proof-structures based on their underlying hypergraph. It is from this inductive representation that address of atoms in a proof-structure will be defined.

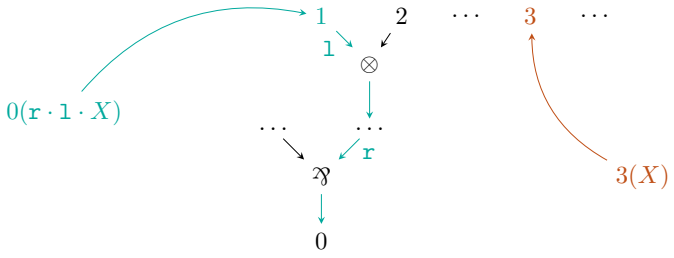


Figure 66.1: Addressing of the atoms 1 and 3 in a proof-structure relatively to the conclusion they come from. Since the first instruction to reach 1 from 0 is to go right, the outermost symbol of the address of 0 should be r.

§66.5 **Remark** (Inductive definition of proof-structures)**.** A proof-structure with only one hyperedge is necessarily an axiom with two conclusions, written $\mathtt{Ax}_{u,v}$. Then a proof-structure $\mathcal{S}$ with n hyperedges is either built from the union of two proof-structures, with respectively k and $n-k$ hyperedges (written $\mathcal{S}_1 \uplus \mathcal{S}_2$), or from a proof-structure with $n-1$ hyperedges extended by either a $\otimes$, $\parr$, or cut hyperedge on two of its conclusions u (left) and v (right). Those latter proof-structures are written $\mathtt{Tens}^{u,v}(\mathcal{S}')$, $\mathtt{Par}^{u,v}(\mathcal{S}')$ and $\mathtt{Cut}^{u,v}(\mathcal{S}')$.

§66.6 **Definition** (Vertex above another one)**.** A vertex v is *above* another vertex u, written in a proof-structure if there exists a directed path from v to u going through only $\otimes$ and $\parr$ hyperedges.

§66.7 **Definition** (Address of an atom)**.** We define the *path address* $\mathtt{pAddr}_{\mathcal{S}}(v)$ of an atom v in a proof-structure $\mathcal{S}$ inductively (*cf.* Remark 66.5):

- if $\mathcal{S} \in \{\mathtt{Ax}_{v,*}, \mathtt{Ax}_{*,v}\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = X$;
- if $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$ and $v \in V^{\mathcal{S}_i}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{pAddr}_{\mathcal{S}_i}(v)$;
- if $\mathcal{S} \in \{\mathtt{Par}^{v',*}(\mathcal{S}'), \mathtt{Tens}^{v',*}(\mathcal{S}')\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{l} \cdot \mathtt{pAddr}_{\mathcal{S}'}(v)$ with v being a conclusion such that v is above v';
- if $\mathcal{S} \in \{\mathtt{Par}^{*,v'}(\mathcal{S}'), \mathtt{Tens}^{*,v'}(\mathcal{S}')\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{r} \cdot \mathtt{pAddr}_{\mathcal{S}'}(v)$ with v being a conclusion such that v is above v';
- $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{pAddr}_{\mathcal{S}'}(v)$ otherwise.

The path address to v is uniquely defined *w.r.t.* to a conclusion $c \in \mathtt{Concl}(\mathcal{S}')$ where $\mathcal{S}'$ is $\mathcal{S}$ without cuts, *i.e.* $E^{\mathcal{S}'} = E^{\mathcal{S}} \backslash \mathtt{Cuts}(\mathcal{S})$ and the rest of $\mathcal{S}$ is defined as in $\mathcal{S}'$.

The *address* of v is then defined as the term $\mathtt{addr}_{\mathcal{S}}(v) := c(\mathtt{pAddr}_{\mathcal{S}}(v))$.

§66.8 **Example.** Figure 66.1 illustrates the idea of addressing of atoms. The address of the atom 1 in Figure 66.2 is $7(\mathtt{l} \cdot X)$ because it is reachable from the conclusion 7 by going to the left premise and the address of the atom 3 is $3(X)$ because it is directly reachable.

§66.9 **Proposition.** Let $\mathcal{S}$ be a proof-structure. For all $v, v' \in \mathtt{Atoms}(\mathcal{S})$ such that $v \neq v'$, we have that $\mathtt{addr}_{\mathcal{S}}(v) \neq \mathtt{addr}_{\mathcal{S}}(v')$, meaning that all atoms have pairwise distinct addresses.

Proof. By definition, all vertices of $\mathcal{S}$ are distinct, and in particular all conclusions are. Assume v is reachable from a conclusion $c \in \mathtt{Concl}(\mathcal{S})$ but not from another conclusion $c' \in \mathtt{Concl}(\mathcal{S})$ such that $c \neq c'$ whereas v' is reachable from c' but not c. The address of v is of shape $c(...)$ whereas the address of v' is of shape $c'(...)$, making the two addresses different. Now assume they both are reachable from the set conclusion c. We have that $\mathtt{addr}_{\mathcal{S}}(v)$ is of shape $c(t)$ and $\mathtt{addr}_{\mathcal{S}}(v')$ of shape $c(t')$ for some t and t'. The terms t and t' are made of sequences of symbols $\mathtt{l}$ and $\mathtt{r}$ depending on if the atom is reachable by going on the left or the right of a $\parr$ or $\otimes$ hyperedge (which have two inputs). If the two atoms v and v' are reached with exactly the same path then they must be identical atoms, which contradicts the hypothesis $v \neq v'$. Hence, they must have a different path and $t \neq t'$. It follows that $c(t) \neq c(t')$ and $\mathtt{addr}_{\mathcal{S}}(v) \neq \mathtt{addr}_{\mathcal{S}}(v')$. □

§66.10 **Definition** (Set of addresses)**.** We define the *set of addresses* as a set $\mathrm{Addr}_x(\mathcal{S})$ containing terms $\mathtt{addr}_{\mathcal{S}}(v)$ for any v, *i.e.* it is the countable set of all terms of the form $c(f_1 \cdot ... \cdot f_n \cdot X)$ where $c \in \mathtt{Concl}(\mathcal{S})$ and $f_i \in \{\mathtt{l}, \mathtt{r}\}$.

§66.11 **Definition** (Translation of the computational content of a proof)**.** The *vehicle* and the *cuts* of a proof-structure $\mathcal{S}$ are respectively defined by the following constellations:

$$\Phi_{\mathcal{S}}^{\mathrm{ax}} := \sum_{e \in \mathtt{Ax}(\mathcal{S})} [\mu(\mathtt{addr}_{\mathcal{S}}(\overleftarrow{e})), \mu(\mathtt{addr}_{\mathcal{S}}(\overrightarrow{e}))], \quad \Phi_{\mathcal{S}}^{\mathrm{cut}} := \sum_{e \in \mathtt{Cuts}(\mathcal{S})} [-\overleftarrow{e}(X), -\overrightarrow{e}(X)].$$

such that $\mu(c(t)) = +c(t)$ when $c = \overleftarrow{e}$ or $c = \overrightarrow{e}$ for some $e \in \mathtt{Cuts}(\mathcal{S})$ (it is related to a cut) and $\mu(x) = x$ otherwise. We define the *computational content* of $\mathcal{S}$ as the constellation $\Phi_{\mathcal{S}}^{\mathtt{comp}} := \Phi_{\mathcal{S}}^{\mathrm{ax}} \uplus \Phi_{\mathcal{S}}^{\mathrm{cut}}$.

§66.12 **Example.** An example of proof-structure translated into a constellation is given in Figure 66.2. Notice that the cut star can interact with the atoms of root 7 and 8. The trick mentioned in the introduction is that we have only one cut star, exactly as in proof-structures but it will duplicate during execution to connect to sub-formulas.

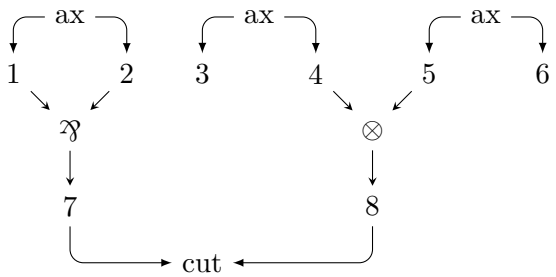


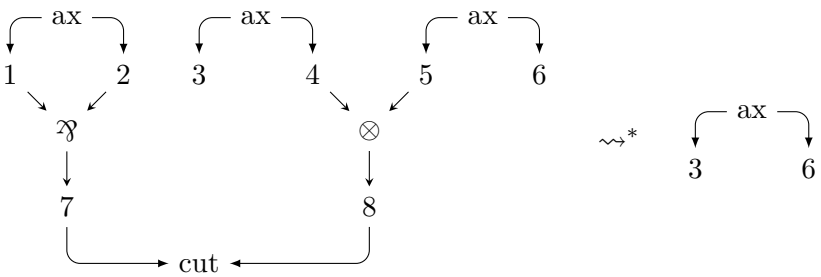
Figure 66.2: The above proof-structure is interpreted by the constellation $[+7(\mathtt{l}\cdot X), +7(\mathtt{r}\cdot X)] + [3(X), +8(\mathtt{l}\cdot X)] + [+8(\mathtt{r}\cdot X), 6(X)] + [-7(X), -8(X)]$.

67 Simulation of cut-elimination

§67.1 Cut-elimination in stellar resolution makes the vehicle interact with cuts by execution. Remark that the shape of proof-structures is directly embedded in the addresses of atoms of the vehicle. The consequence is that cut-elimination is nothing more than a process of contraction / transfer of information by resolution of addresses / conflicts. In particular, it shows that the ax/cut case of cut-elimination is the only "true" case of cut-elimination. The multiplicative case $\otimes/⅋$ purely depends on the shape of objects we are evaluating, and is therefore of a logical nature (in the sense of transcendental syntax).

§67.2 The idea is simply to execute the translation of the vehicle and cuts of a proof-structure. However, execution does not perfectly match the steps of cut-elimination. We start by looking at examples of execution of such constellations representing proof-structures. We expect diagrams to correspond to maximal paths from two ends of a proof-structure alternating between vehicle and cuts.

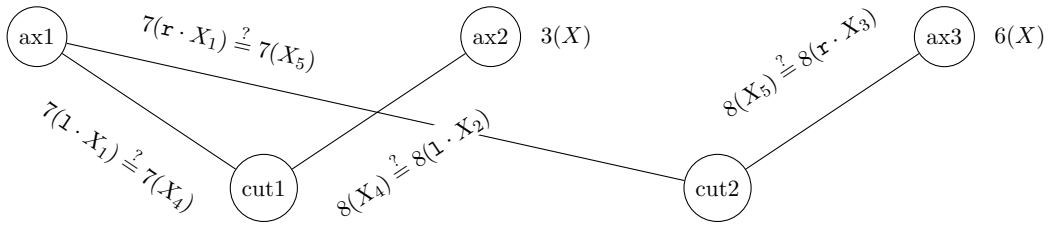
§67.3 **Example** (Correct cut-elimination)**.** We have the following reduction $\mathcal{S} \rightsquigarrow^* \mathcal{S}'$ of proof-structure:



As shown in Figure 66.2, the proof-structure $\mathcal{S}$ is translated into $\Phi_{\mathcal{S}}^{\mathtt{comp}} =$

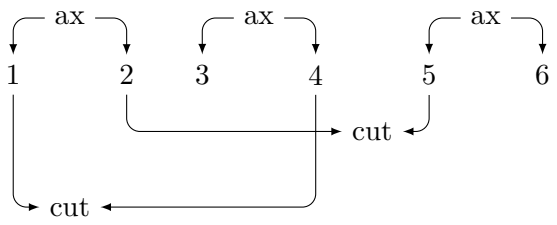
$$[+7(\mathtt{l}\cdot X), +7(\mathtt{r}\cdot X)] + [3(X), +8(\mathtt{l}\cdot X)] + [+8(\mathtt{r}\cdot X), 6(X)] + [-7(X), -8(X)].$$

The ray $-7(X)$ can match either $+7(\mathtt{l}\cdot X)$ or $+7(\mathtt{r}\cdot X)$ and it is the same for $8(X)$. In order to satisfy these requirements of α-unification, the cut star must be duplicated and each occurrence of cut must connect rays with the same address, *i.e.* the path addresses $\mathtt{l}\cdot X$ together and not $\mathtt{l}\cdot X$ with $\mathtt{r}\cdot X$. We obtain the following diagram:



By case analysis, it is easy to check that it is the only possible diagram. Since the α-unification is trivial, it is simply a graph contraction doing no more than renamings and we get $\mathtt{AEx}(\Phi_{\mathcal{S}}^{\mathtt{comp}}) = [3(X), 6(X)] = \mathtt{AEx}(\Phi_{\mathcal{S}'}^{\mathtt{comp}})$.

If we were considering this proof-structure obtained after one step of multiplicative cut-elimination:



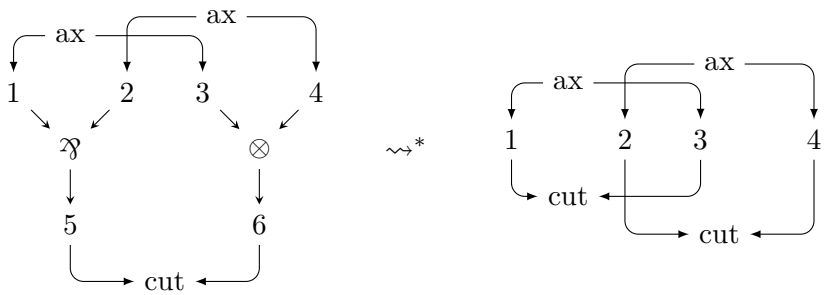
then the translation of axioms would be

$$[+1(X), +2(X)] + [3(X), +4(X)] + [+5(X), 6(X)].$$

Remark that a re-addressing of atoms occurs that the execution of constellations do not consider. Hence, execution do not faithfully and exactly simulate cut-elimination steps but it simulates full cut-elimination properly.

§67.4 In the previous example, although two occurrences of cuts appear in diagrams, exactly like how cuts are duplicated when eliminating $\otimes/\parr$ cuts in proof-structures. However, it is the actualisation of those diagrams that are considered. Actually, if we had other $\otimes$ and $\parr$ hyperedges above the vertices 1, 2, 4 or 5 of Example 67.3, execution of stellar resolution would duplicate and contract the cuts for all possible pairs of atoms without leaving intermediate cuts. It does not care about the structure of proof-structures since it is embedded as addresses of atoms themselves, they do not obstruct evaluation. I now present an example where cut-elimination should fail.

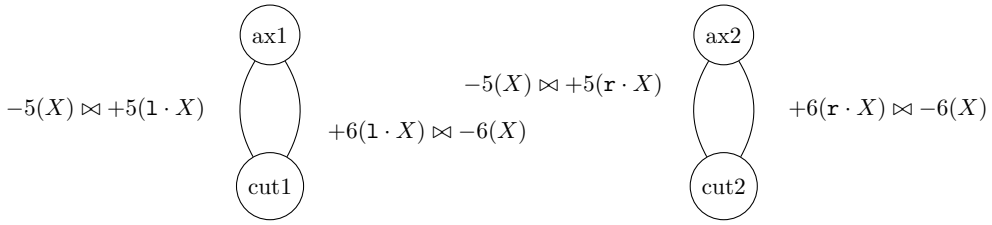
§67.5 **Example** (Incorrect cut-elimination)**.** We have the following reduction $\mathcal{S} \rightsquigarrow^* \mathcal{S}'$ of proof-structure:



The proof-structure $\mathcal{S}$ is translated into $\Phi_{\mathcal{S}}^{\mathtt{comp}} :=$

$$[+5(\mathtt{l} \cdot X), +6(\mathtt{l} \cdot X)] + [+5(\mathtt{r} \cdot X), +6(\mathtt{r} \cdot X)] + [-5(X), -6(X)]$$

with the following dependency graph $\mathfrak{D}[\Phi_S^{\mathtt{comp}}]$:

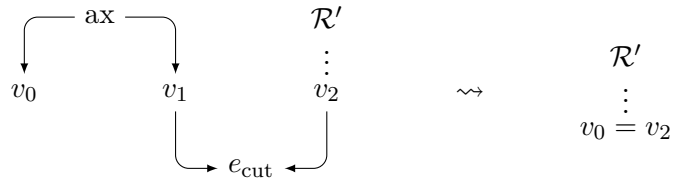


The cycles in $\mathfrak{D}[\Phi_S^{\mathtt{comp}}]$ can be unfolded and it yields infinitely many saturated correct diagrams, all actualising into []. We have $\mathtt{AEx}(\Phi_{\mathcal{S}}^{\mathtt{comp}}) = \sum_{i=1}^{\infty}[] = \mathtt{AEx}(\Phi_{\mathcal{S}'}^{\mathtt{comp}})$.

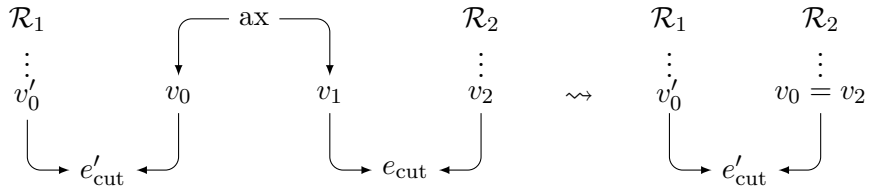
§67.6 In order to prove what we simulate correctly cut-elimination with execution of constellations, we will need to consider structural equivalences of constellations because there is an implicit re-addressing of atoms, as shown in Example 67.3.

§67.7 **Definition** (Structurally equivalent constellations)**.** We say that two constellations Φ and Φ' are *structurally equivalent w.r.t.* two sets of colours C and D, written $\Phi \simeq_S^{C,D} \Phi'$, when there is a bijection $\varphi : I_\Phi \to I_{\Phi'}$ such that $|I_{\Phi[i]}| = |I_{\varphi(\Phi[i])}|$ for all $i \in I_\Phi$. It is extended on rays and we require that $\Phi[i][j] \bowtie \Phi[i'][j']$ if and only if $\varphi(\Phi[i][j]) \bowtie \varphi(\Phi[i'][j'])$ for all $i \in I_\Phi$ and $j \in I_{\Phi[i]}$.

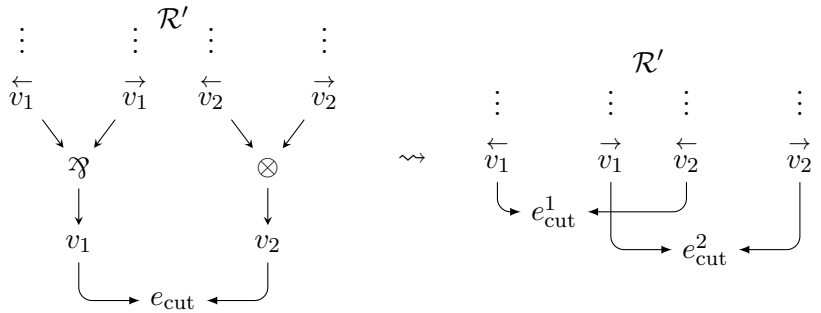
§67.8 **Example.** For instance $[X, +f(X)] + [-f(a)]$ and $[Y, -g(Y)] + [+g(h(X, X))]$ are structurally equivalent constellation. None of them are structurally equivalent to $[+f(X)] + [-f(a)]$.



(a) Case of an ax/cut cut with v_0 not connected to a cut.



(b) Case of an ax/cut cut with v_0 connected to a cut.



(c) Case of an ax/cut cut with v_0 connected to a cut.

Figure 67.1: Illustration of the simulation of cut-elimination. We have $\mathcal{R}$ on the left-hand side and $\mathcal{S}$ on the right-hand side.

§67.9 **Lemma** (Simulation of cut-elimination)**.** Let $\mathcal{R} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E)$ be a proof-structure. If $\mathcal{R} \rightsquigarrow \mathcal{S}$ by eliminating e_{cut}, then $\mathtt{AEx}(\Phi^{\mathtt{comp}}_{\mathcal{R}}) \simeq_S \mathtt{AEx}(\Phi^{\mathtt{comp}}_{\mathcal{S}})$.

Proof. We show that we have $\Downarrow \mathtt{CSatDiags}(\Phi^{\mathtt{comp}}_{\mathcal{R}}) \simeq \Downarrow \mathtt{CSatDiags}(\Phi^{\mathtt{comp}}_{\mathcal{S}})$. By definition of proof-structures, we must have $\mathtt{in}(e_{\text{cut}}) = (v_1, v_2)$ for $v_1, v_2 \in V$. Those vertices must be conclusions of some hyperedges $e_1, e_2 \in E$. By case analysis on $\ell_E(e_1)$ and $\ell_E(e_2)$. The cases are illustrated in Figure 67.1.

◇ **Ax/cut case** Assume e_{cut} is an ax/cut cut with $\ell_E(e_1) = \text{ax}$ such that $\mathtt{out}(e_1) = (v_0, v_1)$. There are two cases depending on if v_0 is related to another cut (which influences the presence of polarity in the translation of v_0).

- Assume v_0 is not related to a cut. This case is illustrated in Figure 67.1a. We have that v_2 must be conclusion of some proof-structure $\mathcal{R}'$ by definition of proof-structure (it cannot be related to a cut because cuts only relate premises). After cut-elimination, we expect to obtain $\mathcal{R}'$ in which v_0 is identified with v_2 (we can replace v_2 by v_0 for instance).

  In $\Phi^{\mathtt{comp}}_{\mathcal{R}}$, we have an axiom $[v_0(X), +v_1(X)]$, a cut $[-v_1(X), -v_2(X)]$, and $\Phi^{\mathtt{comp}}_{\mathcal{R}'}$ in which there are rays $+v_2(t_1)$, ..., $+v_2(t_n)$ for all atoms reachable from the conclusion v_2 (by the translation of proof-structures). Those latter rays must be part of some star $\phi_i := [+v_2(t_i), r_i]$ for some ray r_i by definition (because only axioms and cuts are translated, and they become binary stars). We then expect $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ to be $\Phi^{\mathtt{comp}}_{\mathcal{R}'}\{v_2(t_i) := v_0(t_i)\}$, in which $\{v_2(t_i) := v_0(t_i)\}$ means that the rays $+v_2(t_i)$ are replaced by $v_0(t_i)$.

  We reason on all diagrams (G_δ, δ) of $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ which include e_{cut} and e_1 (which is an axiom), *i.e.* if $G_\delta := (V_\delta, E_\delta, \mathtt{end}_\delta)$ then there are some adjacent vertices $u, u' \in V_\delta$ such that $\delta(u) = [v_0(X), +v_1(X)]$ and $\delta(u') = [-v_1(X), -v_2(X)]$. By confluence of diagram contraction (*cf.* Corollary 49.36), we can contract edges in any order. We contract the link between u and u' and obtain (by fusion) a new star $[v_0(X), -v_2(X)]$, *i.e.* $(G_\delta, \delta) \rightsquigarrow (G_{\delta'}, \delta')$ with $G_{\delta'} := (V_{\delta'}, E_{\delta'}, \mathtt{end}_{\delta'})$, $w \in V_{\delta'}$ and $\delta'(w)$ corresponding to $[v_0(X), -v_2(X)]$. This star $[v_0(X), -v_2(X)]$ is linked to the stars ϕ_i of $\Phi^{\mathtt{comp}}_{\mathcal{R}'}$. By contracting this latter edge, we obtain $\sum_{i=1}^{n}[v_0(t_i), r_i]$ in a diagram $(G_{\delta''}, \delta'')$ such that $(G_{\delta'}, \delta') \rightsquigarrow (G_{\delta''}, \delta'')$. All other edges of $(G_{\delta''}, \delta'')$ and all other diagrams are exactly constructed from edges of $\mathfrak{D}[\Phi^{\mathtt{comp}}_{\mathcal{S}}]$. It follows that $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ and $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ have the same diagrams, and in particular the same correct and saturated diagrams, *i.e.* $\Downarrow \mathtt{CSatDiags}(\Phi^{\mathtt{comp}}_{\mathcal{R}}) \simeq \Downarrow \mathtt{CSatDiags}(\Phi^{\mathtt{comp}}_{\mathcal{S}})$.

- Now assume v_0 is subject to a cut $e'_{\text{cut}} \in \mathtt{Cuts}(\mathcal{R})$ such that $\mathtt{out}(e'_{\text{cut}}) = (v'_0, v_0)$ for some v'_0 conclusion of a sub-proof-structure $\mathcal{R}_0$. This case is illustrated in Figure 67.1b. The vertex v'_0 is conclusion of some proof-structure $\mathcal{R}_1$ and the vertex v_2 is conclusion of some proof-structure $\mathcal{R}_2$. After cut-elimination, we obtain the cut e'_{cut} between v'_0 and v_0 which is identified with v_2.

A difference with the previous case is that the vertex v_0 is translated into $+v_0(X)$ since it is subject to a cut. We have that $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ contains an axiom $[+v_0(X), +v_1(X)]$, two cuts $[-v'_0(X), -v_0(X)] + [-v_1(X), -v_2(X)]$, the constellation $\Phi^{\mathtt{comp}}_{\mathcal{R}_1}$ with rays $+v'_0(u_1)$, ..., $+v'_0(u_m)$ for all atoms reachable from v'_0 and the constellation $\Phi^{\mathtt{comp}}_{\mathcal{R}_2}$ with rays $+v_2(t_1)$, ..., $+v_2(t_n)$ for all atoms reachable from v_2. As in the previous case, those rays $+v_2(t_i)$ must be part of some star $\phi_i := [+v_2(t_i), r_i]$ for some ray r_i. After cut-elimination, we expect $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ to be $\Phi^{\mathtt{comp}}_{\mathcal{R}_1} \uplus \Phi^{\mathtt{comp}}_{\mathcal{R}_2}\{+v_2(t_i) := +v_0(t_i)\} + [-v'_0(X), -v_0(X)]$.

We apply exactly the same reasoning and the same diagram contractions are in the previous case and obtain a diagram $(G_{\delta''}, \delta'')$ except that instead of having stars $\phi_i := [v_0(t_i), r_i]$, we have stars $\phi'_i := [+v_0(t_i), r_i]$. We obtain exactly $\Phi^{\mathtt{comp}}_{\mathcal{R}_1} \uplus \Phi^{\mathtt{comp}}_{\mathcal{R}_2}\{+v_2(t_i) := +v_0(t_i)\} + [-v'_0(X), -v_0(X)]$. All extensions or other diagrams are those of $\Phi^{\mathtt{comp}}_{\mathcal{S}}$. It follows that $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ and $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ have the same diagrams.

Assume e_{cut} is an ax/cut cut with $\ell_E(e_1) = \text{ax}$ such that $\mathtt{out}(e_1) = (v_0, v_1)$. There are two cases depending on if v_0 is related to another cut (which influences the presence of polarity in the translation of v_0).

◇ **Par/tensor case** Assume e_{cut} is a $\parr/\otimes$ cut with $\ell_E(e_1) = \parr$ and $\ell_E(e_2) = \otimes$ with $\mathtt{in}(e_1) = (\overleftarrow{v_1}, \overrightarrow{v_1})$ and $\mathtt{in}(e_2) = (\overleftarrow{v_2}, \overrightarrow{v_2})$. This case is illustrated in Figure 67.1c. The vertices $\overleftarrow{v_1}$, $\overrightarrow{v_1}$, $\overleftarrow{v_2}$, and $\overrightarrow{v_2}$ are conclusion of some proof-structure $\mathcal{R}'$. The resulting proof-structure after cut-elimination is $\mathcal{S} := (V', E', \mathtt{in}', \mathtt{out}', \ell'_E)$ where $V' = V - \{v_1, v_2\}$, $E' = (E - \{e_{\text{cut}}\}) \cup \{e^1_{\text{cut}}, e^2_{\text{cut}}\}$ (we duplicated the cut e_{cut}) such that $\mathtt{in}'(e^1_{\text{cut}}) = (\overleftarrow{e_1}, \overleftarrow{e_2})$, $\mathtt{in}'(e^2_{\text{cut}}) = (\overrightarrow{e_1}, \overrightarrow{e_2})$ with $\mathtt{in}'(x) = \mathtt{in}(x)$ otherwise and $\mathtt{out}'(x) = \mathtt{out}(x)$.

We have that $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ corresponds to $\Phi^{\mathtt{comp}}_{\mathcal{R}'} + [-v_1(X), -v_2(X)]$. The constellation $\Phi^{\mathtt{comp}}_{\mathcal{R}'}$ contains rays $+v_1(\mathtt{l} \cdot t_i)$, $+v_1(\mathtt{r} \cdot u_i)$, $+v_2(\mathtt{l} \cdot v_i)$ and $+v_2(\mathtt{r} \cdot w_i)$ respectively coming from stars $\phi^1_i := [+v_1(\mathtt{l} \cdot t_i), r^1_i]$, $\phi^2_i := [+v_1(\mathtt{r} \cdot u_i), r^2_i]$, $\phi^3_i := [+v_2(\mathtt{l} \cdot v_i), r^3_i]$ and $\phi^4_i := [+v_1(\mathtt{r} \cdot t_i), r^4_i]$ (by definition of proof-structure). The constellation $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ obtained after cut-elimination should be $\Phi^{\mathtt{comp}}_{\mathcal{R}'}$ in which all terms $+v_1(\mathtt{l} \cdot t_i)$, $+v_1(\mathtt{r} \cdot u_i)$, $+v_2(\mathtt{l} \cdot v_i)$, $+v_2(\mathtt{r} \cdot w_i)$ are respectively replaced by $+\overleftarrow{v_1}(t)$, $+\overrightarrow{v_1}(t)$, $+\overleftarrow{v_2}(t)$ and $+\overrightarrow{v_2}(t)$. We also have the cuts $[-\overleftarrow{v_1}(X), -\overleftarrow{v_2}(X)] + [-\overrightarrow{v_1}(X), -\overrightarrow{v_2}(X)]$. Because of this replacement of function symbol, we cannot expect to have exactly $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ after executing $\Phi^{\mathtt{comp}}_{\mathcal{R}}$. We can only expect structural equivalence by $\simeq_S$. We hence show that the diagrams of $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ and those of $\Phi^{\mathtt{comp}}_{\mathcal{S}}$ are equal up to a change of symbols.

We reason on the diagrams of $\Phi^{\mathtt{comp}}_{\mathcal{R}}$ which contain the cut $[-v_1(X), -v_2(X)]$ and the stars ϕ^k_i for $i \in \{1, 2, 3, 4\}$. It is possible to construct two diagrams (G^1_δ, δ_1) and (G^2_δ, δ_2) defined by $G^1_\delta := (V_1, E_1, \mathtt{end}_1)$ and $G^2_\delta := (V_2, E_2, \mathtt{end}_2)$ with:

- some $u, u' \in V_1$ with $\delta(u) = \delta(u') = [-v_1(X), -v_2(X)]$, some $u_i^1, u_i^3 \in V_1$ with $\delta(u_i^k)$ corresponding to ϕ_i^k and edges $e_i^1, e_i^3 \in E_1$ with $\mathtt{end}_1(e_i^1) = \{u, e_i^1\}$ and $\mathtt{end}_1(e_i^3) = \{u', e_i^3\}$.
- some $u, u' \in V_2$ with $\delta(u) = \delta(u') = [-v_1(X), -v_2(X)]$, some $u_i^2, u_i^4 \in V_1$ with $\delta(u_i^k)$ corresponding to ϕ_i^k and edges $e_i^2, e_i^4 \in E_2$ with $\mathtt{end}_2(e_i^2) = \{u, e_i^2\}$ and $\mathtt{end}_2(e_i^4) = \{u', e_i^4\}$.

Notice that we duplicated the cut into two vertices u and u' in order to satisfy the given constraints of term unification. The homomorphisms δ_1 and δ_2 are defined in the unique possible way. Those diagrams can be reunited or not depending on the other rays in $\mathcal{R}'$. We do not even need to do diagram contractions. We update $\delta(u)$ and $\delta(u')$ in both diagrams so that $\delta(u)$ and $\delta(u')$ respectively correspond to $[-\overleftarrow{v_1}(X), -\overleftarrow{v_2}(X)]$ and $[-\overrightarrow{v_1}(X), -\overrightarrow{v_2}(X)]$. In order to preserve the links induced by edges, we must replaced $+v_1(\mathtt{l} \cdot t_i)$, $+v_1(\mathtt{r} \cdot u_i)$, $+v_2(\mathtt{l} \cdot v_i)$, $+v_2(\mathtt{r} \cdot w_i)$ by $+\overleftarrow{v_1}(t)$, $+\overrightarrow{v_1}(t)$, $+\overleftarrow{v_2}(t)$ and $+\overrightarrow{v_2}(t)$ respectively. We obtain exactly diagrams which could be constructed in $\Phi_{\mathcal{S}}^{\mathtt{comp}}$. All other edges must come from $\Phi_{\mathcal{R}'}^{\mathtt{comp}}$. It follows that all extensions of (G_δ^k, δ_k) with $k \in \{1, 2\}$ or other diagrams not relating the edges and vertices of $\mathcal{R}$ previously mentioned must be constructed with edges of $\Phi_{\mathcal{R}'}^{\mathtt{comp}}$ in both $\Phi_{\mathcal{R}}^{\mathtt{comp}}$ and $\Phi_{\mathcal{S}}^{\mathtt{comp}}$. It follows that they both have the same diagrams and hence the same correct and saturated diagrams (up to renaming of target stars for δ_1 and δ_2 while preserving connexions between rays). We finally have $\Downarrow \mathtt{CSatDiags}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) \simeq \Downarrow \mathtt{CSatDiags}(\Phi_{\mathcal{S}}^{\mathtt{comp}})$.

□

§67.10 **Theorem** (Simulation of reduction for proof-nets)**.** For an MLL+MIX proof-net $\mathcal{R}$ such that $\mathcal{R} \rightsquigarrow^* \mathcal{S}$ with $\mathcal{S}$ in normal form, we have $\mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) \simeq_S \Phi_{\mathcal{S}}^{\mathtt{ax}}$.

Proof. By induction on the number n of steps of cut-elimination.

◇ **Base case** Assume $n = 0$. It means that $\mathcal{R}$ is already in normal form ($\mathcal{R} \rightsquigarrow_0 \mathcal{R}$). Hence it has no cut and we trivially have $\Phi_{\mathcal{R}}^{\mathtt{comp}} = \Phi_{\mathcal{R}}^{\mathtt{ax}}$, hence $\mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) = \mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{ax}}) = \Phi_{\mathcal{R}}^{\mathtt{ax}}$.

◇ **Induction case** Assume $n = n' + 1$. By induction hypothesis, if $\mathcal{R} \rightsquigarrow^{n'} \mathcal{S}$ then $\mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) = \Phi_{\mathcal{S}}^{\mathtt{ax}}$. We have to show that if we have $\mathcal{R}_0 \rightsquigarrow \mathcal{R} \rightsquigarrow^{n'} \mathcal{S}$ then $\mathtt{AEx}(\Phi_{\mathcal{R}_0}^{\mathtt{comp}}) = \Phi_{\mathcal{S}}^{\mathtt{ax}}$. By Lemma 67.9 and $\mathcal{R}_0 \rightsquigarrow \mathcal{R}$, we have $\mathtt{AEx}(\Phi_{\mathcal{R}_0}^{\mathtt{comp}}) \simeq_S \mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}})$. By $\mathcal{R} \rightsquigarrow^{n'} \mathcal{S}$ and the induction hypothesis, we have $\mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) = \Phi_{\mathcal{S}}^{\mathtt{ax}}$. By transitivity of equality (up to the structural equivalence $\simeq_S$), we have $\mathtt{AEx}(\Phi_{\mathcal{R}_0}^{\mathtt{comp}}) \simeq_S \mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) = \Phi_{\mathcal{S}}^{\mathtt{ax}}$.

□

68 Simulation of Danos-Regnier correctness test

§68.1 The simulation of Danos-Regnier correctness is a direct translation of the approach of proofs as partitions of a set (*cf.* Section 37). Danos-Regnier tests (correctness hypergraphs without axioms) are translated by a constellation which simply reproduces the hypergraph structure of the test. In particular, it has no dynamics and is just designed to be plugged to a vehicle. For instance, the 3-ary tensor link relating two inputs u, v to one output w becomes a 3-ary star $[-u(X), -v(X), +w(X)]$. We translate conclusions of the whole proof-structure by uncoloured rays.

§68.2 To make tests more readable, they will sometimes be written as *blocks* with inputs (rays of polarity $-$) above and outputs below (rays of polarity $+$). For instance, $[-u(X), -v(X), +w(X)]$ becomes $\begin{bmatrix} -u(X),\ -v(X) \\ +w(X) \end{bmatrix}$. Links are then connected like tiles or bricks, as in sequent calculus.

§68.3 **Definition** (MLL test)**.** Let $\mathcal{S}$ be a proof-structure and φ one of its switchings. The *test* associated with $\mathcal{S}^\varphi$ is the constellation defined by

$$\Phi_{\mathcal{S}}^{\varphi} := \Phi_{\mathcal{S}}^{\mathrm{cut}} \uplus \sum_{v \in V^{\mathcal{S}\varphi}} v^{\bigstar}.$$

where the translation $v^{\bigstar}$ of a vertex v conclusion of an hyperedge e is defined as follows:

- if $\ell_E(e) = \mathrm{ax}$ then $v^{\bigstar} = \begin{bmatrix} -\mathtt{addr}_{\mathcal{S}}(v) \\ +v(X) \end{bmatrix}$;
- if $\ell_E(e) = \parr_L$ and $\mathtt{in}(e) = (u, w)$ then $v^{\bigstar} = \begin{bmatrix} -u(X) \\ +v(X) \end{bmatrix} + \begin{bmatrix} -w(X) \\ \end{bmatrix}$;
- if $\ell_E(e) = \parr_R$ and $\mathtt{in}(e) = (u, w)$ then $v^{\bigstar} = \begin{bmatrix} -u(X) \\ \end{bmatrix} + \begin{bmatrix} -w(X) \\ +v(X) \end{bmatrix}$;
- if $\ell_E(e) = \otimes$ and $\mathtt{in}(e) = (u, w)$ then $v^{\bigstar} = \begin{bmatrix} -u(X),\ -w(X) \\ +v(X) \end{bmatrix}$;
- if $v \in \mathtt{Concl}(\mathcal{S})$ then $v^{\bigstar} = \begin{bmatrix} -v(X) \\ v(X) \end{bmatrix}$.

§68.4 Tests for a proof-structure $\mathcal{S}$ are actually designed so that $\mathfrak{D}[\Phi_{\mathcal{S}}^{\varphi}]$ is isomorphic to $\mathcal{S}^\varphi$ without axioms, as illustrated in Figure 68.1. The point is that when constructing the union of a vehicle with a test, we obtain a constellation with a dependency graph structurally corresponding to a Danos-Regnier correctness hypergraph (*cf.* Definition 30.16). However, there is a minor technical problem. Imagine that we have a

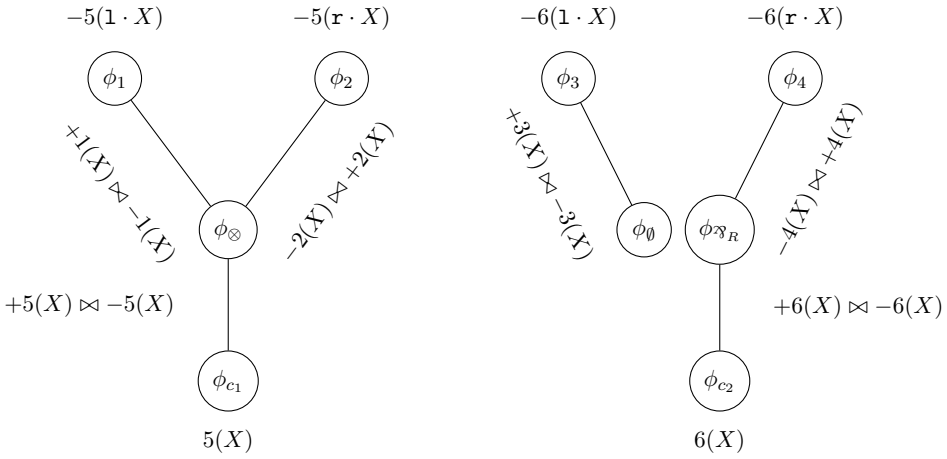


Figure 68.1: Dependency graph of the constellation corresponding to Switching 2 in Figure 30.9.

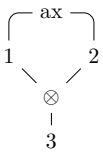
vehicle $[+3(\mathtt{l}\cdot X), +3(\mathtt{r}\cdot X)]$ corresponding to a $\parr$ on two atoms, and a test corresponding to a switching $\parr_L$ with $[-3(\mathtt{l}\cdot X), +1(X)] + [-3(\mathtt{r}\cdot X), +2(X)] + [-1(X), +3(X)] + [-2(X)] + [-3(X), 3(X)]$. The problem is that the conclusion $[-3(X), 3(X)]$ can technically interfere and connect with the vehicle. We only want the stars $[-3(\mathtt{l}\cdot X), +1(X)]$ and $[-3(\mathtt{r}\cdot X), +2(X)]$ to connect with atoms of the vehicle.

§68.5 Girard's solution is to use different colours to distinguish terms coming from the vehicle and terms coming from tests. We can do that by wrapping addresses $+u(t)$ and $-u(t)$ with a colour $+v$ and $-v$ to obtain $+v(+u(t))$ and $-v(-u(t))$. With these wrapped addresses, the other rays of the test cannot interact with the vehicle.

§68.6 I find this solution too cumbersome to write so I choose to *pre-execute* tests to obtain compact tests exactly translating partitions over the set of atoms, as in Section 37. Such compact tests are made of stars for each connected components linking inputs for atoms to unpolarised outputs corresponding to conclusions. In other words, the internal structure of tests does not matter, what is important is only the *visible interface* of tests and their organisation in terms of reunion/separation. For instance, the test of Figure 68.1 simply becomes:

$$\begin{bmatrix} -5(\mathtt{l}\cdot X) & -5(\mathtt{r}\cdot X) \\ & 5(X) \end{bmatrix} + \begin{bmatrix} -6(\mathtt{l}\cdot X) \\ \end{bmatrix} + \begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}.$$

§68.7 The idea of the simulation of logical correctness is that we would like to make tests interact with a fully positively polarised vehicle to reproduce a Danos-Regnier correctness hypergraph (more precisely, the interaction between two partitions as in Section 37). In stellar resolution, testing is done with execution and we would like to obtain $[v_1(X), ..., v_n(X)]$ with $\mathtt{Concl}(\mathcal{S}) = \{v_1, ..., v_n\}$, ensuring all conclusions can be reached



$$[+3(\mathtt{l}\cdot X),+3(\mathtt{r}\cdot X)]+[-3(\mathtt{l}\cdot X),+1(X)]+[-3(\mathtt{r}\cdot X),+2(X)]$$
$$[-1(X),-2(X),+3(X)]+[-3(X),3(X)]$$

Figure 68.2: Incorrect correctness hypergraph for a proof-structure $\mathcal{S}$ and its translation. Notice that the cycle is turned into a computational cycle (a loop in a program).

(several connected components induce several stars in the normal form) only once (cycles are designed to produce several $v_i(X)$), hence the associated correctness hypergraph $\Phi^{\mathcal{O}}_{\mathcal{S}}$ is connected and acyclic as required by correctness criteria for MLL.

§68.8 **Example.** If we consider the test:

$$\begin{bmatrix} -5(\mathtt{l}\cdot X) & -5(\mathtt{r}\cdot X) \\ & 5(X) \end{bmatrix}+\begin{bmatrix} -6(\mathtt{l}\cdot X) \\ \end{bmatrix}+\begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}$$

of Figure 68.1, we can plug it by union of constellation with the vehicle $[+5(\mathtt{l}\cdot X),+6(\mathtt{l}\cdot X)]+[+5(\mathtt{r}\cdot X),+6(\mathtt{r}\cdot X)]$ and obtain:

$$[+5(\mathtt{l}\cdot X),+6(\mathtt{l}\cdot X)]+[+5(\mathtt{r}\cdot X),+6(\mathtt{r}\cdot X)]+$$
$$\begin{bmatrix} -5(\mathtt{l}\cdot X) & -5(\mathtt{r}\cdot X) \\ & 5(X) \end{bmatrix}+\begin{bmatrix} -6(\mathtt{l}\cdot X) \\ \end{bmatrix}+\begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}.$$

The first star $[+5(\mathtt{l}\cdot X),+6(\mathtt{l}\cdot X)]$ reunites the two first blocks and we obtain:

$$[+5(\mathtt{r}\cdot X),+6(\mathtt{r}\cdot X)]+\begin{bmatrix} -5(\mathtt{r}\cdot X) \\ 5(X) \end{bmatrix}+\begin{bmatrix} -6(\mathtt{r}\cdot X) \\ 6(X) \end{bmatrix}.$$

The star $[+5(\mathtt{r}\cdot X),+6(\mathtt{r}\cdot X)]$ of the vehicle reunites the two last blocks of the test and we obtain $[5(X),6(X)]$. If the vehicle is "too small" or "too big", it leaves unpolarised ray either in the test or in the vehicle. If we had a star $[+5(\mathtt{l}\cdot X),+5(\mathtt{r}\cdot X)]$ in the vehicle, it would form a cycle with the first block. This cycle yields infinitely many stars $[5(X)]+[5(X)]+\ldots$ by unfolding the cycle to construct as many cyclic diagrams as we wish.

§68.9 **The need for visible incorrectness.** Before expressing logical correctness in stellar resolution, I would like to mention an interesting error in Girard's original definition. In his original paper [Gir17, Section 2.3], Girard forbids cyclic diagrams and the empty star. However, if we accept his definition, cyclic proof-structures $\mathcal{S}$ will be reduced into the empty constellation $\emptyset$. But if we put such proof-structure next to a correct proof-structure $\mathcal{R}$, we have that $\mathcal{S}\uplus\mathcal{R}$ is correct. This not what we want.

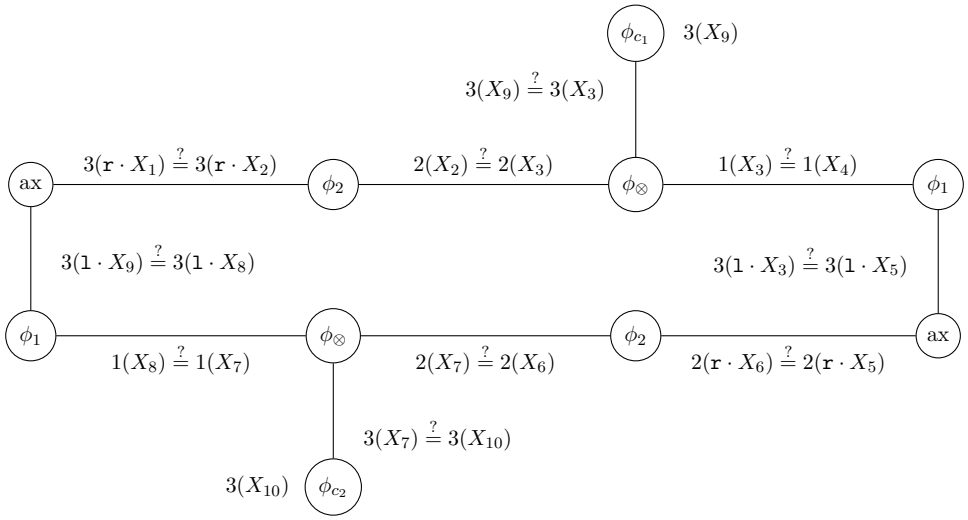


Figure 68.3: Example of a correct and saturated cyclic diagram for the constellation from Figure 68.2 actualising into $[3(X_9), 3(X_{10})]$. The cycle can be extended infinitely many times by adding copies of three stars of the constellation.

§68.10 I give more details about this problem. We write $\mathtt{AEx}[\mathtt{RT}]$ for the execution restricted to tree-shaped diagrams used by Girard which applies the operator $\lightning$ and $\flat$. If we consider the correctness hypergraph of Figure 68.2, infinitely many diagrams can be constructed because of the loop in its dependency graph (an example of loop unfolding is given in Figure 68.3) but all the corresponding diagrams have free polarised rays. Such diagrams are erased by the operator $\lightning$. Therefore, $\mathtt{AEx}[\mathtt{RT}](\Phi^{\mathrm{ax}}_{\mathcal{S}} \uplus \Phi^{\varphi}_{\mathcal{S}}) = \emptyset$.

§68.11 The solution adopted in this thesis is to make incorrect proofs visible in the normal form. Our definition of execution makes this possible. Cycles yield cyclic diagrams, which are accepted. Since proof-structures are always translated into constellations with trivial equations, such diagrams can always be extended into cyclic diagrams which will be evaluated into infinitely many stars (as in Figure 68.3). This makes incorrectness visible in the output of execution.

§68.12 This problem is actually not new and already existed in previous GoI models. For instance, in Seiller's works, it was necessary to be able to detect cycles. The problem has been solved with a notion of *wager* which is a value associated with proofs indicating the presence of cycles but we were able to simulate this idea by modifying the notion of execution instead.

§68.13 We can now finally state the Danos-Regnier correctness criterion in stellar resolution.

§68.14 **Proposition.** If a connected multiplicative correctness hypergraph $\mathcal{S}^{\varphi}$ has no conclusion then it is cyclic.

Proof. Since $\mathcal{S}^\varphi$ has no conclusion, all vertices are source of exactly one hyperedge. By the definition of proof-structure, all vertices are target of exactly one hyperedge. Hence, all vertices have a degree at least 2. Assume there is no cycle. Since the hypergraph is connected, it must be tree-shaped. It follows that there is at least one leaf, *i.e.* a vertex of degree 1 which is incident to only one edge. However, this contradicts the fact that all vertices have a degree at least 2. □

§68.15 **Definition** (Full head polarisation)**.** Let Φ be a constellation defined in a coloured signature $(V, F, \mathtt{ar}, \supset, \lfloor\cdot\rfloor)$. Its *full head polarisation* is a constellation $\overset{+}{\Phi}$ defined by $I_{\overset{+}{\Phi}} := I_\Phi$, $\overset{+}{\Phi}[i] := \Phi[i]$ and:

- $\Phi[i][j] = f(r_1, ..., r_k)$ then, $\overset{+}{\Phi}[i][j] := +f(r_1, ..., r_k)$;
- $\Phi[i][j] = -c(r_1, ..., r_k)$ then, $\overset{+}{\Phi}[i][j] := +c(r_1, ..., r_k)$;
- $\Phi[i][j] = +c(r_1, ..., r_k)$ then, $\overset{+}{\Phi}[i][j] := +c(r_1, ..., r_k)$

where $F = F_0 \uplus F_- \uplus F_+$, $\{f, c\} \subseteq F_0$, $-c \in F_-$ and $+c \in F_+$.

§68.16 **Definition.** If $\Phi := [1(X), +2(X)] + [+3(X), +4(X)]$ then $\overset{+}{\Phi} := [+1(X), +2(X)] + [+3(X), +4(X)]$.

§68.17 **Lemma** (Structural equivalence of correctness hypergraphs)**.** Let $\mathcal{S}^\varphi := (V, E, \mathtt{end}, \ell)$ be a correctness hypergraph for a switching φ and let

$$\mathfrak{D}[\overset{+}{\Phi}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^\varphi_{\mathcal{S}})] := (V_{\mathfrak{D}}, E_{\mathfrak{D}}, \mathtt{end}_{\mathfrak{D}})$$

be the dependency graph of its translation using Definition 69.15. There is a bijection $\rho : V \to V_{\mathfrak{D}}$ preserving adjacency.

Proof. First, a way to understand the connexions in $\mathfrak{D}[\overset{+}{\Phi}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^\varphi_{\mathcal{S}})]$ is that they have the same connexions as $\mathfrak{D}[\overset{+}{\Phi}^{\mathrm{ax}}_{\mathcal{S}} \uplus \Phi^\varphi_{\mathcal{S}}]$ except that rays of $\overset{+}{\Phi}^{\mathrm{ax}}_{\mathcal{S}}$ cannot be connected to rays of $\Phi^\varphi_{\mathcal{S}}$ except for the rays $-\mathtt{addr}_{\mathcal{S}}(v)$.

Let $v \in V$ be a vertex representing a formula. Assume it is a conclusion of an hyperedge $e \in E$ (it is always the case by definition of proof-structure). If $\ell(e) \in \{\mathrm{ax}, \otimes\}$ or $e \in \mathtt{Concl}(\mathcal{S})$ then $\rho(v) := v^\bigstar$ (*cf.* Definition 69.15). If $\ell(e) = \parr_L$ (*resp.* $\ell(e) = \parr_R$) then $\rho(v)$ is the left (*resp.* right) star of $v^\bigstar$. Assume we have v and v' adjacent. A case analysis of all possible situations of adjacency shows that ρ preserves adjacency. Moreover, ρ associates vertices in a unique way. Without loss of generality, we consider the case of $v \in V$ conclusion of a hyperedge $e \in E$ with $\ell(e) = \otimes$ with a premise $v' \in V$ conclusion of

an axiom. Hence we have $\mathtt{end}(e) = \{v, v', w\}$ for some $w \in V$. The vertices v and v' will be translated into some star $[-\mathtt{addr}_{\mathcal{S}}(v'), +v'(X)]$ and $[-v'(X), -w(X), +v(X)]$. The two stars can be linked along the rays $+v'(X)$ and $-v'(X)$ making them adjacent. □

§68.18 **Corollary.** Let $\mathcal{S}^{\varphi}$ be a correctness hypergraph for a switching φ and let $\mathfrak{D}[\Phi^{\varphi}_{\mathcal{S}}]$ be the dependency graph of its translation by Definition 69.15.

We have that $\mathcal{S}^{\varphi}$ is connected or acyclic if and only $\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})$ is.

Proof. By Lemma 68.17, since we have a bijection ρ preserving adjacency, we also preserve reachability and paths in the test. It follows that the set of cycles and connected components in $\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})$ is isomorphic to the cycles and connected components of $\mathfrak{D}[\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})]$. □

§68.19 **Theorem** (Stellar correctness criterion)**.** A proof-structure $\mathcal{S}$ such that $\mathtt{Concl}(\mathcal{S}) = \{v_1, ..., v_n\}$ is MLL-certifiable (*cf.* Definition 30.3) if and only if for all switchings φ, we have:

$$\mathtt{AEx}(\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})) = [v_1(X), ..., v_n(X)].$$

Proof. We show two implications.

◇ ($\Rightarrow$) Assume $\mathcal{S}$ is MLL-certifiable. Then there exists a vertex labelling ℓ making $\mathcal{S}$ an MLL proof-net. By the Danos-Regnier correctness criterion (*cf.* Theorem 30.17), it means that for all switching φ of $\mathcal{S}$, we have a correctness hypergraph $\mathcal{S}^{\varphi}$ which is connected and acyclic. By Corollary 68.18, $\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})$ must be connected and acyclic as well. In Definition 69.15, we see that all connexions in tests are between rays of identical underlying terms and that all rays are deterministic, making them perfect in the sense of Definition 62.15. It follows that $\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})$ is perfect. By the perfection lemma (*cf.* Lemma 62.16), we have a unique diagram. It is then sufficient to use a compression strategy as in Definition 63.2, *i.e.* we directly apply fusion between stars of the constellation. By pre-execution of $\Phi^{\varphi}_{\mathcal{S}}$ we obtain a compact test translating the partition interpretation of Section 37. What axioms and cuts do is simply merging some blocks of the test together. The only rays left in the actualisation of the only diagram of $\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})$ are the unpolarised conclusions of the test, that is $v_1(X), ..., v_n(X)$. We obtain the normal form $[v_1(X), ..., v_n(X)]$.

◇ ($\Leftarrow$) Assume $\mathtt{AEx}(\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{AEx}(\Phi^{\varphi}_{\mathcal{S}})) = [v_1(X), ..., v_n(X)]$. Assume by contradiction that $\mathcal{S}^{\varphi}$ has at least two connected components. Assume that a component has no conclusion (because of cuts). Then, by Proposition 68.14, there is a cycle yielding infinitely many closed diagrams normalising into the empty star []. Hence, all

connected components must have free rays corresponding to conclusions. By the independence of connected component (*cf.* Lemma 62.17), we can independently execute each connected component. Since they correspond to deterministic and exact subconstellations, the normalisation produces the constellation $\phi_1+...+\phi_k$ for the k connected components. It contradicts the hypothesis that we normalise into a single star. Therefore, $\mathcal{S}^\varphi$ must be connected. Now, assume by contradiction that $\mathcal{S}^\varphi$ is cyclic. The cycle can either yield a closed diagram actualising into the empty star [] or pass through a conclusion and produce infinitely many stars containing conclusion rays. In both case, the normalisation is different from $[v_1(X), ..., v_n(X)]$, contradicting the hypothesis. Therefore, $\mathcal{S}^\varphi$ must also be acyclic. This proves that $\mathcal{S}^\varphi$ must be a tree for any switching φ, *i.e.* that $\mathcal{S}$ is MLL-certifiable.

□

§68.20 The following corollary extends the logical correctness to MLL+MIX and suggests a more general variant which also captures MLL.

§68.21 **Corollary.** Let $\mathcal{S}$ be a proof-structure and $\Phi := \overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{Ex}(\Phi^\varphi_{\mathcal{S}})$ be the constellation corresponding to the correctness hypergraph $\mathcal{S}^\varphi$ for some switching φ. We have:

- $\mathcal{S}^\varphi$ is acyclic $\Leftrightarrow$ $\mathfrak{D}[\Phi]$ is acyclic $\Leftrightarrow$ $|\mathtt{Ex}(\Phi)| < \infty$;
- $\mathcal{S}^\varphi$ is connected and acyclic $\Leftrightarrow$ $\mathfrak{D}[\Phi]$ is a deterministic tree $\Leftrightarrow$ $|\mathtt{Ex}(\Phi)| = 1$;
- $\mathcal{S}^\varphi$ is connected and acyclic $\Leftrightarrow$ Φ normalises into the star of its uncoloured rays.

Proof. The first equivalence of each point are direct consequences of Corollary 68.18. It only remains to show the last equivalences.

- If $\mathfrak{D}[\Phi]$ is acyclic, then by Corollary 62.13, $|\mathtt{Ex}(\Phi)| < \infty$ because Φ is finite. Now assume that $|\mathtt{Ex}(\Phi)| < \infty$. The proof of Theorem 68.19 shows that the presence of cycles in correctness hypergraphs is linked to the generation of infinitely many correct saturated diagrams. Hence it cannot be both cyclic and strongly normalising and has to be acyclic.
- We start from the previous point and consider the additional property of connectedness. If $\mathfrak{D}[\Phi]$ is also connected, then by perfection of Φ (*cf.* Definition 62.15), there is a single star in the normal form. Conversely, if Φ normalises into a single star, its dependency graph must be both connected and acyclic, otherwise we would end up with either several stars (*cf.* Theorem 68.19) or infinitely many correct saturated diagrams (since cycles are related to non-termination as stated in the previous point).
- The third case corresponds to an alternative characterisation of correct proof-structures. Assume $\mathcal{S}^\varphi$ is connected and acyclic. Then $\mathfrak{D}[\Phi]$ is a deterministic tree by the previous point. By definition, uncoloured rays are the only free rays in Φ.

Since Φ is exact, it must produce a unique diagram corresponding to the cover tree of $\mathfrak{D}[\Phi]$. By definition, such a diagram reduces into the star of its free rays, hence the star of its uncoloured rays. Now, assume Φ normalises into the star of its uncoloured rays. The reasoning is the same as for the previous point.

□

§68.22 **The problem of tests with cuts.** There is a problem with cuts. Imagine that we have the proof-structure of Figure 66.2. Its executed test for $\parr_L$ is the constellation:

$$\left[\begin{array}{c} -7(\mathtt{l}\cdot X), -8(\mathtt{l}\cdot X), -8(\mathtt{r}\cdot X) \\ \end{array}\right] + \left[\begin{array}{c} -7(\mathtt{l}\cdot X) \\ \end{array}\right] + \left[\begin{array}{c} -3(X) \\ 3(X) \end{array}\right] + \left[\begin{array}{c} -6(X) \\ 6(X) \end{array}\right].$$

But if we execute the constellation corresponding to this proof-structure in order to simulate cut-elimination, we obtain a constellation $[3(X), 6(X)]$ representing the normal form after cut-elimination. When fully polarised with $[+3(X), +6(X)]$ in order to test it for MLL correctness, it cannot even pass the test above. Hence, the fact of passing a test is not preserved by reduction as if we were reducing a λ-term t into u, could assert $t : A$ but not $u : A$ (no preservation of typing, also called "subject reduction").

§68.23 Conversely, imagine that we have the following tests for axioms:

$$\left[\begin{array}{c} +3(X) \\ 3(X) \end{array}\right] + \left[\begin{array}{c} +6(X) \\ 6(X) \end{array}\right].$$

It works for $[3(X), 6(X)]$ but not for the vehicle of the proof of *Figure* 66.2. As if a test for $\vdash A, A^\perp$ could not validate proof-structures containing cuts. It is like having to evaluate a λ-term of type $\vdash A \multimap A$ in order to even typecheck it and say that it behaves like an identity function. If the term is very big then it makes no sense to do that to simply associate a type. Imagine pushing a car to its limits in order to say that it can work as expected and that it is ready to be sold.

§68.24 However, this problem should not be seen as a limitation or a design defect. Consistently with the philosophy of transcendental syntax and the distinction between Usine and Usage, correctness tests should be applied on *cut-free* proof-structures only. It is only after that we can guarantee with more or less certainty that our cut-free objects will interact correctly by cut.

§68.25 **Separating program (tested) and specification (test).** A remark that I got several times is "you are trying to relate correctness criteria with program testing but your test is actually dependent of the program (vehicle)". Yes, this is true. If you look at the definition of test in Definition 68.3, the translation of atoms in the test uses addresses of atoms in axioms. It is as if we were designing specific specifications for each programs instead of generic ones. Actually, the problem is that program (vehicle) and test are *intertwined* in proof-structures (as explained in Section 44). It is possible to define

tests directly from a sequent (corresponding to a specification), hence independently of a proof-structure (program) as we will see in the next section (Usine interpretation).

§68.26 **Finiteness of correctness checking.** Despite all the previous "marketing" of finiteness of reasoning in Chapter 6, we can see that in our definitions, cut-elimination and correctness checking can loop if we try to implement it with concrete or interactive execution. Loops in proof-structures always involve equations between terms containing common variables. For instance, $\phi := [+1(X), +2(X)] + [-1(X), -2(X), 3(X)]$ can be reduced to $\phi' := [+2(X), -2(X), 3(X)]$. In ϕ, the connexion between $+1(X)$ and $-1(X)$ makes variables distinct (by α-unification). However, in ϕ', the connexion between $+2(X)$ and $-2(X)$ involve exactly the same variable because they are part of the same star. It means that loops can be detected by looking for equations involving exactly the same variables. It is up to us to make such diagrams incorrect. We can also, like Girard, choose to consider only tree-shaped diagrams but Thomas Seiller and I wanted something more flexible (also because it is computationally more expressive as illustrated in Chapter 8).

69 Construction of multiplicative formulas

§69.1 As explained in Section 43, formulas should be defined as sets of constellations *w.r.t.* an orthogonality relation between constellations. In this section, I present the Usine and Usage interpretation of formulas in the case of stellar resolution.

§69.2 Corollary 68.21 suggests three orthogonality relations we call $\perp_{\text{fin}}$, $\perp^1$ and $\perp^R$ but others can be designed depending on what we want.

§69.3 **Convention.** In this section, we write $\texttt{Ex}$ for $\texttt{AEx}$.

§69.4 **Definition** (Orthogonality)**.** We define binary relations of *orthogonality* between two constellations Φ_1 and Φ_2 *w.r.t.* a set of colours $C \subseteq F_+ \uplus F_-$:

- $\Phi_1 \perp_C^{\text{fin}} \Phi_2$ when $|\texttt{Ex}_C(\Phi_1 \uplus \Phi_2)| < \infty$;
- $\Phi_1 \perp_C^1 \Phi_2$ when $|\texttt{AEx}_C(\Phi_1 \uplus \Phi_2)| = 1$;
- $\Phi_1 \perp_C^R \Phi_2$ when $\texttt{Ex}_C(\Phi_1 \uplus \Phi_2) = \{\texttt{Roots}(\Phi_1 \uplus \Phi_2)\}$ where $\texttt{Roots}(\Phi)$ is the star of uncoloured rays in Φ.

The orthogonal of a set of constellations $\mathbf{A}$ is defined by:

$$\mathbf{A}^{\perp_C} := \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp_C \Phi'\}$$

for a relation of orthogonality $\perp$.

§69.5 The orthogonality $\perp^R$ will be our favourite since it is very close to the correctness criterion used in stellar resolution. In particular, it forces a full connexion between vehicle and test, as in the proof-as-partitions approach. The other criteria are more lax. In particular, it is possible to have constellations which do not correspond to proof-structures in the orthogonal of tests. It can however be seen as a feature as well[1].

§69.6 In order to allow typing for partial evaluations, the orthogonality relation $\perp_C$ has to be parametrised by a set of colours C but we omit this parameter when considering all colours in F.

§69.7 The orthogonality $\perp^{\text{fin}}$ will define a fully complete model of MLL+MIX, while $\perp^1$ and $\perp^R$ (which captures more directly the correctness criterion for MLL) will define a fully complete model of MLL. However, those notions of orthogonality share most of the properties needed, and we therefore use the generic notation $\perp$ in the following to state results valid for all of them.

§69.8 **Lemma** (Invariance of orthogonality under execution)**.** Let Φ and Φ' be constellations such that $\Cap_C(\Phi, \Phi') = \emptyset$ for a set of colours $C \subseteq F_+ \uplus F_-$. We have $\Phi \perp_C \Phi'$ if and only if $\mathtt{Ex}_C(\Phi) \perp_C \Phi'$ for $\perp_C \in \{\perp_C^1, \perp_C^{\text{fin}}, \perp_C^R\}$.

Proof. These relations are satisfied when $P(\mathtt{Ex}_C(\Phi \uplus \Phi'))$ is satisfied for some property P. By the lemma of partial pre-execution (*cf.* Lemma 64.9), we have $\mathtt{Ex}_C(\mathtt{Ex}_C(\Phi) \uplus \Phi') = \mathtt{Ex}_C(\Phi \uplus \Phi')$. Hence we have $P(\mathtt{Ex}_C(\Phi \uplus \Phi'))$ if and only if $P(\mathtt{Ex}_C(\mathtt{Ex}_C(\Phi) \uplus \Phi'))$, meaning that we have $\Phi \perp_C \Phi'$ if and only if $\mathtt{Ex}_C(\Phi) \perp \Phi'$. □

§69.9 **Remark.** This does not invalidate the problem presented in Paragraph 68.22 despite the appearance because if we had cuts in Φ, then both cuts and a test Φ' would want to connect with rays of Φ. This contradicts the precondition $\Cap_C(\Phi, \Phi') = \emptyset$.

Usine interpretation

§69.10 In this section, we construct formulas by generalising logical correctness. In the Usine interpretation, we are interested in effective verification. I refer to Section 43 for the philosophy of Usine.

§69.11 **Definition** (Type label)**.** A *type (label)* is an object A associated to a finite set of constellations $\mathtt{Tests}(A)$ called its *tests*. We say that a constellation Φ is of type A *w.r.t.* $\perp$ if and only if $\Phi \in \mathtt{Tests}(A)^{\perp}$.

[1] I believe this is the opinion of Thomas Seiller.

§69.12 Type labels appear in model checking [BK08]: given an automata Φ (or labelled transition system), we would like to know whether it satisfies a specification S (often written as a formula of a logic called LTL or CTL). It is then possible to check if Φ satisfies S by turning $\neg S$ into an automaton $\Phi_{\neg S}$ and verifying if $\mathcal{L}(\Phi) \cap \mathcal{L}(\Phi_{\neg S}) = \emptyset$, by analysing paths of the state graph of the automaton [HR04, Section 3.6.3]. This is similar to how we turn a sequent $\vdash \Gamma$ into a set of tests (defined as constellations) allowing us to label/certify a constellation as a proof of A. Moreover, the Danos-Regnier's tests can also be considered as proofs of $A^{\perp}$.

§69.13 The purpose of having finite set of tests is to make type checking computable. However, this only happens under some conditions such as the orthogonality relation being decidable. Even under these conditions, it is possible to "trick" tests so to create infinite loops and make effective type checking impossible. It shows that we need to consider testing *w.r.t.* a specific class of objects (for instance the universe of proof-structures) so to prevent such tricks to happen, consistently with the idea of *epidictics* explained in Section 44. In this restricted case, orthogonality is decidable as we can check for the presence of cycles in dependency graphs in finite time.

§69.14 The definition of orthogonality and interactive testing leads to a reformulation of correctness criterion, showing that MLL sequents define type labels by themselves, independently of a proof-structure. This is based on the fact that the bottom part of proof-structure corresponds to the syntax tree of a sequent which is already a sort of pre-typing constraining cut-elimination. By constructing a syntax hypergraph from a sequent, Definition 68.3 can be used.

§69.15 **Definition** (Test of a sequent)**.** Let $\vdash \Gamma$ be a sequent of MLL where $\Gamma \subseteq \mathcal{F}_{\mathrm{MLL}}$ and all variables are distinct. We define the syntax tree of an MLL formula A inductively:

- $ST(X_i)$ and $ST(X_i^{\perp})$ are vertices labelled by X_i and $X_i^{\perp}$ respectively;
- $ST(A \otimes B)$ is an hyperedge labelled by $\otimes$ linking the conclusion of $ST(A)$ and $ST(B)$ as sources and having a vertex labelled by $A \otimes B$ as target;
- $ST(A \parr B)$ is an hyperedge labelled by $\parr$ linking the conclusion of $ST(A)$ and $ST(B)$ as sources and having a vertex labelled by $A \parr B$ as target.

The syntax hypergraph $ST(\vdash \Gamma)$ of $\vdash \Gamma$ is defined as the hypergraph disjoint union of all $ST(A_i)$ for $A_i \in \Gamma$. A switching (*cf.* Definition 30.16) φ still applies on $ST(\vdash \Gamma)$ as for correction hypergraphs. We write $ST(\vdash \Gamma)^{\varphi}$ for the switching φ applied on the syntax hypergraph $ST(\vdash \Gamma)$.

The *test* associated with the sequent $\vdash \Gamma$ and the switching φ is defined as the constellation $\mathtt{Test}(\vdash \Gamma)^{\varphi}$ such that $I_{\mathtt{Test}(\vdash \Gamma)^{\varphi}} := V^{ST(\vdash \Gamma)^{\varphi}}$ (it is indexed by vertices of the syntax tree) and $\mathtt{Test}(\vdash \Gamma)^{\varphi}[v] := v^{\bigstar}$.

The *set of tests* associated with the sequent $\vdash \Gamma$ is defined by $\mathtt{Tests}(\vdash \Gamma) := \{\mathtt{Test}(\vdash \Gamma)^{\varphi} \mid \varphi \text{ is a switching of } ST(\vdash \Gamma)\}$.

Figure 69.1: We expect this proof-structure to be able to pass any test of $\mathtt{Tests}(\vdash X_1 \otimes X_2, X_1^\perp \parr X_2^\perp)$. However, since the function symbols used in tests are not compatible with the ones of the proof-structure, we need a conversion of function symbols to allow interaction.

§69.16 **Example.** We give the compact (executed) form of tests coming from some sequents with a given switching:

$$\mathtt{Tests}(\vdash A, A^\perp)^\varphi := \begin{bmatrix} -A(X) \\ A(X) \end{bmatrix} + \begin{bmatrix} -A^\perp(X) \\ A^\perp(X) \end{bmatrix};$$

$$\mathtt{Tests}(\vdash A^\perp \parr B^\perp, A \otimes B)^\varphi \qquad (\varphi(\parr) = \parr_L) :=$$
$$\begin{bmatrix} -A^\perp(X) \\ (A^\perp \parr B^\perp)(X) \end{bmatrix} + \begin{bmatrix} -B^\perp(X) \\ \end{bmatrix} + \begin{bmatrix} -A(X), -B(X) \\ (A \otimes B)(X) \end{bmatrix};$$

$$\mathtt{Tests}(\vdash A^\perp \parr B^\perp, A \otimes B)^\varphi \qquad (\varphi(\parr) = \parr_R) :=$$
$$\begin{bmatrix} -A^\perp(X) \\ \end{bmatrix} + \begin{bmatrix} -B^\perp(X) \\ (A^\perp \parr B^\perp)(X) \end{bmatrix} + \begin{bmatrix} -A(X), -B(X) \\ (A \otimes B)(X) \end{bmatrix}.$$

§69.17 We defined MLL sequents as type labels in the sense of Definition 69.11. However, there is a minor technical problem: arbitrary constellations may not match with the tests we defined because of a difference of function symbols, as illustrated in Figure 69.1.

§69.18 A solution is to use binary stars to rename some rays and allow a connexion. This corresponds to a sort of generalised cut allowing a trivial connexion between two rays. I call these stars *adapters* in reference to adapter cables (for instance HDMI to VGA or USB-C to USB-A).

§69.19 **Definition** (Adapter)**.** Let Φ be a constellation. An *adapter* for Φ is a star $[r, r']$ where $\mathtt{op}(r)$ and $\mathtt{op}(r')$ are polarised rays of Φ such that $\mathtt{op}(r) \not\bowtie \mathtt{op}(r')$ (hence we artificially link two rays which could not be linked by using a star $[r, r']$).

§69.20 Notice that the translation of atomic formulas in Definition 69.15 actually corresponds to adapters between a vehicle and a test, hence tests were already independent of vehicles but hardwired adapters were linking vehicle and test in a proof-structure. This dependency is only artificial and appears when considering proof-structures as an entity which cannot be decomposed. This is the difference between soldered switches in keyboards and hot-swappable switches which can be replaced. In Figure 68.1 for instance,

the negative rays on the top are residuals of adapters which enforces a connexion with the vehicle.

"Transcendental syntax's Usine is proof-net theory without soldering components."

§69.21 **Proposition** (Correspondence between proof-structure tests and sequent tests)**.** Let $\mathcal{S}$ be a cut-free proof-structure. For all switching φ of $\mathcal{S}$, there exists an MLL sequent $\vdash \Gamma$ and a constellation of adapters Φ such that $\mathtt{Ex}(\Phi_{\mathcal{S}}^{\varphi}) = \mathtt{Ex}(\mathtt{Test}(\vdash \Gamma)^{\varphi} \uplus \Phi)$.

Proof. The constellation $\Phi_{\mathcal{S}}^{\varphi}$ corresponds to the syntax forest of a sequent with exactly one premise removed for each $\parr$ vertex. Hence, it naturally induces a sequent $\vdash \Delta$ and we define $\Gamma := \Delta$. The constellation $\mathtt{Test}(\vdash \Gamma)^{\varphi}$ structurally corresponds to $\Phi_{\mathcal{S}}^{\varphi}$ without the upper rays $-u_i(t_i)$ which allows connexion with the vehicle $\Phi_{\mathcal{S}}^{\mathrm{ax}}$. Apart from that, they both use the same translation function $(\cdot)^{\bigstar}$ on vertices for correctness hypergraphs. Assume we have a star $[-v(t), +w(X)]$ translating the atom w linked to some star $[-w(X), ...]$ in $\mathtt{Test}(\vdash \Gamma)^{\varphi}$. During the execution, they will merge into $[-v(t), ...]$. However, it is possible to construct Φ so to reproduce this step with an adapter $[-v(t), +A(X)]$ (by definition, the star $[-A(x), ...]$ which is isomorphic to $[-w(x), ...]$ must be present in $\Phi_{\mathcal{S}}^{\varphi}$). Moreover, because of the structural equivalence between the two constellations, they only differ by conjugation. It is then possible to extend Φ so that $\mathtt{Test}(\vdash \Gamma)^{\varphi}$ is turned exactly into $\Phi_{\mathcal{S}}^{\varphi}$. It follows that the two constellations must have the same normal form. □

§69.22 **Definition** (Typing)**.** We say that a constellation Φ is of type $\vdash \Gamma$, written $\vdash \Phi : \Gamma$ when $\Phi \in \mathtt{Ex}(\Phi_{\vdash \Gamma}^{\varphi} \uplus \Phi_{\mu})^{\perp}$ for a set of adapters Φ_{μ} and all switchings φ of $\vdash \Gamma$.

§69.23 **Proposition** (Reformulation of logical correctness)**.** A cut-free proof-structure $\mathcal{S}$ is MLL-certifiable if and only if there exists a sequent $\vdash \Gamma$ and a constellation of adapters Φ such that $\vdash \overset{+}{\Phi}{}_{\mathcal{S}}^{\mathrm{ax}} : \Gamma$ with $\perp \in \{\perp^1, \perp^R\}$. The same statement holds for MLL+MIX *w.r.t.* $\perp^{\mathrm{fin}}$.

Proof. By Proposition 69.21, there exist some sequent $\vdash \Gamma$ such that $\Phi_{\mathcal{S}}^{\varphi}$ is simulated by $\Phi \uplus \mathtt{Test}(\vdash \Gamma)^{\varphi}$ for some constellation of adapters Φ. By invariance of orthogonality under execution (*cf.* Lemma 69.8), this connexion is equivalent to a connexion between $\overset{+}{\Phi}{}_{\mathcal{S}}^{\mathrm{ax}}$ and $\Phi_{\mathcal{S}}^{\varphi}$. The orthogonality $\overset{+}{\Phi}{}_{\mathcal{S}}^{\mathrm{ax}} \in \mathtt{Tests}(\vdash \Gamma)^{\perp}$ and the same statement for MLL+MIX (*w.r.t.* $\perp^{\mathrm{fin}}$) both hold by a direct consequence of Corollary 68.21. □

§69.24 **Example.** From the sequent $\vdash A, A^{\perp}$, we can generate the only test $\begin{bmatrix} -A(X) \\ A(X) \end{bmatrix} + \begin{bmatrix} -A^{\perp}(X) \\ A^{\perp}(X) \end{bmatrix}$. It can be plugged to the vehicle $[+1(X), +2(X)]$ by using the adapters $[-1(X), +A(X)] + [-2(X), +A^{\perp}(X)]$. The normal form of the whole constellation is

$[A(X), A^\perp(X)]$, as expected.

§69.25 **The shape of vehicles.** Notice that in our interpretation, there is no assumption of the shape of vehicles. We are interested in constellations orthogonal to all tests of a given sequent. Consider the orthogonality $\perp^R$. If we have the two tests of $\mathtt{Tests}(\vdash A^\perp \parr B^\perp, A \otimes B)$ given in Example 69.16, it can be linked to the expected vehicle (with adapters applied):

$$[+(A^\perp \parr B^\perp)(\mathtt{l} \cdot X), +(A \otimes B)(\mathtt{l} \cdot X)] + [+(A^\perp \parr B^\perp)(\mathtt{r} \cdot X), +(A \otimes B)(\mathtt{r} \cdot X)]$$

but this also logically wrong vehicle would pass the test as well:

$$[+(A^\perp \parr B^\perp)(\mathtt{l} \cdot X), +(A \otimes B)(\mathtt{r} \cdot X)] + [+(A^\perp \parr B^\perp)(\mathtt{r} \cdot X), +(A \otimes B)(\mathtt{l} \cdot X)].$$

It means that we need another restriction to say that we would like to link "dual atoms". These additional restrictions are not explored here because they corresponds to second-order considerations (epidictics).

§69.26 However, independently of sequents, there are some correct shapes of vehicles that we wish for and which can be made explicit. We say that those correctly shaped vehicles are *well-formed*. They are the ones which will be put against tests of sequents.

§69.27 **Definition** (Well-formed vehicle)**.** Let Φ be a constellation defined in a coloured signature $(V, F, \mathtt{ar}, \frown, \lfloor\cdot\rfloor)$ with $F := F_0 \uplus F_+ \uplus F_-$. It is a *well-formed vehicle* if it is:

1. finite;
2. only made of binary stars $[f_i(t_i), f_j(t_j)]$;
3. all rays are disjoint, *i.e.* not α-unifiable (this is actually a condition that Girard required for constellations in general [Gir17]);
4. for each f_k, we have $f_k \in F_+ \uplus F_0$;
5. there is at least one f_k such that $f_k \in F_+$;
6. all t_k are stacks of directions constructed in the following grammar:

$$t ::= X \mid \mathtt{l} \cdot t \mid \mathtt{r} \cdot X.$$

Usage interpretation

§69.28 I refer to Section 43 for more details about the philosophy of Girard's Usage. Constellations are grouped into arbitrary sets called *pre-behaviours*, giving rise a notion of formula and *behaviours* will correspond to actual formulas of linear logic.

§69.29 **Definition** (Pre-behaviour)**.** A *pre-behaviour* $\mathbf{A}$ is a set of constellations.

§69.30 **Definition** (Behaviour)**.** A pre-behaviour $\mathbf{A}$ is a *behaviour* when there exists a pre-behaviour $\mathbf{B}$ such that $\mathbf{A} = \mathbf{B}^{\perp}$.

§69.31 **Lemma** (Invariance of typing under execution)**.** Let Φ be a constellation and $\mathbf{A}$ a behaviour such that $\Cap_C(\Phi, \Phi') = \emptyset$ for all $\Phi' \in \mathbf{A}^{\perp}$. We have $\Phi \in \mathbf{A}$ if and only if $\mathtt{Ex}_C(\Phi) \in \mathbf{A}$.

Proof. If $\mathbf{A}$ is a behaviour then $\mathbf{A} = \mathbf{A}^{\perp\perp}$, meaning that $\mathbf{A}$ is characterised by some tests $\mathbf{A}^{\perp}$. Hence we have to show that $\Phi \perp_C \Phi'$ such that $\Cap_C(\Phi, \Phi') = \emptyset$ for any $\Phi' \in \mathbf{A}^{\perp}$ if and only if $\mathtt{Ex}(\Phi) \perp_C \Phi'$. This is the consequence of the invariance of orthogonality under execution (*cf.* Lemma 69.8). □

§69.32 **Proposition** (Bi-orthogonal closure)**.** A pre-behaviour $\mathbf{A}$ is a behaviour if and only if $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Proof. The proof can be found in the literature [JS21, Proposition 15]. □

§69.33 **Definition** (Disjointness of behaviours)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two behaviours and a set of colours $C \subseteq F_+ \uplus F_-$. They are *disjoint* when for all $\Phi_A \in \mathbf{A}$ and $\Phi_B \in \mathbf{B}$, we have $\Cap_C(\Phi_A, \Phi_B) = \emptyset$.

§69.34 When two behaviours $\mathbf{A}$ and $\mathbf{B}$ are disjoint, for any pair of constellations $\Phi_A \in \mathbf{A}$ and $\Phi_B \in \mathbf{B}$, there is no path from one constellation to the other in $\mathfrak{D}[\Phi_A \uplus \Phi_B]$.

§69.35 **Definition** (Pre-tensor)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be disjoint pre-behaviours. We define their pre-tensor by $\mathbf{A} \odot \mathbf{B} = \{\Phi_1 \uplus \Phi_2 \mid \Phi_1 \in \mathbf{A}, \Phi_2 \in \mathbf{B}\}$.

§69.36 **Definition** (Tensor)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be disjoint behaviours. We define their tensor by

$$\mathbf{A} \otimes \mathbf{B} = (\mathbf{A} \odot \mathbf{B})^{\perp\perp}.$$

§69.37 The pre-tensor is the natural definition of tensor product pairing constellations of two pre-behaviours. The real tensor product adds a bi-orthogonal closure $(\cdot)^{\perp\perp}$ in order to ensure that we get a behaviour (it is not necessarily the case without the closure, depending on the orthogonality we consider). It is indeed a generalisation of the usual tensor because, depending on the orthogonality relation, its orthogonal can contain way more than what we expect from proof-structures because of the huge space of objects provided by stellar resolution. In case $\mathbf{A} \odot \mathbf{B} = \mathbf{A} \otimes \mathbf{B}$, we say that *internal completeness* holds for tensor.

Figure 69.2: Counter-example of non-associativity which is variant of Figure 64.1. We have $\mathtt{Ex}_{\{c\}}(\Phi_1 \uplus \Phi_2) = [\mathtt{l} \cdot X]$ and $\mathtt{Ex}_{\{c\}}(\mathtt{Ex}_{\{c\}}(\Phi_1 \uplus \Phi_2) \uplus \Phi_3) = [-c(\mathtt{r} \cdot X)] + [\mathtt{l} \cdot X]$, but $\mathtt{Ex}_{\{c\}}(\Phi_1 \uplus \mathtt{Ex}_{\{c\}}(\Phi_2 \uplus \Phi_3)) = [-c(\mathtt{l} \cdot X)] + [\mathtt{r} \cdot X]$ which is different.

§69.38 **Proposition** (Commutativity and associativity of tensor)**.** Given $\mathbf{A}, \mathbf{B}, \mathbf{C}$ pairwise disjoint behaviours, we have (1) $\mathbf{A} \otimes \mathbf{B} = \mathbf{B} \otimes \mathbf{A}$ and (2) $\mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C}$.

Proof. (1) By the definition of tensor, we have $\Phi_1 \uplus \Phi_2 \in \mathbf{A} \otimes \mathbf{B}$ when $\Phi_1 \uplus \Phi_2 \in \{\Phi_1 \uplus \Phi_2 \mid \Phi_1 \in \mathbf{A}, \Phi_2 \in \mathbf{B}\}^{\perp\perp}$. We also have $\Phi_2 \uplus \Phi_1 \in \mathbf{B} \otimes \mathbf{A}$. But since $\Phi_1 \uplus \Phi_2 = \Phi_2 \uplus \Phi_1$ by commutativity of multiset disjoint union, we obtain $\mathbf{A} \otimes \mathbf{B} = \mathbf{B} \otimes \mathbf{A}$. (2) In the same fashion, by using the associativity of multiset disjoint union, we obtain $\mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C}$. □

§69.39 The other connectives are then defined by interactive testing, *e.g.* the elements of $\mathbf{A} \parr \mathbf{B}$ are the elements passing the tests of $\mathbf{A}^\perp \otimes \mathbf{B}^\perp$. This is why we can speak about *interactive types*.

§69.40 **Definition** (Par and linear implication)**.** Let $\mathbf{A}, \mathbf{B}$ be disjoint behaviours. We define:

$$\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp \quad \text{and} \quad \mathbf{A} \multimap \mathbf{B} = \mathbf{A}^\perp \parr \mathbf{B}.$$

§69.41 **Remark** (Implicit exchange)**.** The commutativity and associativity of $\otimes$ are preserved for $\parr$: we have $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp = (\mathbf{B}^\perp \otimes \mathbf{A}^\perp)^\perp = \mathbf{B} \parr \mathbf{A}$. This corresponds to the fact that the exchange rule is implicit in linear logic.

§69.42 In Figure 69.2, we can see that associativity fails when execution is treated as a binary operator on constellations. As in partial pre-execution (*cf.* Section 64), we need a restriction on the interaction between constellations. A technical precondition is defined for the associativity, and *trefoil property* [Sei16a, Theorem 40] is stated as a corollary (*cf.* Section 65).

§69.43 **Theorem** (Associativity of (pairwise) execution)**.** Choose a set of colours $C \subseteq F_+ \uplus F_-$. For constellations Φ_1, Φ_2 and Φ_3 such that $\Cap_C(\Phi_1, \Phi_2, \Phi_3) = \emptyset$, we have:

$$\mathtt{Ex}_C(\Phi_1 \uplus \mathtt{Ex}_C(\Phi_2 \uplus \Phi_3)) = \mathtt{Ex}_C(\mathtt{Ex}_C(\Phi_1 \uplus \Phi_2) \uplus \Phi_3).$$

Proof. Assume we have $\Cap_C(\Phi_1, \Phi_2, \Phi_3) = \emptyset$. Hence, by definition, no ray is shared by the three constellations. Let $P(i,j)$, be the set of paths reaching $(\Phi_1 \uplus \Phi_2 \uplus \Phi_3)[i][j]$ in $\mathfrak{D}[\Phi_1 \uplus \Phi_2 \uplus \Phi_3; A]$. By the previous statement, these paths traverse at most two constellations in $\{\Phi_1, \Phi_2, \Phi_3\}$. By using the reasoning of the proof of partial pre-execution (*cf.* Lemma 64.9), the paths $P(i,j)$ traversing Φ_2 and Φ_3 can be reduced with no effect on other connexions (since no rays are shared). Hence, the stars of Φ_1 can connect to the stars of $\mathtt{Ex}_C(\Phi_2 \uplus \Phi_3)$ in the same way as in $\Phi_2 \uplus \Phi_3$. It follows that $\mathtt{Ex}_C(\Phi_1 \uplus \mathtt{Ex}_C(\Phi_2 \uplus \Phi_3)) = \mathtt{Ex}_C(\Phi_1 \uplus \Phi_2 \uplus \Phi_3)$. By the same reasoning, we also have $\mathtt{Ex}_C(\mathtt{Ex}_C(\Phi_1 \uplus \Phi_2) \uplus \Phi_3) = \mathtt{Ex}_C(\Phi_1 \uplus \Phi_2 \uplus \Phi_3)$, hence execution is associative. □

§69.44 **Theorem** (Trefoil Property for execution-based orthogonality)**.** Let $C \subseteq F_+ \uplus F_-$ be a set of colours. For constellations Φ_1, Φ_2, Φ_3 and for $i, j, k \in \{1, 2, 3\}$ such that $\Cap_C(\Phi_1, \Phi_2, \Phi_3) = \emptyset$, we have:

$$\Phi_1 \perp_C \mathtt{Ex}_C(\Phi_2 \uplus \Phi_3) \quad \text{if and only if} \quad \mathtt{Ex}_C(\Phi_1 \uplus \Phi_2) \perp_C \Phi_3.$$

Proof. Assume that P is a property corresponding to the orthogonality relation $\perp$ based on execution, *i.e.* we have $\Phi_1 \perp_C \Phi_2$ if and only if $P(\mathtt{Ex}_C(\Phi_1 \uplus \Phi_2))$. The statement can be rewritten as follows: $P(\mathtt{Ex}_C(\Phi_1 \uplus \mathtt{Ex}_C(\Phi_2 \uplus \Phi_3)))$ if and only if $P(\mathtt{Ex}_C(\mathtt{Ex}_C(\Phi_1 \uplus \Phi_2) \uplus \Phi_3))$. This is a direct consequence of the associativity (*cf.* Theorem 69.43). □

§69.45 It is then possible to recover the adjunction property that the trefoil property generalises (*cf.* Section 65).

§69.46 **Corollary** (Adjunction)**.** Choose a set of colours $C \subseteq F_+ \uplus F_-$. For all constellations Φ_f, Φ_a and Φ_b such that $\Cap_C(\Phi_a, \Phi_b) = \emptyset$, we have:

$$\Phi_f \perp_C \Phi_a \uplus \Phi_b \quad \text{if and only if} \quad \mathtt{Ex}_C(\Phi_f \uplus \Phi_a) \perp_C \Phi_b.$$

Proof. By symmetry of orthogonality relations and invariance of orthogonality under execution (*cf.* Lemma 69.8), we have $\Phi_f \perp_C \Phi_a \uplus \Phi_b$ if and only if $\Phi_f \perp_C \mathtt{Ex}_C(\Phi_a \uplus \Phi_b)$. In order to conclude with the trefoil property, it remains to show the precondition, *i.e.* that we have $\Cap_C(\Phi_f, \Phi_a, \Phi_b) = \emptyset$. We assumed $\Cap_C(\Phi_a, \Phi_b) = \emptyset$, meaning that no variable were shared by both Φ_a and Φ_b. It follows that a variable cannot be shared by Φ_f, Φ_a and Φ_b at the same time because otherwise, it would be shared by Φ_a and Φ_b as well. □

§69.47 As explained in Section 65, the adjunction allows to show that the behaviour $\mathbf{A} \multimap \mathbf{B}$ corresponding to linear implication has a functional behaviour. This is a necessary condition for the definition of linear logic.

§69.48 **Proposition** (Alternative linear implication)**.** Let $\mathbf{A}, \mathbf{B}$ be two disjoint behaviours. We have $\mathbf{A} \multimap \mathbf{B} = \{\Phi_f \mid \forall\ \Phi_a \in \mathbf{A}, \mathtt{Ex}(\Phi_f \uplus \Phi_a) \in \mathbf{B}\}$.

Proof. By Definition 69.40, we have $\mathbf{A} \multimap \mathbf{B} = (\mathbf{A} \otimes \mathbf{B}^{\perp})^{\perp}$. We have $\Phi_f \in \mathbf{A} \multimap \mathbf{B}$ if and only if for all $\Phi_a \in \mathbf{A}$, $\mathtt{Ex}(\Phi_f \uplus \Phi_a) \in \mathbf{B}$. Since $\mathbf{B}$ is a behaviour, by Definition 69.30, there exists $\Phi_{b'} \in \mathbf{B}^{\perp}$ such that $\mathtt{Ex}(\Phi_f \uplus \Phi_a) \perp \Phi_{b'}$. By the adjunction (*cf.* Corollary 69.46), $\Phi_f \perp (\Phi_a \uplus \Phi_{b'})$, hence $\Phi_f \in (\mathbf{A} \otimes \mathbf{B}^{\perp})^{\perp}$. The proof only relies on equivalences hence a bi-inclusion is proved. □

70 Soundness and completeness

§70.1 In this section:

- Girard's adequacy between Usine and Usage presented in Section 43 is formalised;
- more classic results such as soundness and completeness *w.r.t.* proof-net theory are presented.

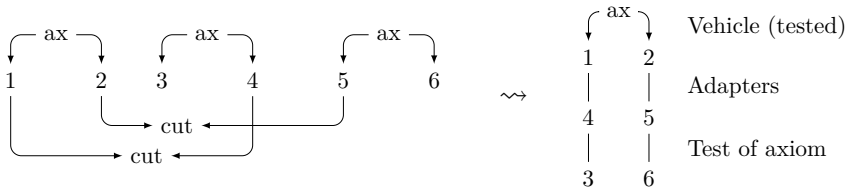
Adequacy between Usine and Usage

§70.2 In this subsection, our favourite orthogonality relation which will be mentioned is $\perp^R$ which require that the normal form is the star of roots (unpolarised rays) coming from the two interacting constellations.

§70.3 As explained in Section 43, in transcendental syntax, the process of execution is able to express both cut-elimination and correctness testing. What changes is the shape of objects which interact:

- with cut-elimination, two vehicles interact;
- with correctness testing, a vehicle and a (compact) test interact.

The orthogonality relation $\perp^R$ formalises the two situations. We already know how it captures MLL correctness but it also works for cut-elimination. What is a "correct" interaction by cut-elimination? When the two vehicles Φ and Φ' interacting only leave the star of uncoloured rays which were not related to cuts (that all rays related to cuts have been eliminated), in other words when $\Phi \perp^R \Phi'$. In the same fashion of mixing correctness and cut-elimination, plugging two vehicles by axiomatic cuts can be seen as plugging a vehicle and a sort of test (which coincides with another vehicle) by using cuts as adapters as shown in Figure 70.1.



(a) Proof-structure view.

$$[+1(X),+2(X)]+[3(X),+4(X)]+[+5(X),6(X)] \rightsquigarrow [-1(X),-4(X)]+[-2(X),+5(X)]$$

$$[+1(X),+2(X)] \qquad \begin{bmatrix} -1(X) \\ +4(X) \end{bmatrix} + \begin{bmatrix} -2(X) \\ +5(X) \end{bmatrix} \qquad \begin{bmatrix} -4(X) \\ 3(X) \end{bmatrix} + \begin{bmatrix} -5(X) \\ 6(X) \end{bmatrix}$$

(b) Constellation view in cut-elimination style (left) and block style for correctness testing (right).

Figure 70.1: Cut-elimination between two vehicles (on the left) seen as testing (on the right). Exactly as for testing, we expect to obtain the roots by execution. The cuts correspond to adapters. We see that the cut-elimination corresponding to testing an axiom against the only test of $\vdash A, A^{\perp}$. The free rays explicitly becomes roots of the test.

§70.4 **The meaning of cut-elimination.** Cut-elimination can then be reformulated. Its meaning is that every interaction between vehicles is valid (normalises into the star of roots). More formally: for all well-formed vehicles Φ, Φ', we have $\Phi \perp^R \Phi$. This is not always the case. A trivial example is to consider a case where cut-elimination fails (*cf.* Example 67.5). The two proof-structures interacting are indeed well-formed but we do not normalise into the star of roots. This is where correctness tests are needed.

§70.5 **Theorem** (Multiplicative adequacy)**.** Let Φ and Φ' be two well-formed vehicles. If $\vdash \overset{+}{\Phi} : \Gamma$ and $\overset{+}{\Phi'} : \Gamma^{\perp}$, then $\Phi \perp^R \Phi' \uplus \Psi$ where Ψ is a constellation of fully negative adapters (representing cuts).

Proof. This is another formulation of Theorem 67.10. □

§70.6 Adequacy corresponds to a relation between Usine and Usage because the shape of objects given by Usine tests guarantees the use of objects (behaving like proof-structures). More precisely, Usine tests are also related to Usage's behaviours because if $\vdash \Phi : \Gamma$ then, $\Phi \in [\![\vdash \Gamma]\!]$, meaning that a constellation passing the tests of $\vdash \Gamma$ behaves like an object of the idealised type $[\![\vdash \Gamma]\!]$. This implies that $\texttt{Tests}(\vdash \Gamma)^{\perp} \subseteq [\![\vdash \Gamma]\!]$.

A complete model of MLL+MIX

§70.7 We can also choose to prove more traditional properties of soundness and completeness *w.r.t.* proof-structures. This shows that we correctly captured the behaviour of proof-structures, their cut-elimination and their correctness. We start by associating a behaviour to formulas.

§70.8 Formula labels are interpreted by behaviours where distinct behaviours are associated with occurrences of variables by a function called *basis of interpretation*. Following previous works of Seiller [Sei12a, Definition 46], the behaviours corresponding to formula labels are *localised* formulas: they are defined using the same grammar as MLL formulas, except that variables are of the form $X_i(t)$, where t is a term (here representing the path address described in Definition 66.7) used to distinguish occurrences of a same atomic formula X_i. Two behaviours $X_i(t)$ and $X_i(u)$ with $t \neq u$ represent the same atom at different locations and should correspond to the same behaviour modulo application of an adapter.

§70.9 **Definition** (Basis of interpretation)**.** A *basis of interpretation* is a function Ω producing a behaviour $\Omega(A, i, t)$ when given a formula $A \in \mathcal{F}_{\mathrm{MLL}}$, a natural number i (index of occurrence) and a term $t \in \mathrm{Addr}_x(\mathcal{S})$ (*cf.* Definition 66.7). A basis of interpretation has to satisfy the condition that $\mathtt{Ex}\big(\Omega(A, i, t) + [+A(t), +B(u)]\big) = \Omega(B, j, u)$ when $i = j$ and otherwise $\Omega(A, i, t)$ and $\Omega(B, j, u)$ are disjoint, such that $\mathbf{A} + \phi = \{\Phi + \phi \mid \Phi \in \mathbf{A}\}$ for a behaviour $\mathbf{A}$ and a star ϕ.

§70.10 **Definition** (Interpretation of MLL formulas)**.** Given a basis of interpretation Ω, a formula C representing the conclusion of a sequent, and an MLL formula occurrence A identified by a unique unary function symbol $A(X)$ (*cf.* Definition 66.7). We define the *interpretation* $[\![A, t]\!]_\Omega$ along Ω and a term t (encoding the address of A *w.r.t.* a conclusion C) inductively:

- $[\![C, X_i, t]\!]_\Omega = \Omega(C, i, t)$;
- $[\![C, X_i^\perp, t]\!]_\Omega = \Omega(C, i, t)^\perp$;
- $[\![C, A \otimes B, t]\!]_\Omega = [\![C, A, \mathtt{l} \cdot t]\!]_\Omega \otimes [\![C, B, \mathtt{r} \cdot t]\!]_\Omega$;
- $[\![C, A \parr B, t]\!]_\Omega = [\![C, A, \mathtt{l} \cdot t]\!]_\Omega \parr [\![C, B, \mathtt{r} \cdot t]\!]_\Omega$.

We write $[\![C]\!]$ for $[\![C, C, X]\!]$ and extend the interpretation to sequents with:

$$[\![\vdash C_1, ..., C_n]\!]_\Omega := [\![C_1]\!]_\Omega \parr ... \parr [\![C_n]\!]_\Omega.$$

§70.11 **Remark.** The interpretation of an axiom under an basis of interpretation Ω is defined by $[\![\vdash X_1, X_1^\perp]\!]_\Omega = [\![X_1]\!]_\Omega \parr [\![X_1^\perp]\!] = \Omega(X_1, 1, X) \parr \Omega(X_1^\perp, 1, X)^\perp$.

§70.12 We prove soundness and completeness for MLL+MIX. Theorem 68.19 shows that asking for a strongly normalising union between vehicle and test corresponds to MLL+MIX

correctness. This is the key ingredient in the proof of completeness. In this section, we consider the orthogonality $\perp^{\text{fin}}$ exclusively.

§70.13 Instead of the usual soundness property, we prove an extension called *full soundness* [Sei12a, Theorem 55] which takes cut-elimination into account. In terms of the adequacy used in realisability interpretations, proving the soundness property corresponds to showing that $\vdash \Phi : \Gamma$ implies $[\![\vdash \Gamma]\!]_\Omega$ for some basis of interpretation Ω, except that for $\vdash \Phi : \Gamma$ we only consider constellations coming from proof-nets.

§70.14 **Lemma.** Let A, B be MLL formulas, $\Gamma = C_1, ..., C_n, \Delta = D_1, ..., D_m$ be sets of MLL formulas and Ω be a basis of interpretation. We have $([\![\vdash \Gamma]\!]_\Omega \parr [\![A]\!]_\Omega) \otimes ([\![\vdash \Delta]\!]_\Omega \parr [\![B]\!]_\Omega) \subseteq [\![\vdash \Gamma]\!]_\Omega \parr [\![\vdash \Delta]\!]_\Omega \parr [\![A \otimes B]\!]_\Omega$.

Proof. The idea is to show $([\![C_1^\perp \otimes ... \otimes C_n^\perp]\!]_\Omega \multimap [\![A]\!]_\Omega) \otimes ([\![D_1^\perp \otimes ... \otimes D_m^\perp]\!]_\Omega \multimap [\![B]\!]_\Omega) \subseteq ([\![C_1^\perp \otimes ... \otimes C_n^\perp]\!]_\Omega \otimes [\![D_1^\perp \otimes ... \otimes D_m^\perp]\!]_\Omega) \multimap [\![A \otimes B]\!]_\Omega$ which is equivalent to $([\![C]\!]_\Omega \multimap [\![A]\!]_\Omega) \otimes ([\![D]\!]_\Omega \multimap [\![B]\!]_\Omega) \subseteq ([\![C]\!]_\Omega \otimes [\![D]\!]_\Omega) \multimap [\![A \otimes B]\!]_\Omega$ for $C := C_1^\perp \otimes ... \otimes C_n^\perp$ and $D := D_1^\perp \otimes ... \otimes D_m^\perp$. Assume we have two functions $\Phi_{C,A} \in [\![C]\!]_\Omega \multimap [\![A]\!]_\Omega$ and $\Phi_{D,B} \in [\![D]\!]_\Omega \multimap [\![B]\!]_\Omega$. We can construct their disjoint union $\Phi_{C,A} \uplus \Phi_{D,B} \in ([\![C]\!]_\Omega \multimap [\![A]\!]_\Omega) \otimes ([\![D]\!]_\Omega \multimap [\![B]\!]_\Omega)$. If we provide to $\Phi_{C,A} \uplus \Phi_{D,B}$ an argument $\Phi \in [\![C]\!]_\Omega \otimes [\![D]\!]_\Omega$, then since C and D are disjoint, each function $\Phi_{C,A}$ and $\Phi_{D,B}$ will take their argument separately and produce $\Phi' \in [\![A \otimes B]\!]_\Omega$. Therefore, $\Phi_{C,A} \uplus \Phi_{D,B} \in ([\![C]\!]_\Omega \otimes [\![D]\!]_\Omega) \multimap [\![A \otimes B]\!]_\Omega$. □

§70.15 **Lemma.** If $\mathbf{A}$ is a pre-behaviour then $\mathbf{A}^\perp \neq \emptyset$.

Proof. Any constellation with only uncoloured rays strongly normalise with any constellation so it is always part of the orthogonal of a pre-behaviour. □

§70.16 **Lemma.** If $\mathbf{A}$ is a behaviour and $\Phi \in \mathbf{A}$ then $|\mathtt{Ex}(\Phi)| < \infty$.

Proof. Assume we have a behaviour $\mathbf{A}$ and a constellation $\Phi \in \mathbf{A}$. By definition of behaviour, we have $\mathbf{A} = \mathbf{A}^{\perp\perp}$. By Lemma 70.15 there must be some $\Phi' \in \mathbf{A}^\perp$ such that $\Phi \uplus \Phi'$ is strongly normalising. Assume Φ is not strongly normalising. Then, Φ can produce infinitely many *saturated* correct diagrams. Such diagrams cannot be extended with stars of Φ' in $\Phi \uplus \Phi'$ because they are in disjoint union, hence these infinitely many saturated diagrams are preserved and $\Phi \uplus \Phi'$ cannot be strongly normalising, which is contradictory. Therefore, Φ must be strongly normalising. □

§70.17 **Theorem** (Full soundness for MLL+MIX)**.** Let $\vdash \mathcal{S} : \Gamma$ be an MLL+MIX proof-net and Ω a basis of interpretation. We have $\mathtt{Ex}(\Phi_{\mathcal{S}}^{\mathtt{comp}}) \in [\![\vdash \Gamma]\!]_\Omega$.

Proof. We start with the case of cut-free proofs normalising into themselves. The proof is done by induction on the proof-net structure of $\mathcal{S}$.

- Assume we have $\vdash \mathcal{S} : X_i, X_i^\perp$. We would like to show that $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![X_i]\!]_\Omega \parr [\![X_i^\perp]\!]_\Omega = [\![X_i, X_i, X]\!]_\Omega \parr [\![X_i^\perp, X_i, X]\!]^\perp_\Omega = \Omega(X_i, i, X) \parr \Omega(X_i^\perp, i, X)^\perp = \big(\Omega(X_i, i, X)^\perp \otimes \Omega(X_i^\perp, i, X)\big)^\perp$. Let $\Phi_1 \uplus \Phi_2 \in \Omega(X_i, i, X)^\perp \otimes \Omega(X_i^\perp, i, X)$ with $\Phi_1 \in \Omega(X_i, i, X)^\perp$ and $\Phi_2 \in \Omega(X_i^\perp, i, X)$. It is sufficient to show that $|\mathtt{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}})| < \infty$, *i.e.* that the axiom strongly normalises with its tests. By Definition 70.9 since we have $\Phi^{\mathrm{ax}}_{\mathcal{S}} = [+X_i(X), +X_i^\perp(X)]$, we have $\mathtt{Ex}(\Phi^{\mathrm{ax}}_{\mathcal{S}} \uplus \Phi_2) \in \Omega(X_i, i, X)$ which is orthogonal to Φ_1. By orthogonality, it follows that $|\mathtt{Ex}(\Phi_1 \uplus \mathtt{Ex}(\Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}}))| < \infty$. Now, by definition of tensor,
- Assume we have $\vdash \mathcal{S} : \Gamma, \Delta, A \otimes B$ coming from $\vdash \mathcal{S}_1 : \Gamma, A$ and $\vdash \mathcal{S}_2 : \Delta, B$. We have to show $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![\vdash \Gamma]\!]_\Omega \parr [\![\vdash \Delta]\!]_\Omega \parr [\![A \otimes B]\!]_\Omega$. By induction hypothesis, we have $\Phi^{\mathrm{ax}}_{\mathcal{S}_1} \in [\![\vdash \Gamma, A]\!]_\Omega = [\![\vdash \Gamma]\!]_\Omega \parr [\![A]\!]_\Omega$ and $\Phi^{\mathrm{ax}}_{\mathcal{S}_2} \in [\![\vdash \Delta, B]\!]_\Omega = [\![\vdash \Delta]\!]_\Omega \parr [\![B]\!]_\Omega$. By a conjugation μ such that $\Phi^{\mathrm{ax}}_{\mathcal{S}_1}$ and $\Phi^{\mathrm{ax}}_{\mathcal{S}_2}$ are made distinct, we can relocate the atoms and obtain a constellation $\Phi_\mu \in ([\![\vdash \Gamma]\!]_\Omega \parr [\![A]\!]_\Omega) \otimes ([\![\vdash \Delta]\!]_\Omega \parr [\![B]\!]_\Omega)$ such that $\Phi_\mu = \mu(\Phi^{\mathrm{ax}}_{\mathcal{S}_1}) \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}_2}$. Now, by the definition of tensor for proof-structures, we have a preservation of axioms and $\Phi^{\mathrm{ax}}_{\mathcal{S}}$ equivalent to Φ_μ up to conjugation (and this conjugation could be chosen for μ). By Lemma 70.14, we have $([\![\vdash \Gamma]\!]_\Omega \parr [\![A]\!]_\Omega) \otimes ([\![\vdash \Delta]\!]_\Omega \parr [\![B]\!]_\Omega) \subseteq [\![\vdash \Gamma]\!]_\Omega \parr [\![\vdash \Delta]\!]_\Omega \parr [\![A \otimes B]\!]_\Omega$, hence $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![\vdash \Gamma]\!]_\Omega \parr [\![\vdash \Delta]\!]_\Omega \parr [\![A \otimes B]\!]_\Omega$.
- Assume we have $\vdash \mathcal{S} : \Gamma, A \parr B$ coming from $\vdash \mathcal{S}' : \Gamma, A, B$. We would like to show that $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![\vdash \Gamma]\!]_\Omega \parr [\![A]\!]_\Omega \parr [\![B]\!]_\Omega$. This directly follows from the induction hypothesis and the fact that we have $[\![\vdash \Gamma, A, B]\!]_\Omega = [\![\vdash \Gamma]\!]_\Omega \parr [\![A]\!]_\Omega \parr [\![B]\!]_\Omega$ by definition.
- Assume we have $\vdash \mathcal{S} : \Gamma, \Delta$ coming from $\vdash \mathcal{S}_1 : \Gamma$ and $\vdash \mathcal{S}_2 : \Delta$ (by using the MIX rule). We have to show $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![\vdash \Gamma]\!]_\Omega \parr [\![\vdash \Delta]\!]_\Omega$ knowing that the induction hypothesis states that $\Phi^{\mathrm{ax}}_{\mathcal{S}_1} \in [\![\vdash \Gamma]\!]_\Omega$ and $\Phi^{\mathrm{ax}}_{\mathcal{S}_2} \in [\![\vdash \Delta]\!]_\Omega$. Since the MIX rule only places two proofs next to each other, we have $\Phi^{\mathrm{ax}}_{\mathcal{S}} = \Phi^{\mathrm{ax}}_{\mathcal{S}_1} \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}_2}$ by definition. By definition of tensor, we have $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![\vdash \Gamma]\!]_\Omega \otimes [\![\vdash \Delta]\!]_\Omega$. It remains to show that $\mathbf{A} \otimes \mathbf{B} \subseteq \mathbf{A} \parr \mathbf{B}$ in general, which would imply $\Phi^{\mathrm{ax}}_{\mathcal{S}} \in [\![\vdash \Gamma]\!]_\Omega \parr [\![\vdash \Delta]\!]_\Omega$. We have $\mathbf{A} \parr \mathbf{B} \subseteq (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp$, hence we have to show that $\mathbf{A} \otimes \mathbf{B} \subseteq (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp$. Let $\Phi_1 \uplus \Phi_2 \in \mathbf{A} \otimes \mathbf{B}$ and $\Phi'_1 \uplus \Phi'_2 \in \mathbf{A}^\perp \otimes \mathbf{B}^\perp$. We know that $\Phi_1 \perp \Phi'_1$ and $\Phi_2 \perp \Phi'_2$. We have $\Phi_1 \uplus \Phi_2 \perp \Phi'_1 \uplus \Phi'_2$, *i.e.* that $\Phi_1 \uplus \Phi_2 \uplus \Phi'_1 \uplus \Phi'_2$ is strongly normalising. In particular, we cannot have a crossed infinite interaction between Φ_1 and Φ'_2 or between Φ_2 and Φ'_1 because otherwise one constellation would have to not be strongly normalising (because a strongly normalising constellation produces finitely many saturated diagrams which cannot be extended so to make an infinite execution) but this would contradict Lemma 70.16.

If the proof has cuts, then by Theorem 67.10, we can execute its translation (a constellation) so that the normal form corresponds to the normal form of the proof. This proof is necessarily cut-free, hence the case of cut-free proofs also applies to this case. □

§70.18 **Lemma.** Let Ω be a basis of interpretation and $\vdash \Gamma$ an MLL sequent. Then, we have $\mathtt{Tests}(\vdash \Gamma) \subseteq [\![\vdash \Gamma]\!]^{\perp\mathrm{fin}}_\Omega$.

Proof. Assume we have $\mathtt{Test}(\vdash \Gamma)^\varphi \in \mathtt{Tests}(\vdash \Gamma)$ for a switching φ of $\vdash \Gamma$. The proof is done by induction on $\vdash \Gamma$.

- If $\Gamma = \{A_1, ..., A_n\}$ where the A_i are formulas X_i or $X_i^\perp$, then there is a single switching φ. Because typing is invariant under execution, we can consider a simplification of tests by fusion $\mathtt{Ex}(\mathtt{Test}(\vdash \Gamma)^\varphi) = \sum_{i=1}^n [-A_i(t_i), A_i(X)]$ where t_i is the expected encoding of the address of the atom A_i. We would like to show that $\mathtt{Test}(\vdash \Gamma)^\varphi \in [\![\vdash A_1, ..., A_n]\!]_\Omega^\perp = [\![A_1, A_1, t_1]\!]_\Omega^\perp \otimes ... \otimes [\![A_n, A_n, t_n]\!]_\Omega^\perp$. We show that $[-A_i(t_i), A_i(X)] \in [\![A_i, A_i, t_i]\!]_\Omega^\perp$. Let $\Phi_i \in [\![A_i, A_i, t_i]\!]_\Omega$. Because $[\![A_i, A_i, t_i]\!]_\Omega$ is a behaviour, we can use Lemma 70.16 and infer that $|\mathtt{Ex}(\Phi_i)| < \infty$. Adding $[-A_i(t_i), A_i(X)]$ to a strongly normalising constellation cannot cause divergence, hence we must have $[-A_i(t_i), A_i(X)] \perp \Phi_i$ and $[-A_i(t_i), A_i(X)] \in [\![A_i, A_i, t_i]\!]_\Omega^\perp$. Now, since $\mathtt{Test}(\vdash \Gamma)^\varphi$ is made of a disjoint union of constellations of $[\![A_i, A_i, t_i]\!]_\Omega^\perp$, it follows that $\mathtt{Test}(\vdash \Gamma)^\varphi \in [\![\vdash A_1, ..., A_n]\!]_\Omega^\perp$.
- If $\vdash \Gamma$ is $\vdash \Delta, A \parr B$, then a switching φ of $\vdash \Delta, A \parr B$ is a switching $\bar{\varphi}$ of $\vdash \Delta, A, B$ extended with a left or right selection of premise between A and B, both linked by a $\parr$ connective. By the induction hypothesis, we have $\mathtt{Test}(\vdash \Delta, A, B)^{\bar{\varphi}} \in [\![\vdash \Delta, A, B]\!]_\Omega = [\![\vdash \Delta]\!]_\Omega \parr [\![A]\!]_\Omega \parr [\![B]\!]_\Omega$ and we would like to show that $\mathtt{Test}(\vdash \Delta, A \parr B)^\varphi \in [\![\vdash \Delta, A \parr B]\!]_\Omega = [\![\vdash \Delta]\!]_\Omega \parr [\![A \parr B]\!]_\Omega = [\![\vdash \Delta]\!]_\Omega \parr [\![A \parr B, A, \mathtt{l}\cdot X]\!] \parr [\![A \parr B, B, \mathtt{r}\cdot X]\!]_\Omega$. The constellation $\mathtt{Test}(\vdash \Delta, A, B)^{\bar{\varphi}}$ uses terms $A(t)$ and $B(u)$ but when we add a $\parr$ link between A and B, these terms are relocated relatively to the conclusion $A \parr B$ and we obtain $(A \parr B)(\mathtt{l}\cdot t)$ and $(A \parr B)(\mathtt{r}\cdot u)$. Since they only differ by a conjugation, the two tests will react in the same way with respects to strong normalisation, *i.e.* $\mathtt{Test}(\vdash \Delta, A, B)^{\bar{\varphi}} \in ([\![\vdash \Delta]\!]_\Omega^\perp \otimes [\![A]\!]_\Omega^\perp \otimes [\![B]\!]_\Omega^\perp)^\perp$ implies $\mathtt{Test}(\vdash \Delta, A \parr B)^\varphi \in ([\![\vdash \Delta]\!]_\Omega^\perp \otimes [\![A \parr B]\!]_\Omega^\perp)^\perp$.
- If $\vdash \Gamma$ is $\vdash \Delta, A \otimes B$, a switching of $\vdash \Gamma$ is a switching of $\vdash \Delta, A, B$ extended to the additional $\otimes$ connective linking A and B, and $\mathtt{Test}(\vdash \Delta, A \otimes B)^\varphi$ can be defined from $\mathtt{Test}(\vdash \Delta, A, B)^\varphi$ by removing the uncoloured rays $A(x)$ and $B(x)$, and adding new stars $[-A(X), -B(X), +(A \otimes B)(X)] + [-(A \otimes B)(X), (A \otimes B)(X)]$. One can show that $[\![\vdash \Delta, A \otimes B]\!]_\Omega$ is generated (in the sense of bi-orthogonal closure) by a pre-behaviour E, *i.e.* that $[\![\vdash \Delta, A \otimes B]\!]_\Omega = E^{\perp\perp}$ for some E, similarly to how $\mathbf{A} \otimes \mathbf{B}$ is generated by a bi-orthogonal closure on the pre-tensor $\mathbf{A} \odot \mathbf{B}$ (*cf.* Definition 69.35). In this pre-behaviour E, the rays coming from A are disjoint from the rays coming from B (because of the requirement of exclusion of interaction). By using the induction hypothesis $\mathtt{Tests}(\vdash \Delta, A, B) \subseteq [\![\vdash \Delta, A, B]\!]_\Omega^{\perp\,\mathrm{fin}}$, this shows the result since this implies that $\mathtt{Test}(\vdash \Delta, A \otimes B)^\varphi \in E^{\perp\,\mathrm{fin}}$ and $\mathtt{Test}(\vdash \Delta, A \otimes B)^\varphi \in E^{\perp\perp\perp} = [\![\vdash \Delta, A \otimes B]\!]_\Omega^\perp$ since it is known that $X^\perp = X^{\perp\perp\perp}$ in general for any pre-behaviour X [JS21, Corollary 9].

□

§70.19 **Definition** (Proof-like constellation)**.** The syntax tree $ST(\vdash \Gamma)$ of a sequent induces a set of rays by Definition 66.7 by computing the address of each atom in $ST(\vdash \Gamma)$.

We note this set $\sharp\Gamma$. A constellation Φ is *proof-like w.r.t.* an MLL sequent $\vdash \Gamma$ if it is well-formed and $\mathtt{IdRays}\Phi = \sharp\Gamma$.

§70.20 **Example.** A constellation which is proof-like *w.r.t.* $\vdash X_1^\perp \parr X_2^\perp, X_1 \otimes X_2$ is

$$[+(X_1^\perp \parr X_2^\perp)(\mathtt{l}\cdot X), +(X_1 \otimes X_2)(\mathtt{l}\cdot X)] + [+(X_1^\perp \parr X_2^\perp)(\mathtt{r}\cdot X), +(X_1 \otimes X_2)(\mathtt{r}\cdot X)].$$

However, even the wrong linking

$$[+(X_1^\perp \parr X_2^\perp)(\mathtt{l}\cdot X), +(X_1^\perp \parr X_2^\perp)(\mathtt{r}\cdot X)] + [+(X_1 \otimes X_2)(\mathtt{l}\cdot X), +(X_1 \otimes X_2)(\mathtt{r}\cdot X)]$$

is proof-like as well.

§70.21 **Theorem** (Completeness for MLL+MIX)**.** If a constellation $\Phi \in [\![\vdash \Gamma]\!]_\Omega$ is proof-like *w.r.t.* $\vdash \Gamma$, then there exists an MLL+MIX proof-net $\vdash \mathcal{S} : \Gamma$ such that $\Phi = \Phi^{\mathrm{ax}}_{\mathcal{S}}$.

Proof. A proof-like constellation $\Phi \in [\![\vdash \Gamma]\!]_\Omega$ can always be considered as the interpretation of a proof-structure with only axioms; we can then construct a proof-structure $\mathcal{S}$ by considering the union of the latter with $ST(\vdash \Gamma)$ by placing the axioms on the right places in $ST(\vdash \Gamma)$ (at this point, the linking can still be wrong). Since $\Phi \in [\![\vdash \Gamma]\!]_\Omega$ we can use Lemma 70.18 and infer that for all switchings φ of $\vdash \Gamma$ (equivalently, of $\mathcal{S}$), $\mathtt{Test}(\vdash \Gamma)^\varphi = \Phi^\varphi_{\mathcal{S}} \perp \Phi$, excluding "wrong linking". By Corollary 68.21, it follows that $\mathcal{S}$ is acyclic, *i.e.* satisfies the correctness criterion for MLL+MIX. Therefore, $\mathcal{S}$ must be a proof-net of vehicle Φ. □

A complete model of MLL

§70.22 The soundness property actually holds for MLL with the same arguments as for the previous section whether we use $\perp^1$ or $\perp^R$ as orthogonality relation. In this section, we only mean $\perp^1$ or $\perp^R$ whenever $\perp$ is written.

§70.23 **Theorem** (Full soundness for MLL)**.** Let $\vdash \mathcal{S} : \Gamma$ be an MLL proof-net and Ω a basis of interpretation. We have $\mathtt{Ex}(\Phi^{\mathtt{comp}}_{\mathcal{S}}) \in [\![\vdash \Gamma]\!]_\Omega$.

Proof. The idea of the proof is exactly the same as for Theorem 70.17. The only difference is in the axiom case. We need to show that $\mathtt{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}}) = \mathtt{Roots}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}})$ (respectively, $|\mathtt{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}})| = 1$). However, the properties of the basis of interpretation ensures that $\Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}}$ will be orthogonal to Φ_1. Hence $\mathtt{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}}) = \mathtt{Roots}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}})$ (respectively, $|\mathtt{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi^{\mathrm{ax}}_{\mathcal{S}})| = 1$). □

§70.24 The proof of Lemma 70.18 which is essential for the completeness property does not hold anymore because of a minor technical problem. This is because a general sequent $\vdash A_1, ..., A_n$ for A_i being atomic formulas is used for the base case. This is valid for

MLL+MIX proof-nets since we only require acyclicity when testing with the switchings. However, this is not a correct base case for MLL proof-nets which are more demanding by requiring connectedness. We need to start from a single axiom and therefore, induction could be done on the MLL sequent calculus instead by considering provable formulas in MLL. This would be sufficient to get a completeness result. However, instead of restricting to correct formulas, which would identify $[\![\vdash \Gamma]\!]_\Omega$ and a subset of $\mathtt{Tests}(\vdash \Gamma)^\perp$ corresponding to proof-structures, it is sufficient to identify $[\![\vdash \Gamma]\!]_\Omega$ and $\mathtt{Tests}(\vdash \Gamma)^\perp$ directly. We would then have to prove $\mathtt{Tests}(\vdash \Gamma) \subseteq \mathtt{Tests}(\vdash \Gamma)^{\perp\perp}$ which is always true in general [JS21, Proposition 7]. We do so by considering a notion of *strict interpretation*.

§70.25 **Definition** (Strict interpretations)**.** We define the two strict interpretations for a given basis of interpretation Ω and an MLL sequent $\vdash \Gamma$:

$$\langle\!\langle \vdash \Gamma \rangle\!\rangle^1_\Omega = \mathtt{Tests}(\vdash \Gamma)^{\perp^1} \text{ and } \langle\!\langle \vdash \Gamma \rangle\!\rangle^R_\Omega = \mathtt{Tests}(\vdash \Gamma)^{\perp^R}.$$

§70.26 **Theorem** (Completeness for MLL)**.** If a constellation $\Phi \in \langle\!\langle \vdash \Gamma \rangle\!\rangle^R_\Omega$ (respectively $\Phi \in \langle\!\langle \vdash \Gamma \rangle\!\rangle^1_\Omega$) is proof-like *w.r.t.* a provable sequent $\vdash \Gamma$ of MLL, then there exists an MLL proof-net $\vdash \mathcal{S} : \Gamma$ such that $\Phi = \Phi^{\mathrm{ax}}_{\mathcal{S}}$.

Proof. The proof begins like the proof of completeness for multiplicative linear logic extended with the MIX rule (*cf.* Theorem 70.21) and reach the construction of a proof-structure with axioms translated into Φ. Now, $\Phi \in \langle\!\langle \vdash \Gamma \rangle\!\rangle^R_\Omega$ (respectively $\Phi \in \langle\!\langle \vdash \Gamma \rangle\!\rangle^1_\Omega$) implies that, in particular, Φ passes the Danos-Regnier correctness test for MLL (byCorollary 68.21). Therefore, the proof-structure we constructed must be correct. □

§70.27 Notice that if we have a constellation $\Phi \in \langle\!\langle \vdash \Gamma \rangle\!\rangle^X_\Omega$ for some Ω, MLL sequent $\vdash \Gamma$ and $X \in \{1, R\}$, its Danos-Regnier tests $\Phi_1, ..., \Phi_n$ are constellations of $(\langle\!\langle \vdash \Gamma \rangle\!\rangle^X_\Omega)^\perp$. This formalises the intuition in proof-nets that tests are sort of proofs of the dual.

71 The case of multiplicative units

§71.1 Until now, I ignored neutral elements because they were problematic but it is still possible to say few (speculative) things about them.

§71.2 We try to look for behaviours corresponding to neutral elements for $\otimes$ and $\parr$ respectively. It is possible to define a pre-behaviour $\perp\!\!\!\perp$ called a *pole* (*cf.* Section 22) such that $\Phi \perp \Phi'$ if and only if $\mathtt{Ex}(\Phi \uplus \Phi') \in \perp\!\!\!\perp$ for an execution-based orthogonality $\perp$ and $\perp\!\!\!\perp$ must be closed under *anti-evaluation* (*cf.* Section 22), *i.e.* if $\Phi \in \perp\!\!\!\perp$ and $\mathtt{Ex}(\Phi') = \Phi$, then $\Phi' \in \perp\!\!\!\perp$. For instance, if we consider $\perp^R$, then $\perp\!\!\!\perp$ is the set of all constellations normalising into a single uncoloured star. The pole will be useful for a definition of neutral elements.

§71.3 A natural choice of behaviour for the neutral element of $\otimes$ *w.r.t.* $\bot^R$ is the pre-behaviour $\{\emptyset\}$ only containing the empty constellation since $\Phi \uplus \emptyset = \Phi$ for any constellation Φ. Fortunately, it is a behaviour.

§71.4 **Proposition.** The pre-behaviour $\{\emptyset\}$ is a behaviour.

Proof. A constellation of $\{\emptyset\}^\bot$ must self-normalise into the set of its roots since $\emptyset$ has not effect when in interaction with another constellation. We have $\{\emptyset\}^\bot = \perp\!\!\!\perp$. Now, a constellation $\Phi \in \perp\!\!\!\perp^\bot$ is a constellation such that when it interacts with a constellation $\Phi' \in \perp\!\!\!\perp^\bot$, we have $\mathtt{Ex}(\Phi \uplus \Phi') \in \perp\!\!\!\perp$. We can theoretically imagine that Φ has rays linked to Φ but this is impossible because Φ' is self-normalising into an element of $\perp\!\!\!\perp$ by constructing a saturated diagram which cannot be extended and which must be present in the normal form. Actually, Φ must be the empty constellation because otherwise we would get more than the star of roots. Therefore, $\perp\!\!\!\perp^\bot = \{\emptyset\}^{\bot\bot} = \{\emptyset\}$. □

§71.5 **Definition** (One)**.** We define the behaviour $\mathbf{1} := \{\emptyset\} = \perp\!\!\!\perp^\bot$.

§71.6 **Proposition.** We have $\mathbf{A} \otimes \mathbf{1} = \mathbf{A}$ for any behaviour $\mathbf{A}$.

Proof. By definition, we have $\mathbf{A} \otimes \mathbf{1} = \{\Phi_A \uplus \emptyset \mid \Phi_A \in \mathbf{A}\}^{\bot\bot} = \{\Phi_A \mid \Phi_A \in \mathbf{A}\}^{\bot\bot} = \mathbf{A}^{\bot\bot} = \mathbf{A}$. □

§71.7 As for bottom, as usual in linear logic, we define it as $\mathbf{1}^\bot = \perp\!\!\!\perp$.

§71.8 **Proposition.** The pre-behaviour $\mathbf{1}^\bot = \{\emptyset\}^\bot = \perp\!\!\!\perp$ is a behaviour.

Proof. Since it is known that $\mathbf{A}^\bot = \mathbf{A}^{\bot\bot\bot}$ for any behaviour $\mathbf{A}$ [JS21, Corollary 9], it follows that $\mathbf{1}^\bot$ (and thus $\{\emptyset\}^\bot$) is a behaviour. □

§71.9 **Definition** (Bottom)**.** We define the behaviour $\perp := \mathbf{1}^\bot$.

§71.10 **Proposition.** We have $\mathbf{A} \parr \perp = \mathbf{A}$ for any behaviour $\mathbf{A}$ when considering $\bot^R$.

Proof. We have $\mathbf{A} \parr \perp = (\mathbf{A}^\bot \otimes \perp^\bot)^\bot = (\mathbf{A}^\bot \otimes \{\emptyset\}^{\bot\bot})^\bot = (\mathbf{A}^\bot \otimes \{\emptyset\})^\bot = \mathbf{A}^{\bot\bot} = \mathbf{A}$ (since $\mathbf{A}$ is a behaviour). □

§71.11 **Proposition.** We have $\mathbf{A}^{\perp} = \mathbf{A} \multimap \bot$ for any behaviour $\mathbf{A}$ when considering $\perp^R$.

Proof. We have $\mathbf{A} \multimap \bot = \mathbf{A}^{\perp} \parr \bot$. Since $\bot$ is a neutral element for $\parr$, it follows that $\mathbf{A}^{\perp} \parr \bot = \mathbf{A}^{\perp}$. □

§71.12 We defined interactive types for units which correspond to idealised neutral elements (Usage). Now, considering a constellation Φ in the wild, are we able to effectively tell whether it is in $\mathbf{1}$ (respectively $\bot$) or not (Usine).

§71.13 We consider $\perp^R$. In order to tell if $\Phi \in \bot$, we can use the fact that $\bot = \{\emptyset\}^{\perp}$. Hence, it is sufficient to consider the set of tests $\{\emptyset\}$. When testing Φ against the empty constellation $\emptyset$, if we have $\Phi \perp \emptyset$ then $\Phi \in \bot$. As for $\mathbf{1}$, we just need to be able to tell if $\Phi = \emptyset$. This can be done with any constellation of $\mathbf{1}^{\perp} = \bot$.

§71.14 This provides a notion of correct constellations for multiplicative units. However, although they fulfil their role as constellations having the behaviour of neutral elements for multiplicative connectives, it is not quite the *real thing* as they do not exactly correspond to the units of proof-nets. In particular, if we look at the rule for $\perp$, the constant $\perp$ is introduced in a given context Γ to which it is dependent. Hence, it will be disconnected when considering a switching in a correct proof-structure. This breaks the connectedness condition of the Danos-Regnier correctness criterion. The usual hack is to consider jumps (*cf.* Section 30) between $\perp$ nodes and either axioms or 1 nodes to represent the dependency between $\perp$ and its context. Girard's idea [Gir18a, Section 2.1.1] is to encode multiplicative units in second order linear logic because of this non-local dependency (*cf.* Section 44).

72 Discussion: what is a multiplicative proof?

§72.1 We consider the orthogonality $\perp^R$. Correctness tests characterise the shape of proofs and testing is done by checking if all stars collapse to a single star of uncoloured rays by edge contraction. If we look at the test for $\otimes$, it is a 3-ary star with two inputs and one output. It forces any constellation interacting with it to be made of two disjoint connected components. In other words, the test for $\otimes$ *reunites* proof-structures. As for $\parr$, the two tests $\parr_L$ and $\parr_R$ are made of two disjoint parts. They *separates* proof-structures. It seems that MLL proofs are about how some primitive data are organised in terms of reunion/separation.

§72.2 Typically, if we have an axiom (which is also an identity function) $[+1(X), +2(X)]$, then it is possible to connect it with $[+3(X)]$ by using a cut $[-1(X), -3(X)]$ and obtain $[+2(X)]$. But we could also connect it with the cut $[-2(X), -3(X)]$. This emphasizes the equality $A \multimap B = B^{\perp} \multimap A^{\perp}$ of linear logic. But there are some other things we can notice:

- whether we have a proof of $X_1^\perp \parr X_1$ or $A^\perp \parr A^1$ for any multiplicative formula A, their translation is not so different: it is materialised by some binary star $[r, r']$ where r and r' are as complex (in their internal structure) as A is;
- the difference between $X_1^\perp$ and X_1 looks rather artificial in stellar resolution. Both are translated into two different rays. It does not mean that negation does not exist but that it seems to be an external consideration over constellations.

All these remarks about the translation of multiplicative proofs constitute an analysis of MLL through stellar resolution.

§72.3 **Real multiplicatives.** Actually, we did not even get "real" multiplicative proof-structures since multiplicative proof-structures are known to be generic objects in which atomic formulas (which are variables) can be replaced by more complex formulas. Original MLL proof-structures are polymorphic. This shows that we need to add more structure over our interpretation but this is the role of Girard's epidictics. The multiplicatives we got are sort of "first-order" multiplicatives free of external considerations.

§72.4 **Hidden duplication in MLL proof-structures.** In the geometry of interaction, what we do is representing MLL proofs by permutations over a set of atoms (dots, natural numbers, ...) where cuts are represented by partial injections linking some atoms. If we wanted to translate this situation into stellar resolution, we would only need constants. But this is *not* how it works in real proof-structures. In proof-net theory, the situation is more sophisticated: cuts are not links between atoms but links between *conclusion* or *whole* proof-structures, then cuts are duplicated and distributed to atoms. In our stellar interpretation of MLL proofs, this is done by internalising the complex shape of proof-structure directly into terms of rays, then in order to have the cut distributed to its two left premises and two right premises, we must use variables. This shows that technically speaking, MLL proof-net theory is not exactly duplication-free: it hides some non-linear behaviours directly related to how terms of the translation are designed (do we use constants? do we use variables? etc).

Chapter 11

Interpretation of intuitionistic implication

The interpretation of exponentials follows ideas already presented in Section 37. The idea is that we are looking for mechanisms of duplication and erasure in the model of computation itself (stellar resolution in our case). Formulas corresponding to the exponentials of linear logic are then specifications for these primitive behaviours: correctness tests (in the case of Usine) or behaviours (in the case of Usage). Richer computational behaviours related to duplication and erasure can then lead to alternative exponentials. In stellar resolution, duplication occurs when a ray (for instance $+c(X)$) is matchable with several other rays (for instance $-c(1)$ and $-c(2)$). Diagrams will duplicate stars to satisfy the constraints.

Instead of defining full exponentials as in MELL, Girard chooses to restrict the interpretation to *intuitionistic implication* [Gir17, Section 5], hence ? and ! only appear in $?A⅋B$ and $!A⊗B$. New binary exponential connectives $A ⋉ B := ?A⅋B$ and $A ⦶ B := !A⊗B$ are introduced for that purpose. We could equivalently consider intuitionistic implication $A ⇒ B$ and its dual $(A ⇒ B)^{⊥}$ instead.

This restriction is consistent with Girard's point of view (*cf.* Section 44) that full exponentials and multiplicative neutral elements are second-order constructions since from the formula 1 it is possible to define ! with $!A := A ⦶ 1$. Despite this restriction, intuitionistic implication is sufficient to interpret simply typed $λ$-calculus.

In this chapter, I do not detail the interpretation as much as for MLL. In particular, no result of soundness or completeness are stated.

73 MLL with intuitionistic implication (MLL2I)

§73.1 The logic MLL2I defined in this section is simply MELL (*cf.* Section 27) with a different notation. Hence, there is no need to define cut-elimination for sequent calculus. We can simply unfold the notation and we work on MELL.

$$\frac{\vdash \Gamma, \underline{\Delta}, A \quad \vdash \Gamma', \underline{\Delta}', A^\perp}{\vdash \Gamma, \Gamma', \underline{\Delta}, \underline{\Delta}'}\ \text{cut} \qquad \frac{\vdash \Gamma, \underline{\Delta}, A \quad \vdash \Gamma', \underline{\Delta}', B}{\vdash \Gamma, \Gamma', \underline{\Delta}, \underline{\Delta}', A \otimes B}\ \otimes \qquad \frac{\vdash \Gamma, \underline{\Delta}, A, B}{\vdash \Gamma, \underline{\Delta}, A \parr B}\ \parr$$

$$\frac{}{\vdash A, A^\perp}\ \text{ax} \qquad \frac{\vdash \Gamma, \underline{\Delta}}{\vdash \Gamma, \underline{\Delta}, \underline{A}}\ \text{w} \qquad \frac{\vdash \Gamma, \underline{\Delta}, A}{\vdash \Gamma, \underline{\Delta}, \underline{A}}\ \text{d} \qquad \frac{\vdash \Gamma, \underline{\Delta}, \underline{A}, \underline{A}}{\vdash \Gamma, \underline{\Delta}, \underline{A}}\ \text{c}$$

$$\frac{\vdash \underline{\Delta}, A \quad \vdash \Gamma', \underline{\Delta}', B}{\vdash \Gamma', \underline{\Delta}, \underline{\Delta}', A \mathbin{⅋} B}\ ⅋ \qquad \frac{\vdash \Gamma, \underline{\Delta}, \underline{A}, B}{\vdash \Gamma, \underline{\Delta}, A \mathbin{\ltimes} B}\ \ltimes$$

Figure 73.1: Multiplicative linear logic with intuitionistic implication (MLL2I).

MLL2I sequent calculus

§73.2 The connective $?A$ is written $\underline{A}$ and it only applies to formulas at top-level, *i.e.* on whole conclusions of the sequent and not subformulas. Exponentials connectives are ways to handle these underlined formulas.

§73.3 **Definition** (MLL2I pre-formulas)**.** We define the set of *pre-formulas* $\mathcal{F}^{\mathtt{pre}}_{\mathrm{MLL2I}}$ of MLL2I inductively by the following grammar:

$$C, D = X_i \mid X_i^\perp \mid C \otimes D \mid C \parr D \mid C \mathbin{⅋} D \mid C \mathbin{\ltimes} D \qquad (i \in \mathbf{N})$$

§73.4 **Definition** (MLL2I formulas)**.** We define the set of *formulas* $\mathcal{F}_{\mathrm{MLL2I}}$ of 2MLL inductively by the following grammar for $C \in \mathcal{F}^{\mathtt{pre}}_{\mathrm{MLL2I}}$:

$$A, B = C \mid \underline{C}$$

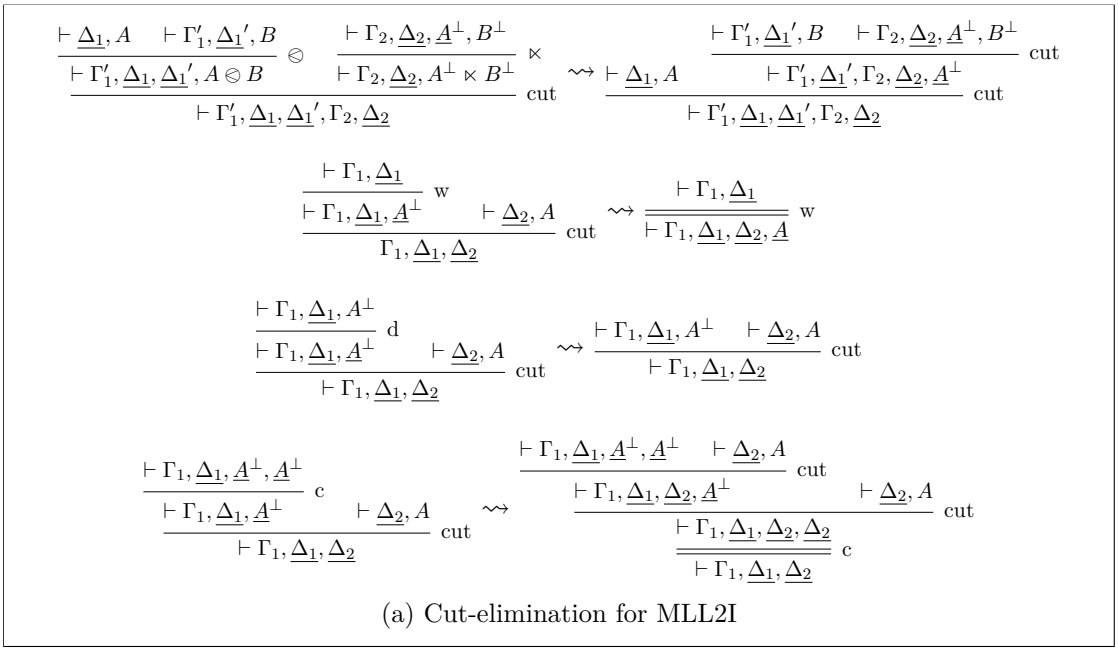
§73.5 The rules of MLL2I, shown in Figure 73.1 are derived from the restriction of MELL to the intuitionistic implication. Notice that the promotion rule is implicit because in the $⅋$ rule, the left premise is $\vdash \underline{\Delta}, A$ which corresponds to $\vdash ?\Delta, !A$. In case we have a cut between a formula $\underline{A}^\perp := ?B^\perp$ (for some B) and another formula A, it is implicit that $A := !B$. It is as if promotion was only accessible from $⅋$.

§73.6 **Definition** (Translation back to MELL formulas)**.** We define a translation of a MLL2I formula A into a corresponding MELL formulas $\lfloor A \rfloor$. If A is a pre-formula C then $\lfloor A \rfloor := \lfloor C \rfloor_{\mathtt{pre}}$. If A is an underlined pre-formula $\underline{C}$ then $\lfloor A \rfloor := ?\lfloor C \rfloor_{\mathtt{pre}}$.

The translation of a pre-formula is defined by:

$$\lfloor X_i \rfloor_{\mathtt{pre}} := X_i \qquad \lfloor X_i^\perp \rfloor_{\mathtt{pre}} := X_i^\perp \qquad \lfloor C \otimes D \rfloor_{\mathtt{pre}} := \lfloor C \rfloor_{\mathtt{pre}} \otimes \lfloor D \rfloor_{\mathtt{pre}}$$

$$\lfloor C \parr D \rfloor_{\mathtt{pre}} := \lfloor C \rfloor_{\mathtt{pre}} \parr \lfloor D \rfloor_{\mathtt{pre}} \qquad \lfloor C \mathbin{\ltimes} D \rfloor_{\mathtt{pre}} := ?\lfloor C \rfloor_{\mathtt{pre}} \parr \lfloor D \rfloor_{\mathtt{pre}}$$



(a) Cut-elimination for MLL2I

$$\lfloor C \oslash D \rfloor_{\mathtt{pre}} := !\lfloor C \rfloor_{\mathtt{pre}} \otimes \lfloor D \rfloor_{\mathtt{pre}}.$$

§73.7 The cut-elimination rules for MLL2I, shown in Figure 73.2a are similar to usual exponential rules. In order to apply cut-elimination on a MLL2I sequent proof, we have to replaced all formulas A by $\lfloor A \rfloor$ and apply the usual rules of MELL. It is easy to show that this replacement preserves validity of the proof.

MLL2I proof-structures

§73.8 The constructors for MLL2I proof-structures are given in Figure 73.3. The difference with the usual presentation with boxes is that we will have links between the left premise of $\oslash$ and other vertices of the proof-structure. These links represent the dependency between a box and its isolated sub-proof-structures of the context (the context $\underline{\Delta}$ on the top of the sequent calculus rule for $\oslash$). Remark that it is not useful to link vertices which are already reachable from the left premise of $\oslash$ since it is already explicit that those vertices are in a box. An exemple of MLL2I proof-structure with such dependencies is given in Figure 73.4.

§73.9 **Definition** (MLL2I proof-structure)**.** An *MLL2I proof-structure* is defined by

$$\mathcal{S} = (V, E, \mathtt{in}, \mathtt{out}, \ell_E, \mathtt{dep})$$

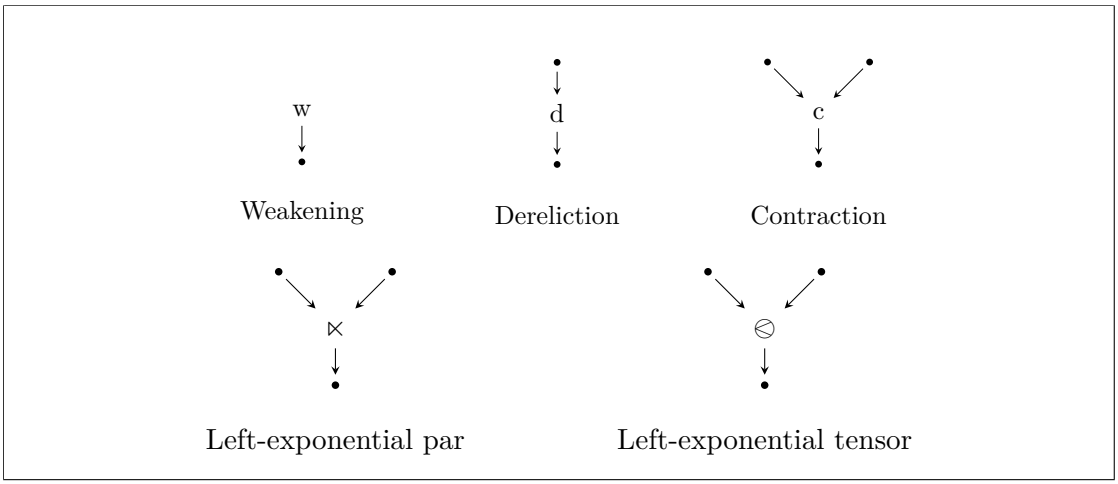


Figure 73.3: MLL2I proof-structures hyperedge constructors.

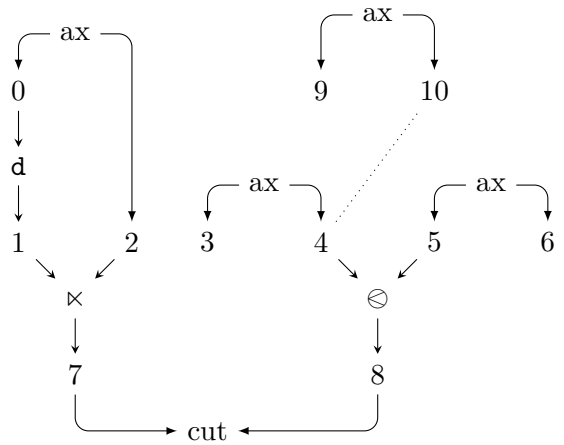


Figure 73.4: Example of MLL2I proof-structure with dependencies.

where $(V, E, \mathtt{in}, \mathtt{out})$ is a ordered directed hypergraph (*cf.* Appendix C),

$$\ell_E : E \to \{\otimes, \parr, \mathrm{ax}, \mathrm{cut}, \ltimes, \invamp, \mathtt{w}, \mathtt{d}, \mathtt{c}\}$$

is a labelling map on hyperedges and $\mathtt{dep} : V \to \mathcal{P}(V)$ associates vertices with left premises of a $\invamp$ hyperedge (it represents dependencies) A proof-structure is subject to these additional constraints:

- hyperedges satisfy the arities and labelling constraints shown in Figure 73.3;
- each vertex must be the target of exactly one hyperedge, and the source of at most one hyperedge;
- cut hyperedges must connect either:
  - the conclusion of a $\parr$ hyperedge with the conclusion of a $\otimes$ hyperedge, or
  - the conclusion of a $\ltimes$ hyperedge with the conclusion of a $\invamp$ hyperedge, or
  - two atoms.

§73.10 **Definition** (Exponential box)**.** Let $\mathcal{S} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E, \mathtt{dep})$ be an MLL2I proof-structure. We define the *(exponential) box* of a vertex by an injective function $\mathtt{box} : V \to \mathcal{P}(V) \times \mathcal{P}(E)$ associating to each left premise v of a $\invamp$ hyperedge (there is some e such that $\mathtt{out}(e) = v$ and $\ell_E(e) = \invamp$) a sub-proof-structure corresponding to all the vertices and edges connected (by non-oriented paths) to v and all vertices of $\mathtt{dep}(v)$.

§73.11 **Convention.** MLL2I proof-structures follow the same conventions and notations as in MLL proof-structures. In particular, $\ltimes$ and $\invamp$ hyperedges are structurally identical to $\parr$ and $\otimes$ hyperedges. Labelled proof-structures are also defined in the same way.

§73.12 **Definition** (MLL2I cut-elimination)**.** Let $\mathcal{S} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E, \mathtt{dep})$ be an MLL2I proof-structure with a cut $e_{\mathrm{cut}} \in E$ such that $\mathtt{in}(e_{\mathrm{cut}}) = (v_1, v_2)$ with both v_1 and v_2 being conclusions of some hyperedges e_1 and e_2 such that $\ell_E(e_1) = (u_1, u_2)$ and $\ell_E(e_2) = (w_1, w_2)$. Assume u_1 is conclusion of some hyperedge $e_\ltimes$ and that we have $\mathtt{box}(w_1) = (V_b, E_b)$. There are several cases depending on $\ell_E(e_\ltimes)$.

◇ **Weakening** Assume $\ell_E(e_\ltimes) = \mathtt{w}$ with $\mathtt{in}(e_\ltimes) = \emptyset$. We erase w_1, all its parents and dependent vertices. The elimination of e_{cut} is a new proof-structure

$$\mathcal{S}' := (V', E', \mathtt{in}', \mathtt{out}, \ell_E, \mathtt{dep}')$$

with $V' := V - \{v_1, v_2, u_1, w_1\} - V_b$, $E' := (E - \{e_{\mathrm{cut}}, e_1, e_2, e_\ltimes\} - E_b) \cup \{e'_{\mathrm{cut}}\}$, $\mathtt{in}'(e'_{\mathrm{cut}}) = (v_2, w_2)$ and $\mathtt{in}'(x) = \mathtt{in}(x)$ otherwise, $\mathtt{dep}'$ is $\mathtt{dep}$ with its domain restricted to $\mathtt{dom}(\mathtt{dep}) - \mathtt{dep}(w_1)$.

◇ **Dereliction** Assume $\ell_E(e_\ltimes) = \mathtt{d}$ with $\mathtt{in}(e_\ltimes) = v_d$. We remove the dependencies of

w_1. The elimination of e_{cut} is a new proof-structure

$$\mathcal{S}' := (V', E', \mathtt{in}', \mathtt{out}, \ell_E, \mathtt{dep}')$$

with $V' := V - \{v_1, v_2\}$, $E' := (E - \{e_{\mathrm{cut}}, e_1, e_2, e_{\ltimes}\}) \cup \{e^1_{\mathrm{cut}}, e^2_{\mathrm{cut}}\}$, $\mathtt{in}'(e^1_{\mathrm{cut}}) = (v_d, w_1)$, $\mathtt{in}'(e^2_{\mathrm{cut}}) = (v_2, w_2)$ and $\mathtt{in}'(x) = \mathtt{in}(x)$ otherwise, $\mathtt{dep}'$ is $\mathtt{dep}$ with its domain restricted to $\mathtt{dom}(\mathtt{dep}) - \mathtt{dep}(w_1)$.

◇ **Contraction** Assume $\ell_E(e_{\ltimes}) = \mathtt{c}$ with $\mathtt{in}(e_{\ltimes}) = (v^1_c, v^2_c)$. We duplicate the dependencies. The elimination of e_{cut} is a new proof-structure

$$\mathcal{S}' := (V', E', \mathtt{in}', \mathtt{out}', \ell'_E, \mathtt{dep}')$$

with $V' := (V - \{v_1, v_2\}) \cup \sigma(V_b \cup \{w_1\})$ with σ renaming vertices with fresh names, $E' := (E - \{e_{\mathrm{cut}}, e_1, e_2, e_{\ltimes}\} - \sigma(E_b)) \cup \{e^{1,1}_{\mathrm{cut}}, e^{1,2}_{\mathrm{cut}}, e^2_{\mathrm{cut}}\}$ with σ renaming edges with fresh names, $\mathtt{in}'(e^{1,1}_{\mathrm{cut}}) = (v^1_c, w_1)$, $\mathtt{in}'(e^{1,2}_{\mathrm{cut}}) = (v^2_c, \sigma(w_1))$, $\mathtt{in}'(e^2_{\mathrm{cut}}) = (v_2, w_2)$ and $\mathtt{in}'(x) = \mathtt{in}(x)$ otherwise, $\mathtt{dep}'(\sigma(w_1)) = \sigma(\mathtt{dep}(w_1))$ and $\mathtt{dep}'(x) = \mathtt{dep}(x)$ otherwise and $\mathtt{out}'$ defined as expected.

74 Simulation of cut-elimination

§74.1 Mechanisms of duplication and erasure are already present in stellar resolution. It is similar to the fact that, in biology, we are observing and speaking about cell division.

§74.2 **Definition** (Exponential basis of representation)**.** The multiplicative basis of representation $\mathbb{B} = (V, F, \mathtt{ar}, \lfloor\cdot\rfloor)$ is extended with a binary function symbol $\bullet \in F$ with $\mathtt{ar}(\bullet) = 2$, considered left associative, *i.e.* $t \bullet u \bullet v := (t \bullet u) \bullet v$, variables $Y_i \in V$ for $i \in \mathbf{N}$ representing boxes and three constants $\mathtt{w}, \mathtt{c}, \mathtt{d} \in F$ such that $\mathtt{ar}(\mathtt{w}) = \mathtt{ar}(\mathtt{c}) = \mathtt{ar}(\mathtt{d}) = 0$. The symbol $\cdot$ has priority over $\bullet$, *i.e.* $t \cdot u \bullet v := (t \cdot u) \bullet v$ and $t \bullet u \cdot v := t \bullet (u \cdot v)$.

§74.3 **Remark** (Inductive definition of MLL2I proof-structures)**.** The inductive definition of proof-structure (*cf.* Remark 66.5) is extended with the two new connectives $\ltimes$ and $\rtimes$. We write $\mathtt{ETens}^{u,v}(\mathcal{S})$ (*resp.* $\mathtt{EPar}^{u,v}(\mathcal{S})$) for a proof-structure $\mathcal{S}$ linked by $\rtimes$ (*resp.* $\ltimes$) hyperedge with premises u and v.

In the same fashion, we introduce inductive constructions for structural rules: $\mathtt{W}(\mathcal{S})$ (nullary), $\mathtt{D}^u(\mathcal{S})$ (unary) and $\mathtt{C}^{u,v}(\mathcal{S})$ (binary).

§74.4 The idea is that given a multiplicative address $u(t)$ of a well-formed vehicle, it can be turned into a non-linear address with $u(t \bullet Y)$ for some fresh variable Y. The right part of $\bullet$ contains exponential information. It can then interact by cut with several copies of a same atom: $v(t \bullet (\mathtt{l} \cdot Y))$ and $v'(t \bullet (\mathtt{r} \cdot Y'))$.

§74.5 **Definition** (Address of an atom)**.** We define the *path address* $\mathtt{pAddr}_{\mathcal{S}}(v)$ of an atom v in a proof-structure $\mathcal{S}$ inductively (*cf.* Remark 66.5):

- if $\mathcal{S} \in \{\mathtt{Ax}_{v,*}, \mathtt{Ax}_{*,v}, \mathtt{W}(\mathcal{S}')\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = X$;
- if $\mathcal{S} = \mathtt{D}^{v}(\mathcal{S}')$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{pAddr}_{\mathcal{S}'}(v) \bullet \mathtt{d}$;
- if $\mathcal{S} = \mathtt{C}^{w_1',w_2'}(\mathcal{S}')$ and $\mathtt{pAddr}_{\mathcal{S}'}(w_i) = t \bullet u$ for $i \in \{1,2\}$ then $\mathtt{pAddr}_{\mathcal{S}}(w_1) = t \bullet (\mathtt{l} \cdot u)$ and $\mathtt{pAddr}_{\mathcal{S}}(w_2) = t \bullet (\mathtt{r} \cdot u)$ such that w_i' is a conclusion with w_i above w_i';
- if $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$ and $v \in V^{\mathcal{S}_i}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{pAddr}_{\mathcal{S}_i}(v)$;
- if $\mathcal{S} \in \{\mathtt{Par}^{v',*}(\mathcal{S}'), \mathtt{Tens}^{v,*}(\mathcal{S}'), \mathtt{EPar}^{v',*}(\mathcal{S}')\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{l} \cdot \mathtt{pAddr}_{\mathcal{S}'}(v)$ such that v' is a conclusion with v above v';
- if $\mathcal{S} \in \{\mathtt{Par}^{*,v'}(\mathcal{S}'), \mathtt{Tens}^{*,v'}(\mathcal{S}'), \mathtt{EPar}^{*,v}(\mathcal{S}')\}$ then $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{r} \cdot \mathtt{pAddr}_{\mathcal{S}'}(v)$ such that v' is a conclusion with v above v';
- if $\mathcal{S} = \mathtt{ETens}^{w_1',w_2'}(\mathcal{S}')$, then for a fresh variable $Y_i \in V$ (representing a box),
  - $\mathtt{pAddr}_{\mathcal{S}}(w_1) = (\mathtt{l} \cdot \mathtt{pAddr}_{\mathcal{S}'}(w_1)) \bullet Y_i$;
  - $\mathtt{pAddr}_{\mathcal{S}}(w_2) = (\mathtt{r} \cdot \mathtt{pAddr}_{\mathcal{S}'}(w_2)) \bullet Y_i$;
  - we update the path address of all $u \in \mathtt{box}(w_i)$ from $\mathtt{pAddr}_{\mathcal{S}'}(u) = t \bullet u$ to $(t \bullet u) \bullet Y_i$

  such that w_i' is a conclusion with w_i above w_i'.
- $\mathtt{pAddr}_{\mathcal{S}}(v) = \mathtt{pAddr}_{\mathcal{S}'}(v)$ otherwise.

The path address to v is uniquely defined *w.r.t.* to a conclusion $c \in \mathtt{Concl}(\mathcal{S}')$ where $\mathcal{S}'$ is $\mathcal{S}$ without cuts, *i.e.* $E^{\mathcal{S}'} = E^{\mathcal{S}} \backslash \mathtt{Cuts}(\mathcal{S})$ and the rest of $\mathcal{S}$ is defined as in $\mathcal{S}'$.

There are two cases depending on if v is dependent (there is some vertex u such that $v \in \mathtt{box}(u)$) or not.

- Assume that v is independent. The *address* of v is then defined as the term $\mathtt{addr}_{\mathcal{S}}(v) := c(\mathtt{pAddr}_{\mathcal{S}}(v))$.
- If there is some u such that $v \in \mathtt{box}(u)$, then the address of v is then defined as the term $\mathtt{addr}_{\mathcal{S}}(v) := \mathtt{pAddr}_{\mathcal{S}}(u)\{X := \mathtt{pAddr}_{\mathcal{S}}(v)\}$.

§74.6 **Box/dependencies in stellar resolution.** The trick which simulates exponential boxes (or dependencies in the case of MLL2I) is that translation of proof-structures will be designed so that cut between a $\ltimes$ and a $\Subset$ hyperedge will apply a cut between the left premise of $\ltimes$ and the left premise of $\Subset$ together will all its dependencies, all thanks to term unification. Hence, all connexions (and thus operations) described by the left premise of $\ltimes$ (erasure, duplication, dereliction or more) are locally applied to all elements of an exponential box (arguments of functions).

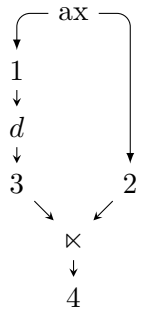
§74.7 **Erasure in stellar resolution.** Whenever a weakening is linked to the left premise of $\Subset$ by a cut, the box of this atom must be erased. Now, how to erase stars in a constellation? There exists two translations of weakening nodes. Girard's one and mine.

- Girard's solution is to not translate weakened atoms. Whenever a cut is connected to where the weakening link should be, since there is a hole with a polarised free ray which cannot be filled, it triggers the erasure of the diagram. We can reproduce this behaviour by using the operator $\lightning$, as explained in Paragraph 52.6. We can choose to do that but then our execution becomes less flexible (this not necessarily a problem);
- my personal solution is to translate weakened atoms u by a star (known as "black-hole" in my thesis):
$$[+u(t), +\omega(X), -\omega(f(X))]$$
where t is the path address of u and ω is a fresh symbol not appearing in the translation of the whole proof-structure. It triggers an infinite loop for any diagram connected to it and hence it becomes impossible to construct a saturated diagram. I do not claim that this is necessarily a better solution. A problem is that we introduce infinity but we can argue that such loops can be easily detected by modifying concrete or interactive execution.

Let v be a weakened atom in a proof-structure $\mathcal{S}$. We write $v^\star$ its corresponding translation (either nothing or the star with black hole). In this thesis, I choose to use my solution, hence:
$$v^\star := [\mathtt{addr}_{\mathcal{S}}(v), +\omega(X), -\omega(f(X))]$$
where ω does not appear in the translation of $\mathcal{S}$.

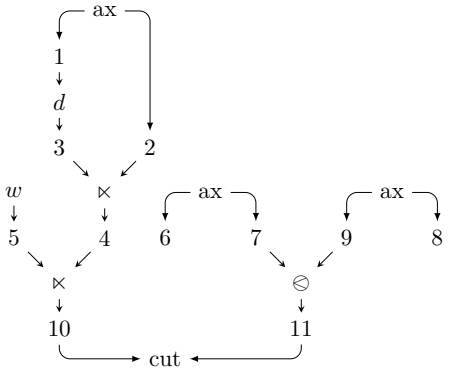
§74.8 **Definition** (Weakened atoms)**.** Let $\mathcal{S} := (V, E, \mathtt{in}, \mathtt{out}, \ell_E, \mathtt{dep})$ be a MLL2I proof-structure. Its set of weakened atoms is defined by $\mathtt{Weak}(\mathcal{S}) := \{u^\bigstar \mid \exists e \in E. \ell(e) = \mathtt{w}, \mathtt{out}(e) = u\}$.

§74.9 **Definition** (Translation of the computational content of a proof)**.** The *vehicle* and the *cuts* of a MLL2I proof-structure $\mathcal{S}$ are respectively defined by the following constellations:
$$\Phi^{\mathrm{ax}}_{\mathcal{S}} := \sum_{e \in \mathtt{Ax}(\mathcal{S})} [\mu(\mathtt{addr}_{\mathcal{S}}(\overleftarrow{e})), \mu(\mathtt{addr}_{\mathcal{S}}(\overrightarrow{e}))] + \sum_{u \in \mathtt{Weak}(\mathcal{S})} v^\bigstar,$$
$$\Phi^{\mathrm{cut}}_{\mathcal{S}} := \sum_{e \in \mathtt{Cuts}(\mathcal{S})} [-\overleftarrow{e}(X), -\overrightarrow{e}(X)].$$
such that $\mu(c(t)) = +c(t)$ when $c = \overleftarrow{e}$ or $c = \overrightarrow{e}$ for some $e \in \mathtt{Cuts}(\mathcal{S})$ (it is related to a cut) and $\mu(x) = x$ otherwise. We define the *computational content* of $\mathcal{S}$ as the constellation $\Phi^{\mathtt{comp}}_{\mathcal{S}} := \Phi^{\mathrm{ax}}_{\mathcal{S}} \uplus \Phi^{\mathrm{cut}}_{\mathcal{S}}$.



$$\rightsquigarrow \quad [+4(\mathtt{l}\cdot X \bullet \mathtt{d}), +4(\mathtt{r}\cdot X)]$$

(a) Proof-structure representing the translation of the identity function of λ-calculus.



$$\rightsquigarrow \quad \begin{array}{l} [+10(\mathtt{l}\cdot X), +\omega(X), -\omega(f(X))]+ \\ [+10(\mathtt{r}\cdot \mathtt{l}\cdot X \bullet \mathtt{d}), +10(\mathtt{r}\cdot \mathtt{r}\cdot X)]+ \\ [+6(X \bullet Y_0), +11(\mathtt{l}\cdot X \bullet Y_0)]+ \\ [+11(\mathtt{r}\cdot X), +8(X)]+ \\ [-10(X), -11(X)] \end{array}$$

(b) Proof-structure representing the translation of the left projection function $\lambda xy.x$ of λ-calculus.

Figure 74.1: Examples of translations of MLL2I proof-structures. With Girard's translation, we would not have $[+10(\mathtt{l}\cdot X), +\omega(X), -\omega(f(X))]$ and $+11(\mathtt{l}\cdot X \bullet Y_0)$ would be linked to nothing through the cut $[-10(X), -11(X)]$.

§74.10 Examples of MLL2I proof-structures and their translation in stellar resolution are given in Figure 74.1. As for MLL, the idea is to execute interpretation of proof-structures in order to simulate their cut-elimination.

§74.11 **Theorem** (Simulation of MLL2I cut-elimination)**.** For an MLL2I proof-net $\mathcal{R}$ such that $\mathcal{R} \rightsquigarrow^* \mathcal{S}$ with $\mathcal{S}$ in normal form, we have $\mathtt{AEx}(\Phi_{\mathcal{R}}^{\mathtt{comp}}) \simeq_S \Phi_{\mathcal{S}}^{\mathtt{ax}}$.

Proof. We sketch the proof without giving details. All multiplicative cases have already been treated in the stellar interpretation of MLL (*cf.* Lemma 67.9). The only new cut case for MLL2I corresponds to a cut between $\ltimes$ and $\multimap$. It makes the left premises interact with the right ones as in a $\otimes/\parr$ cut-elimination. The right premises are multiplicative cases so we focus on the interaction between the left premises which connects boxes/dependencies with structural rules.

- If the left premise of $\ltimes$ is conclusion of a weakening link $\mathtt{w}$, then it is translated with a black hole which prevents any rays $+v(\mathtt{1} \cdot t_i \bullet Y_i)$ coming from the left premise of $\multimap$ (because of the path through the cut) to construct a saturated diagram. Hence all these stars are erased in the normal form. Those rays are exactly the translation of vertices of the box associated with the left premise of $\multimap$.
- If the left premise of $\ltimes$ correspond to derelicted atoms $+u(t_i \bullet \mathtt{d})$ with interacts with left premises $-v(w_j \bullet Y_j)$ of $\multimap$ such that $t_i \bowtie w_j$, then variables Y_j are replaced by $\mathtt{d}$, linearising all rays $-v(w_j \bullet Y_j)$. Hence, all potential of duplication given by the variable Y_j is cancelled with the constant $\mathtt{1}$. This corresponds to a box opening.
- Finally, if the left premise of $\ltimes$ corresponds to n copies $+u_1(t \bullet u_1), ..., +u_n(t \bullet u_n)$, then they all match with the left premises $+v_1(w \bullet Y_1), ..., +v_m(w \bullet Y_m)$ of $\multimap$ such that $t \bowtie w$. This triggers duplications of all the v_i which are considered as part of the same box. Each $+v_i(w \bullet Y_i)$ is duplicated into $+v_1(w \bullet u_1), ..., +v_n(w \bullet u_n)$ where the u_i are copy identifiers induced by trees of contraction.

□

§74.12 **Example.** If we take the proof-structure of Figure 74.1b, the left premise of $\multimap$ will be entirely erased by a weakening link. Then the axiom between 1 and 2 will be connected to the output axiom between 9 and 8. The proof-structure encodes the term $(\lambda x.\lambda y.y)z$ reducing into $\lambda y.y$. As expected, the normal form of the proof structure is the axiom between 1 and 3 where 1 is made non-linear with dereliction.

75 Girard's original correctness criterion

§75.1 In the last paper of geometry of interaction [Gir13a], Girard suggested a correctness criterion for the interpretation of MLL2I in stellar resolution. In the first paper of transcendental syntax [Gir17, Section 5], he updated his criterion to a simpler criterion with switchings (as in Danos-Regnier correctness).

§75.2 This correctness criterion cannot be expressed directly with proof-structures (at least without changing the definition of proof-structures) since it deeply relies on mechanisms of stellar resolution. Moreover, I choose to not give a notion of MLL2I proof-net and consider MLL2I proof-structures as independent from the usual theory of linear logic. Hence we are not trying to simulate a specific correctness criterion. I simply describe Girard's correctness and informally explain its purpose. No adequacy result is stated but Girard sketched a proof in his paper [Gir17, Section 5.7].

§75.3 **Definition** (MLL2I switching)**.** Let $\mathcal{S}$ be a MLL2I proof-structure. A *MLL2I switching* is defined as a MLL switching φ extended with $\varphi(e) \in \{\invamp_X, \invamp_{\mathtt{1}}\}$ when $\ell(e) = \invamp$ and $\varphi(e) \in \{\ltimes_L, \ltimes_R\}$ when $\ell(e) = \ltimes$.

§75.4 **Definition** (MLL2I test)**.** Let $\mathcal{S}$ be a MLL2I proof-structure and φ one of its switchings. The *test* associated with $\mathcal{S}^\varphi$ is the constellation defined by

$$\Phi_{\mathcal{S}}^{\varphi} := \Phi_{\mathcal{S}}^{\mathrm{cut}} \uplus \sum_{v \in V^{\mathcal{S}^\varphi}} v^{\bigstar}.$$

We define the translation $v^{\bigstar}$ of a vertex v conclusion of an hyperedge e as follows:

- if v is conclusion of a $\mathtt{d}$ hyperedge of input itself below some e such that $\ell(e) = \mathrm{ax}$, then
$$v^{\bigstar} = \begin{bmatrix} -\mathtt{addr}_{\mathcal{S}}(v) \\ +v(X \bullet Y) \end{bmatrix};$$
- if $\ell_E(e) = \invamp_X$ and $\mathtt{in}(e) = (u, w)$ then $v^{\bigstar} = \begin{bmatrix} -u(X \bullet X),\ -w(X) \\ +v(X) \end{bmatrix}$;
- if $\ell_E(e) = \invamp_{\mathtt{1}}$ and $\mathtt{in}(e) = (u, w)$ then $v^{\bigstar} = \begin{bmatrix} -u(X \bullet \mathtt{1}),\ -w(X) \\ +v(X) \end{bmatrix}$;
- if $\ell_E(e) = \ltimes_L$ and $\mathtt{in}(e) = (u, w)$ then
$$v^{\bigstar} = \begin{bmatrix} -u(X \bullet Y) \\ +v(X \bullet Y) \end{bmatrix} + \begin{bmatrix} -w(X), -\infty(X) \\ +\infty(X) \end{bmatrix};$$
- if $\ell_E(e) = \ltimes_R$ and $\mathtt{in}(e) = (u, w)$ then
$$v^{\bigstar} = \begin{bmatrix} -u(X \bullet Y) \\ \end{bmatrix} + \begin{bmatrix} -u(X' \bullet Y') \\ \end{bmatrix} + \begin{bmatrix} -w(X) \\ +v(X) \end{bmatrix}$$
where the star $[-u(X' \bullet Y')]$ is not used when X' can be instantiated *exactly* to X, meaning that it is replaced by exactly the variable X (name of variables are taken into account, unlike usual).

The other cases are the same as for the multiplicative case (*cf.* Definition 68.3).

§75.5 Similarly to the multiplicative case, we require that the interaction $\mathtt{Ex}(\overset{+}{\Phi}{}^{\mathrm{ax}}_{\mathcal{S}} \uplus \mathtt{Ex}(\Phi^{\wp}_{\mathcal{S}}))$ between a vehicle and a MLL2I test produces the star of roots $[v_1(X), ..., v_n(X)]$ where $\{v_1, ..., v_n\} \subseteq \mathtt{Concl}(\mathcal{S})$. We require that $v_1, ..., v_n$ correspond to all *linear conclusions* not having addresses of the shape $X \bullet u_i$ for some u_i. My understanding of Girard's restriction is that we allow underlined conclusions in sequents but such conclusions can be erased or duplicated an arbitrary number of times and we cannot require a specific number of copies. Therefore, no conditions are required for non-linear conclusions.

§75.6 **Left-exponential tensor case.** The purpose of the two tests for $\Subset$ is to check that the terms used have the right shape: we need that the left input is a non-linear atom of address $t \bullet Y$ where Y does not appear in t. A non-linear atom $r := +u(t \bullet v)$ passes the two tests $\Subset_X$ and $\Subset_{\mathtt{1}}$ when t matches with X and v with both X and $\mathtt{1}$. First, t must be variable because otherwise, in the two tests, it would be connected to the variable X of $-u(X \bullet X)$ and $-u(X \bullet \mathtt{1})$ which propagates a term to the conclusion which makes impossible to obtain all roots of the form $u'(X)$. Hence we have $r = +u(X' \bullet v)$. We must have the matchings $v \bowtie X$ (hence $v \bowtie X'$ and $X' \bowtie X$) and $v \bowtie \mathtt{1}$. The only way to have $v \bowtie \mathtt{1}$ is that either $v = \mathtt{1}$ or v is a variable.

- If $v = \mathtt{1}$ then when $r = +u(X' \bullet \mathtt{1})$ connected to $-u(X \bullet X)$ of the test of $\Subset_X$, the constant $\mathtt{1}$ is propagated and we produce an output $+u'(\mathtt{1})$ which prevents us to obtain the exact star of roots as normal form. Therefore, we cannot have $v = \mathtt{1}$.
- If $v = Y$, then we have $r = +u(X' \bullet Y)$. In the test $\Subset_{\mathtt{1}}$, it has to match with $-u(X \bullet \mathtt{1})$. Hence $Y = \mathtt{1}$. If $X' = Y$ then, again, the constant $\mathtt{1}$ will be propagated to a conclusion. Finally, Y must be a variable different from X'.

§75.7 **Left-exponential par case (right switching).** The test $\ltimes_R$ is very similar to the test $⅋_R$. The difference is that we would like to cancel $n \in \mathbf{N}$ copies of non-linear atoms of the form $+u(t_i \bullet u_i)$. We would like to ensure that all t_i are variables (the u_i can be exponential copy identifiers). Girard's trick is to take the actual name of variables into account and to consider *coherent constellations* (which are introduced in his second paper on transcendental syntax [Gir16b] and not introduced here) which excludes some substitutions between stars. In our definition of test, this is expressed in the side condition of $\ltimes_R$. Girard's trick enforces t_i to be *exactly* X.

- Imagine that all t_i are X. Then the test $\ltimes_R$ behaves in the same way as a test $⅋_R$ because X' can be instantiated to X and hence the star $[-u(X' \bullet Y')]$ cannot be used.
- If one t_i is different from X then X' cannot be instantiated to X and the two stars $[-u(X \bullet Y)]$ and $[-u(X' \bullet Y')]$ are used. This duplicates the star $+u(t_i \bullet u_i)$ twice which alters the normal form which is expected to be the exact star of roots.

Personally, I do not like the trick used but I never took the time to think about another solution.

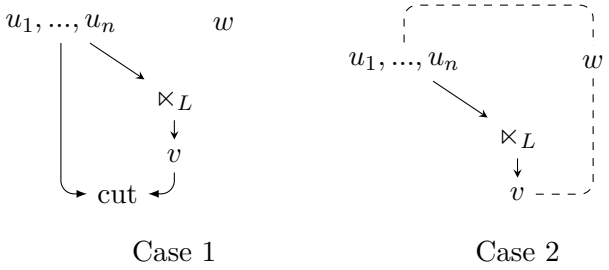


Figure 75.1: Two possible situations for the switching $\ltimes_L$.

§75.8 **Left-exponential par case (left switching).** The test $\ltimes_L$ is more subtle. Girard requires that $\ltimes_L$ is *cancelling*, meaning that any interaction with it normalises into the empty constellation $\emptyset$ (this is the main point which cannot really be represented with usual proof-structures and which takes advantage of the mechanisms of stellar resolution). The problem is that the current flows through the left premise but the left premise, which is non-linear, can be erased or duplicated. I changed Girard's definition by adding a black hole so that it is consistent with the execution I defined. Girard's original definition can be found in his paper [Gir17, Section 5.5]. The effect of the black hole is that any stars reaching $\begin{bmatrix} -w(X), -\infty(X) \\ +\infty(X) \end{bmatrix}$ will never be able to form a saturated diagram. The two possible situations are given in Figure 75.1.

◇ **Case 1** The star corresponding to $\ltimes_L$ is connected to all (possible none) the copies of atoms $u_1, ..., u_n$. Assume one of these atoms has a path leading to v by a cut. We obtain a cycle and infinitely many correct diagrams. Execution is not strongly normalising in this case. Hence, the proof-structure is not correct.

◇ **Case 2** Assume we do not have the cycle of Case 1 and that some atoms among $u_1, ..., u_n$ possibly have a path reaching an atom of w. If we wish to keep connectedness then all stars must be connected. Because of the star

$$[-w(X), +\infty(X), -\infty(X)]$$

of $\ltimes_L$, all stars connected to an atom of w (actually the whole constellation) will be erased because unable to produce a saturated diagram. The normal form is $\emptyset$.

By using the trick of black hole (Girard simply leaves a free polarised ray which destructs the whole diagram with the operator $\lightning$, *cf.* Paragraph 62.5), we are able to keep connectedness (which was the main problems of exponentials). In particular, if there is no u_i, in Case 2 the black hole still erases everything and in Case 1, we loose connectedness since w is isolated.

§75.9 **Example** (Correct proof-structures)**.** The proof-structure representing the identity function in Section 74 has two switchings $\ltimes_L$ and $\ltimes_R$ for its only $\ltimes$ link.

- The first test (right switching) is:

$$\left[\begin{array}{c} -4(\mathtt{l}\cdot X\bullet\mathtt{d}) \\ +3(X\bullet Y)\end{array}\right]+\left[\begin{array}{c} -4(\mathtt{r}\cdot X) \\ +2(X)\end{array}\right]+$$
$$\left[\begin{array}{c} -3(X\bullet Y)\end{array}\right]+\left[\begin{array}{c} -3(X'\bullet Y')\end{array}\right]+\left[\begin{array}{c} -2(X) \\ +4(X)\end{array}\right]+$$
$$\left[\begin{array}{c} -4(X) \\ 4(X)\end{array}\right].$$

  The test is executed into:

$$\left[\begin{array}{c} -4(\mathtt{l}\cdot X\bullet\mathtt{d})\end{array}\right]+\left[\begin{array}{c} -4(\mathtt{r}\cdot X) \\ 4(X)\end{array}\right].$$

  We connected to the vehicle $[+4(\mathtt{l}\cdot X\bullet\mathtt{d}),+4(\mathtt{r}\cdot X)]$ of Section 74, we obtain $[4(X)]$, as expected.

- the second test (left switching) is:

$$\left[\begin{array}{c} -4(\mathtt{l}\cdot X\bullet\mathtt{d}) \\ +3(X\bullet Y)\end{array}\right]+\left[\begin{array}{c} -4(\mathtt{r}\cdot X) \\ +2(X)\end{array}\right]+$$
$$\left[\begin{array}{c} -3(X\bullet Y) \\ +4(X\bullet Y)\end{array}\right]+\left[\begin{array}{c} -2(X),-\infty(X) \\ +\infty(X)\end{array}\right]+$$
$$\left[\begin{array}{c} -4(X) \\ 4(X)\end{array}\right].$$

  It reduces into:

$$\left[\begin{array}{c} -4(\mathtt{l}\cdot X\bullet\mathtt{d}) \\ 4(X)\end{array}\right]+\left[\begin{array}{c} -4(\mathtt{r}\cdot X),-\infty(X) \\ +\infty(X)\end{array}\right].$$

  When connected to the vehicle $[+4(\mathtt{l}\cdot X\bullet\mathtt{d}),+4(\mathtt{r}\cdot X)]$ of Section 74, we obtain a connected constellation which normalises into $\emptyset$ because of the black hole using the symbol ∞. This cancels the test, as expected.

76 Discussion: what is a non-linear proof?

§76.1 Non-linear proofs are proofs which are able to erase or duplicate some logical entities such as occurrences of labels representing formulas in sequent calculus. In proof-net theory, it would correspond to the duplication and erasure of some sub-proof-structures.

§76.2 When expressing duplication and erasure in stellar resolution, we have very general and alogical notions which can be used. Duplication is expressed by the fact that a ray can

be required by several other matchable rays. For instance, if we can have a ray $-1(X)$ compatible with two rays $+1(\mathtt{l})$ and $+1(\mathtt{r})$. The execution will duplicate $+1(X)$ to satisfy the two constraints. Now, what is exponential linear logic in regards to those "natural/primitive" non-linear mechanisms?

§76.3 It appears that exponentials (at least the intuitionistic implication) can be seen as a way to format those non-linear mechanisms. It is *one way* to behave non-linearly. We already knew that we could obtain several non-linear logics by changing the exponential rules of linear logic (to obtain soft linear logic or elementary linear logic for instance) but we can now even go outside of any primitive logical system. The only limit is the primitive computational mechanisms we consider.

§76.4 By formatting, I mean choosing a specific shape for objects. The rays we considered in the chapter are very specific: they have the shape $c(t{\cdot}u)$. This allows for the consideration of nested boxes. But we could also consider alternative non-linear formatting which could not (at least not naturally) exist if we were starting from linear logic as a primitive notion.

Chapter 12

Apodictic experiments

Now that we defined a few interpretations of proof-structures within stellar resolution, we will dive even more into the peculiarities of transcendental syntax. In this chapter, we explore the *apodictic* fragment of linear logic which is linear logic "with no external constraints" or "without system". I refer to Section 44 for conceptual explanations about what Girard means by first-order logic (apodictic).

In apodictic linear logic, we are no more interested in reproducing known objects. We are trying to do (linear) logic with stellar resolution as elementary material by designing constellations and tests which define non-primitive logical concepts.

In his recent unpublished papers, Girard used the expression of "system-free logic" [Gir19, Gir20b]. But in what sense is all this "system-free"? A remark I often encountered is that linear logic is no more "system-free" than other systems: it is itself a logical system. This is indeed correct. Linear logic is a logical system since we suggested an interpretation of it into stellar resolution in the previous chapters but in transcendental syntax it is possible to speak about linear logic with no reference to a system. In particular, there is no notion of logical atomic variable, no primitive connectives and many connectives can co-exist[1].

Everything in this chapter is my own interpretation of Girard's ideas. I sometimes choose to change the original presentation according to my liking and understanding. Everything in this chapter is experimental since no true and complete development of these ideas has been suggested yet.

77 Logical constants

§77.1 According to Girard, proof-structures correspond to the first occurrence of first-order logic in his works. Proof-structures were initially defined with logical atoms as leaves. These atoms are substitutable objects which are all bound to a name. All atoms of

[1] This is similar to the notion of function/procedure coming from modules/libraries in programming. It happens that some imported programs are incompatible but it may also be possible to make them compatible with a bit of hacking (you can also think of LaTeX packages as Girard himself does).

same name can be replaced by a same proof-structure. In particular, proof-nets remain correct under substitution. But this replacement of variables is an *external* consideration of proof-structures. Proof-structures by themselves can be handled without substitutable atoms like non-generic objects (such as natural numbers). By doing so, we necessarily focus on the *individuality* of logical objects, independently of their generic meaning (usual logical objects are representatives of a class instead of individuals).

§77.2 In the stellar interpretation of proof-structures, atoms are simply translated as rays with nothing special about them. Because proof-structures can be handled *as it is*, with atoms which are simply points in space, Girard designed atomic formulas/behaviours to give a first-order status to these points. In the approach Girard calls *morphologism* [Gir18a, Section 1.1.2] (*cf.* Paragraph 39.2), formulas capture the shape of objects. There exists two natural constants we can imagine:

- フ (fu) which contains atomic constellations with a single objective polarised ray and
- ヲ (wo) which contains atomic constellations with a single subjective polarised ray.

Combinations of these constants will characterise proof-structures. For instance, $フ \otimes フ$ will be the type of proof-structures of shape $[r] + [r']$ where r and r' are objective rays. There is no restriction over unpolarised rays (roots).

§77.3 **Remark.** Because our variables were not substitutable in our previous stellar interpretation of MLL, we actually defined "apodictic MLL" or "system-free MLL" and not the "real" MLL coming from the MLL sequent calculus[a].

[a] Yes, after all these pages and more than 3 years of thesis, I did not even interpreted MLL, the simplest fragment of linear logic.

Objective constant

§77.4 The idea (which has been imagined in Girard's second paper on transcendental syntax [Gir16b, Appendix A.3]) is to reproduce the behaviour of interactive partitions (*cf.* Section 37) in which only shape matters, not the semantic content. We would like to see rays as points and stars as sets of points. Points of two constellations can be linked bijectively and we expect the connexion to form a connected and acyclic graph. The logical constant corresponds to a single point. It can only be connected to another single point, hence it is *self-dual*, which is usually contradictory if we interpret orthogonality as a logical negation.

§77.5 We need to distinguish between objective and subjective points but also between linear and non-linear points. These principles did not exist with partitions but since they exist in stellar resolution, we would like to take them into account. In particular, we need adapters to connect rays of *same class*. Typically, it makes no sense in proof-net theory to connect a linear point with a non-linear one (although not recommended, it is still

possible in our case). In the definition below, I suggest a classification of rays. I tried several naive definitions but none were satisfying so the right definition of an apodictic fragment of linear logic is still open.

§77.6 **Definition** (Ray order)**.** The *order* of a polarised ray r is given by its number of distinct variables $\mathtt{ord}(r) := |\mathtt{vars}(r)|$.

§77.7 **Example.** The ray $+1$ is of order 0, the ray $+1(X)$ of order 1 and $+1(X \bullet Y)$ of order 2.

§77.8 **Remark.** My previous interpretation of the weakening rule produces rays of order 1 but it should not be a problem to add a variable to make them of order 2 without affecting their computational behaviour.

§77.9 We first focus on rays of order 0 which are actually sufficient for MLL although we actually used rays of order 1. Rays of order 0 directly translate the interpretation of MLL into finite permutations and partitions over set of natural numbers.

§77.10 **Definition** (Regular adapter)**.** Let $[r, s]$ be an adapter of a constellation Φ. It is *regular* if:

- it is non-animist (*cf.* Definition 48.10), *i.e.* r and s are either both subjective or both objective;
- $\mathtt{ord}(r) = \mathtt{ord}(s)$.

§77.11 **Definition** (Structural orthogonality)**.** Two constellations Φ and Φ' are *structurally orthogonal*, written $\Phi \perp^S \Phi'$ when $\Phi \perp^R \Phi' \uplus \Phi_\mu$ where Φ_μ is a constellation of regular adapters linking rays of Φ to rays of Φ', and $|\pm\,\mathtt{IdRays}\Phi| = |\pm\,\mathtt{IdRays}\Phi'|$.

§77.12 In the case we interpret MLL with rays of order 1, the last requirement ensures that we do not end up in weird situations not corresponding to proof-structures, such as the star $[+1(X)]$ being orthogonal to the star $[-1(\mathtt{l} \cdot X), -1(\mathtt{r} \cdot X), 3(X)]$ not having the shape of a test for atoms: the ray $+1(X)$ is required by two rays $-1(\mathtt{l} \cdot X)$ and $-1(\mathtt{r} \cdot X)$, and it triggers a duplication leading to the expected normal form $3(X)$. In other words, rays (which can be seen as ports) of the two constellations should match in a bijective way. Moreover, it faithfully translates the interpretation of MLL with partitions of a set.

§77.13 **Definition** (Fu)**.** We define the following pre-behaviour called *fu*:

$$\mathcal{F} = \{\{[r, r_1, ..., r_k]\} \mid$$

$$\mathtt{ord}(r) = 0, r \text{ is polarised and objective, and all } r_i \text{ are uncoloured}\}.$$

§77.14 **Proposition.** The pre-behaviour ⅁ is a self-dual behaviour, *i.e.* $⅁ = ⅁^{\bot} = ⅁^{\bot\bot}$.

Proof. Thanks to the condition of non-animism, all rays considered here must be objective because non-animist adapters preserve the class of rays. Consider a constellation $\Phi := [r, r_1, ..., r_k] \in ⅁$. We try to characterise the shape of constellations Φ' in $⅁^{\bot}$. By definition of $\bot^S$, Φ' must have a single coloured ray r'. If there are other stars, they must be uncoloured and $\Phi \uplus \Phi_\mu \uplus \Phi'$ with the right constellation of adapters Φ_μ cannot normalise into a single star as needed for $\bot^S$. Hence Φ' only contains a single star ϕ' containing r'. This star ϕ' can contain as many uncoloured rays as possible and $\Phi \uplus \Phi_\mu \uplus \Phi'$ would still normalise into the star of uncoloured rays. Hence, Φ' must be of the form $[r', r'_1, ..., r'_l]$ for $r'_1, ..., r'_l$ uncoloured. It follows that $\Phi' \in ⅁$. But since it fully characterises $⅁^{\bot}$, we have $⅁ = ⅁^{\bot}$. Since $⅁ = ⅁^{\bot}$ we have $(⅁^{\bot})^{\bot} = ⅁^{\bot} = ⅁$, proving that ⅁ is a behaviour. □

Subjective constant

§77.15 The logical constant ⅄ which is similar to ⅁ except that it contains a subjective ray instead of an objective one.

§77.16 **Definition** (Wo)**.** We define the following pre-behaviour called *wo*:

$$⅄ = \{\{[r, r_1, ..., r_k]\} \mid$$

$$\texttt{ord}(r) = 0, r \text{ is polarised and subjective, and all } r_i \text{ are uncoloured}\}.$$

§77.17 **Proposition.** The pre-behaviour ⅄ is a self-dual behaviour, *i.e.* $⅄ = ⅄^{\bot} = ⅄^{\bot\bot}$.

Proof. Same proof as for ⅁ (*cf.* Proposition 77.14). □

§77.18 In his French unpublished paper *"La logique 2.0"* [Gir18a], Girard suggests to define a test for ⅄ with the brick $\begin{bmatrix} -⅄(-\mathtt{1} \cdot X) \\ ⅄(X) \end{bmatrix}$ which forces the orthogonal to have an internal colour. However, since I define ⅄ as atomic constellation with ray of order 0, the corresponding test has to be $\begin{bmatrix} -⅄(-\mathtt{1}) \\ ⅄ \end{bmatrix}$.

§77.19 **Example.** Formulas only using ⅁ yield objective stars. If we only use ⅄, we have subjective stars. A typical case of animism is $⅁ ⅋ ⅄$ which can be instantiated with $[+1, +2(+c)]$ where $[+1] \in ⅁$ and $[+2(+c)] \in ⅄$.

Shape specification

§77.20 The purpose of logical constants is to specify the shape of proofs. Behaviours only using ⅁ and ∀ as atoms show how the constellations they type must be shaped.

§77.21 **Full tensor.** A big problem we now face is that the tensor does not work anymore on the constants ⅁ and ∀ because they are not disjoint with themselves and hence cannot be combined with the connective ⊗. It is still possible to design a new tensor which behaves as expected. I choose to keep the same symbol to make notations simpler. This tensor is always defined for any constellation.

§77.22 **Definition** (Full tensor)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two behaviours (not necessarily disjoint). We define the *full tensor* by the following behaviour:

$$\mathbf{A} \otimes \mathbf{B} := \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}, \Phi_A \not\bowtie \Phi_B\}^{\perp\perp}$$

where $\Phi_A \not\bowtie \Phi_B$ means that for all rays r_A in Φ_A and r_B in Φ_B, $r_A \not\bowtie r_B$.

§77.23 **Proposition** (Apodictic internal completeness)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be two closed behaviours only using ⅁ and ∀ as atoms. We have that

$$\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}, \Phi_A \not\bowtie \Phi_B\}.$$

Proof. We consider $\Phi_A \uplus \Phi_B \in \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}, \Phi_A \not\bowtie \Phi_B\}$ where Φ_A and Φ_B have disjoint rays. By the definition of $\perp^S$, a constellation $\Phi \in \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}, \Phi_A \not\bowtie \Phi_B\}^{\perp}$ must link rays of $\Phi_A \uplus \Phi_B$ so to have a tree by making bridges between Φ_A and Φ_B. In particular, $\Phi_A \uplus \Phi_B$ has the same number of polarised rays as Φ. If we try to characterise the shape of elements of $\mathbf{A} \otimes \mathbf{B}$ from that, we obtain the constellations separating back Φ in order to obtain disjoint components Φ'_A and Φ'_B, hence exactly the elements of $\Phi_A \uplus \Phi_B \in \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}, \Phi_A \not\bowtie \Phi_B\}$. □

§77.24 The other connectives ⅋ and ⊸ are defined as usual (*cf.* Section 69).

§77.25 **Example.** We have the following constellations together with their shape in proof-net theory:

- ◇ **Atom of proof-structure** $[+1] \in$ ⅁;
- ◇ **Axiom** $[+1, +2] \in (⅁ \otimes ⅁)^{\perp} =$ ⅁ ⅋ ⅁;
- ◇ **Test for axiom** $[-1, 1] + [-2, 2] \in$ ⅁ ⊗ ⅁;
- ◇ **Switching par left** $[-1, 1] + [-2] \in$ ⅁ ⊗ ⅁;
- ◇ **Switching par right** $[-1] + [-2, 2] \in$ ⅁ ⊗ ⅁;
- ◇ **Tensor proof-structure** $[+1] + [+2] \in$ ⅁ ⊗ ⅁.

§77.26 Using the canonical basis of interpretation $\Omega_{フ}(A, i, t) = フ$, it is possible to interpret all MLL formulas by multiplicative combinations of フ. All MLL sequents using フ are inhabited. It is then sufficient to use combinations of フ to speak about constellations in a handy way without caring about the specific terms we use. Given a behaviour written with フ and ヲ, it is possible to extract a proof-structure (like how we synthesize circuits or programs from specifications) by an arbitrary selection of terms satisfying the conditions of $\perp^S$.

§77.27 **Implication and par.** Remark that we have $フ \multimap フ = フ \parr フ$, however this is not true that $\mathbf{A} \multimap \mathbf{B} = \mathbf{A} \parr \mathbf{B}$ in general if we have complex formulas $\mathbf{A}, \mathbf{B} \notin \{フ, ヲ\}$ since negation exchanges $\otimes$ and $\parr$.

Order 0 multiplicative linear logic

§77.28 Because MLL could already be treated with natural numbers, it is sufficient to define MLL tests with rays of order 0. It is even more natural to do so.

§77.29 **Definition** (0rder 0 MLL test)**.** Since we are system-free, we define tests directly on switched formulas (*cf.* Definition 69.15), independently of proof-structures.

- $(A \otimes B)^\bigstar = \begin{bmatrix} -A,\ -B \\ +(A \otimes B) \end{bmatrix}$;
- $(A \parr_L B)^\bigstar = \begin{bmatrix} -A \\ +(A \parr B) \end{bmatrix} + \begin{bmatrix} -B \end{bmatrix}$;
- $(A \parr_R B)^\bigstar = \begin{bmatrix} -A \end{bmatrix} + \begin{bmatrix} -B \\ +(A \parr B) \end{bmatrix}$.

The definition is then extended to switched sequents with:

$$(\vdash A_1, ..., A_n)^\bigstar := \sum_{i=1}^{n} (A_i^\bigstar + \begin{bmatrix} -A_i \\ A_i \end{bmatrix}).$$

§77.30 **Non-linear atoms.** Non-linear atoms used in exponentials can be constructed with $\circledcirc$ and $\ltimes$. For instance, if we have $フ \circledcirc フ$ then the left premise is turned into a non-linear atom of order 2 of a type written !フ. The formula !!フ is of order 3 and so on.

78 Expansional connectives

§78.1 In his unpublished paper *"La logique 2.0"* [Gir18a], Girard imagined alternative exponentials called *expansionals*. Because we are system-free, we are able to freely design connectives by designing tests with constellations. By characterising mechanisms of duplication in stellar resolution, it is then possible to obtain exponential connectives which

could not exist in proof-net theory. In particular, it is possible to imagine several ways to regulate duplication and obtain various sort of exponentials which may be connected to computational complexity. I only give the Usine interpretation (with formula labels and finite tests).

§78.2 Unlike exponentials, these new connectives handle duplication but only for atoms of order 1, while exponentials treat atoms of order ≥ 2. The idea is that from a ray $+c(X)$ of order 1, it is possible to duplicate it by requiring it with two rays $-c(\mathtt{l}\cdot X)$ and $-c(\mathtt{r}\cdot X)$. However, this way of duplicating is weaker than exponentials since we cannot consider nested boxes.

§78.3 Girard's expansionals are dual unary connectives $\downarrow A$ and $\uparrow A$ which can only be used together with multiplicative connectives, as with $\invamp$ and $\ltimes$ (*cf.* Chapter 11). We have the two new constructions of formulas:

- ◇ **Node replication** $\downarrow A \otimes B$;
- ◇ **Node co-replication** $\uparrow A \invamp B$.

It is then possible to define a new sort of implication called *insinuation*:

$$A \rightarrowtail B := \downarrow A \multimap B = \uparrow A^{\perp} \invamp B.$$

§78.4 **Definition** (Expansional switching)**.** We extend the notion of switching with the two following cases:

$$\varphi(\uparrow) \in \{\uparrow_L, \uparrow_R\} \qquad \text{and} \qquad \varphi(\downarrow) \in \{\downarrow_{\mathtt{f}}, \downarrow_{\mathtt{g}}\}.$$

§78.5 **Definition** (Expansional test)**.** We define tests directly on switched formulas (*cf.* Definition 69.15), independently of proof-structures.

- $(\downarrow A \otimes B)^{\bigstar} = \begin{bmatrix} -A(X),\ -B \\ +(\downarrow A \otimes B) \end{bmatrix}$;
- $(\uparrow_L A \invamp B)^{\bigstar} = \begin{bmatrix} -A(X) \\ +(\uparrow A \invamp B) \end{bmatrix} + \begin{bmatrix} -B, -\infty(X) \\ +\infty(X) \end{bmatrix}$;
- $(\uparrow_R A \invamp B)^{\bigstar} = \begin{bmatrix} -A(X) \\ \end{bmatrix} + \begin{bmatrix} -B \\ +(\uparrow A \invamp B) \end{bmatrix}$.

§78.6 The test for $\downarrow$ requires that the left premise of $\downarrow A \otimes B$ is of order 1. The test $\uparrow_L$ allows for a non-linear behaviour of the left premise and $\uparrow_R$ is the same as $\invamp_R$. My tests are different from Girard's tests [Gir18a, Section 2.1.3] because I distinguish between several orders of rays. It is possible that my definition of expansional does not match with Girard's but I will not provide a further exploration of the subject.

§78.7 As we already did previously in Chapter 10, MLL can be defined with rays of order 1. In particular, although permutations and partitions for MLL are more naturally defined with rays of order 0, usual proof-structures are more natural with rays of order 1 because it is the variable X which allows the duplication of cuts. It shows that even in MLL, an elementary duplication occurs.

§78.8 **Remark about our definition of apodictic in regards to exponentials.** There is a problem with the presentation of apodictic linear logic I suggest in this chapter. Because I connect rays of same order, derelicted atoms (coming from the left premise of $\ltimes$) and non-linear atoms corresponding to the output of a box (left premise of $\oslash$) are technically incompatible. I have no satisfying solution for that. We could say that all terms of address $t \bullet u$ are of same order and hence both a derelicted and other non-linear atoms would be of same order. But this we would give too much credit to the symbol $\bullet$. All this does not really matter since I simply want to present the idea so that it can be developed. It does not need to be perfect.

79 Visibility and non-classical truth

§79.1 Without diving into philosophical questions, what is truth? Girard suggested to characterise truth with two technical conditions. It is a predicate over formulas such that:

- there is a special behaviour **0** which is not true;
- it is preserved by execution (cut-elimination).

We can agree or disagree for philosophical reasons but this is still a way to start discussions in formal logic.

§79.2 **Classical truth.** Girard's truth is a property of *constellations* before even being a property of formulas. A classical way to define truth is to say that a constellation Φ is *true* when it passes some given tests (typically Danos-Regnier tests). Hence, truth depends on some subjective tests but also on what it means to be a "correct proof". Truth matches with provability providing we have a mature definition of proof. We say that a behaviour (representing a formula) is true when it contains a true constellation. Correctness criteria, even the weirdest one we can design, yield different notions of truth. We can say that truth is related to the *objective* relation between an entity and a specification, which although subjective, serves a purpose or represents the expression of a will.

§79.3 **Non-classical truth.** Whereas there is a classical way to define truth, there is also *non-classical ways* which challenge common sense. In his fourth paper on transcendental syntax [Gir20a], Girard suggested a notion of truth called *visibility* which implements the intuition of hidden and visible files (*cf.* Paragraph 40.6). But how to design a notion of truth? We should make clear what we absolutely want to make true and false (and the interesting part is to leave some gap on which there is no restriction). Typically,

we would like constellations of binary stars to be true since they correspond to axioms and we would like a special behaviour **0** to be false. The constant **0** is defined by using subjective rays which have a role in epidictics.

§79.4 **Epidictic and visibility.** As explained in Section 44, Girard's second-order logic uses a vehicle together with a mould for existential witnesses. The vehicle is defined with *objective* rays whereas the mould is defined with *subjective* ones. Given a constellation "in the wild", it is correct when it has no animist star: the objective and subjective part can be separated. To be consistent with this interpretation, we would like to make animist star a reason for falsity. Girard chose to base visibility on the Euler-Poincaré invariant which is a necessary condition for Danos-Regnier correctness but not a sufficient one[2].

§79.5 **Proposition** (Euler-Poincaré invariant)**.** Let (V, E) be a bipartite multigraph (*cf.* Appendix C). If $|Cy|$ is the number of minimal cycles and $|CC|$ is the number of connected components, then we have $|V| - |E| + |Cy| - |CC| = 0$.

§79.6 Since we are interested in trees, we have $|CC| = 1$ and $|Cy| = 0$. We can divide V into V_1 and V_2 since it is bipartite. Hence:

$$2(|V_1| + |V_2| - |E|) = 2|V_1| + 2|V_2| - 2|E| = (2|V_1| - |E|) + (2|V_2| - |E|) = 2.$$

§79.7 Considering that constellations corresponding to proof-structures and tests are related to partitions, the above equations induce a weight on objective constellations. A weight for subjective rays is designed in order to be consistent with our notion of truth.

§79.8 **Definition** (Weight of a star)**.** The *weight of a star* ϕ with objective rays o_i and subjective rays s_j is defined by:

- $\omega([o_1, ..., o_n]) := 2 - n$ when it is fully objective, and
- $\omega([o_1, ..., o_n, s_1, ..., s_m]) := -n$ otherwise.

§79.9 **Definition** (Weight of a constellation)**.** The *weight of a constellation* Φ is defined by:

$$\omega(\phi_1 + ... + \phi_n) := \sum_{i=0}^{n} \omega(\phi_i).$$

[2]It is not clear why he chose this invariant in particular but my current understanding is that: it satisfies the conditions of truth and it is odd enough. It also seems that he developed this truth notion to express Peano arithmetic in apodictic linear logic so this may be his original purpose. I once asked him why he did things like that and told him (probably impolitely) that is seemed rather arbitrary for something related to the foundations of logic. I don't know if I hurt his feelings but he ironically told me that Tarski was probably better since he calls true what is true.

$$\omega(\mathbf{フ}) := 1 \qquad \omega(\mathbf{ヲ}) := 0 \qquad \omega(\mathbf{A} \otimes \mathbf{B}) := \omega(\mathbf{A}) + \omega(\mathbf{B})$$
$$\omega(\mathbf{A}^\perp) := 2 - \omega(\mathbf{A}) \qquad \omega(\mathbf{A} \multimap \mathbf{B}) := \omega(\mathbf{B}) - \omega(\mathbf{A})$$
$$\omega(\mathbf{A} \parr \mathbf{B}) := \omega(\mathbf{A}) + \omega(\mathbf{B}) \text{ (if both } \mathbf{A} \text{ and } \mathbf{B} \text{ contain } \mathbf{ヲ})$$
$$\omega(\mathbf{A} \parr \mathbf{B}) := \omega(\mathbf{A}) + \omega(\mathbf{B}) - 2 \text{ (otherwise)}$$
$$\omega(\mathbf{A} \ltimes \mathbf{B}) := \omega(\mathbf{A} \parr \mathbf{B}) \qquad \omega(\mathbf{A} \circledcirc \mathbf{B}) := \omega(\mathbf{A} \otimes \mathbf{B})$$

Figure 79.1: Weights of behaviours.

§79.10 **Example.** For instance, for a constellation $\Phi \in \mathbf{フ} \otimes \mathbf{フ}$, we would have $\omega(\Phi) = 2 \times 2 - 2 = 2$ and $\omega(\Psi) = 2 \times 1 - 2 = 0$ for $\Psi \in \mathbf{フ} \parr \mathbf{フ}$. By the Euler-Poincaré invariant, it is then expected that the sum of the weight of two orthogonal constellations is 2.

§79.11 **Definition** (Weight of a behaviour)**.** We define the *weight of a behaviour* $\mathbf{A}$ as the maximal weight of its constellations: $\omega(\mathbf{A}) := \mathtt{max}\{\omega(\Phi) \mid \Phi \in \mathbf{A}\}$.

§79.12 We obtain the weights of Figure 79.1 for behaviours. Now that we have a notion of weight, it is possible to define a behaviour $\mathbf{0}$ which is invisible (false). Interestingly, $\mathbf{0}$ is not empty, unlike the usual denotational interpretations where $\mathbf{0}$ is empty because it has no proofs. We have constellations but none are correct. In particular, this gives an interactional content to contradictions.

§79.13 **Definition** (Zero)**.** The behaviour *zero* is defined by $\mathbf{0} := (\mathbf{フ} \parr \mathbf{ヲ}) \circledcirc \mathbf{ヲ}$.

§79.14 We can finally define visibility then prove that it corresponds to a notion of truth by satisfying the conditions given in the introduction of this section.

§79.15 **Definition** (Visibility)**.** A constellation Φ is *visible* (true) when $\omega(\Phi) \geq 0$. The definition is extended to behaviours by saying that a behaviour $\mathbf{A}$ is visible when $\omega(\mathbf{A}) \geq 0$.

§79.16 Remark that, as expected, constellations of binary stars are visible because all binary objective stars are of weight 0 and the sum of all weights of the constellation is still 0. As for animist stars, by definition they have a weight at most -1. For every additional objective ray, the weight is lowered. For instance: $\omega(\mathbf{フ} \parr \mathbf{ヲ}) = -1$, $\omega(\mathbf{フ} \parr \mathbf{フ} \parr \mathbf{ヲ}) = -2$ and $\omega(\mathbf{フ} \parr \mathbf{フ} \parr \mathbf{フ} \parr \mathbf{ヲ}) = -3$. Oddly enough, it is possible to make an animist constellation visible by using a tensor with visible constellations. For instance, we have $\omega((\mathbf{フ} \parr \mathbf{ヲ}) \otimes \mathbf{フ}) = 0$. We can also remark that all formulas using only $\mathbf{ヲ}$ are always visible.

§79.17 **Proposition.** Visibility defines a truth notion: **0** is invisible and it is closed under cut-elimination for proof-structures.

Proof. We have $\omega(\mathbf{0}) = \omega((\mathbf{フ} \parr \mathbf{ヲ}) \oslash \mathbf{ヲ}) = \omega(\mathbf{フ} \parr \mathbf{ヲ}) + \omega(\mathbf{ヲ}) = \omega(\mathbf{フ} \parr \mathbf{ヲ}) = \omega(\mathbf{フ} \parr \mathbf{ヲ}) = \omega(\mathbf{フ}) + \omega(\mathbf{ヲ}) - 2 = 1 - 2 = -1 < 0$. Notice that it is the presence of animist star which makes **0** invisible. As for cut-elimination, consider that we have Φ made of binary stars only (since proof-structures are translated only with binary stars). We then have $\omega(\Phi) = 0$, meaning that Φ is visible. However, cut-elimination only reunites binary stars and hence preserves binarity. We have $\omega(\mathtt{AEx}(\Phi)) = 0$ which makes $\mathtt{AEx}(\Phi)$ visible. □

§79.18 Now that we defined the additive contradiction **0**, it is possible to define the additive constant top by duality.

§79.19 **Definition** (Top)**.** The behaviour *top* is defined by $\top := \mathbf{0}^{\perp} = (\mathbf{フ} \otimes \mathbf{ヲ}) \ltimes \mathbf{ヲ}$.

§79.20 **Proposition.** $\top$ is visible.

Proof. We have $\omega((\mathbf{フ} \otimes \mathbf{ヲ}) \ltimes \mathbf{ヲ}) = \omega(\mathbf{フ} \otimes \mathbf{ヲ}) + \omega(\mathbf{ヲ}) = \omega(\mathbf{フ} \otimes \mathbf{ヲ}) = \omega(\mathbf{フ}) + \omega(\mathbf{ヲ}) = \omega(\mathbf{フ}) = 1$. Hence $\top$ is visible. □

§79.21 To be sure that we defined additive constants in the right way, we should check that they correspond to neutral elements for the additive connectives $\oplus$ and & but since they are defined in second-order, we leave these definitions unverified.

§79.22 We define a notation for tensor and par of logical constants [Gir20a, Section 4.1]. This definition will be useful to make explicit the weirdness of this notion of truth.

§79.23 **Definition** (Fu sequence)**.** We define $\mathbf{フ}_n$ for $n \in \mathbf{Z}$ by:

- $\mathbf{フ}$ when $n = 1$;
- $\mathbf{フ} \otimes \mathbf{フ}_{n-1}$ when $n > 1$;
- $\mathbf{フ} \parr \mathbf{フ}_{n+1}$ when $n < 1$.

§79.24 **Definition** (Wo sequence)**.** We define $\mathbf{ヲ}_n$ for $n \in \mathbf{N}$ by:

- $\mathbf{ヲ}$ when $n = 0$;
- $\mathbf{フ}_n \otimes \mathbf{ヲ}$ otherwise.

§79.25 As presented in the fourth paper of transcendental syntax [Gir20a, Section 4.3], Figure 79.2 illustrates a "truth table" with counter-intuitive truth values.

$\mathbf{A}$	$\mathbf{B}$	$\mathbf{A} \otimes \mathbf{B}$	$\mathbf{A} \parr \mathbf{B}$	$\mathbf{A}^\perp$
1	1		0	1
0	1	1	0	

Figure 79.2: Odd cases of non-classical truth. The value 1 represents visibility and 0 invisibility.

◇ **Case of tensor** If we have $\mathbf{A} := \text{ヲ} \parr \text{ヲ}$ (which is not visible since $\omega(\text{フ} \parr \text{フ}) = 0 + 0 - 2 = -2$) and $\mathbf{B} := \text{フ} \otimes \text{フ}$ (which is visible since $\omega(\text{フ} \otimes \text{フ}) = 1 + 1 = 2$), then we have $\omega(\mathbf{A} \otimes \mathbf{B}) = -2 + 2 = 0 \geq 0$ hence $\mathbf{A} \otimes \mathbf{B}$ is visible.

◇ **Case of par** As for $\parr$, we have $\text{フ}_0 = \text{フ} \parr \text{フ}$ of weight 0 (visible) but $\text{フ}_0 \parr \text{フ}_0$ of weight $0 + 0 - 2 = -2$ is invisible. However, if we have $\mathbf{A} := \text{フ}_{-1} = \text{フ} \parr \text{フ} \parr \text{フ}$ (weight -1) which is invisible and $\mathbf{B} := \text{フ}_1 = \text{フ}$ (weight 2) which is visible, then $\mathbf{A} \parr \mathbf{B}$ is of weight $-1 + 2 - 2 = -1$, hence invisible.

◇ **Case of dual** Finally, for the orthogonal, if we have $\mathbf{A} := \text{フ}_0 = \text{フ} \parr \text{フ}$ (visible), then $\mathbf{A}^\perp = \text{フ} \otimes \text{フ} = \text{フ}_2$, which is visible as well. Hence, orthogonality does not exchange visibility although it satisfies the condition of truth we wish for.

80 System-free arithmetic on relative numbers

§80.1 It is possible to define relative numbers as behaviours by using the logical constants.

§80.2 **Definition** (Encoding of relative numbers)**.** Given $p \in \mathbf{Z}$, we define its encoding by a behaviour $[\![p]\!]$ such that:

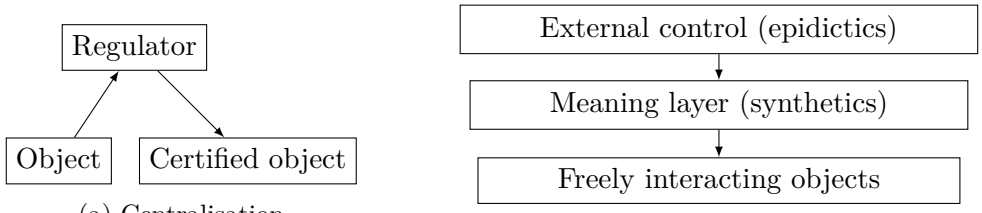
- $[\![0]\!] := \text{ヲ}$;
- $[\![p]\!] := \text{フ}_p \otimes \text{ヲ}$ when $p > 0$;
- $[\![p]\!] := \text{フ}_{p+2} \parr \text{ヲ}$ when $p < 0$.

§80.3 **Example.** We have $[\![-1]\!] := \text{フ} \parr \text{ヲ}$, $\top = [\![-1]\!] \Rightarrow [\![0]\!]$ and $\mathbf{0} = \top^\perp = [\![-1]\!] \oslash [\![0]\!]$. Also remark that $[\![-n]\!] = [\![n]\!]^\perp$.

§80.4 We present some properties which will be useful to reason about this interpretation of relative numbers.

§80.5 **Proposition.** The following statement holds:

1. $\mathbf{A} \equiv \text{フ}_{\omega(\mathbf{A})}$ (this implies that $\omega([\![p]\!]) = p$, *i.e.* encoded number is reflected in weight of encoding). The notation $\equiv$ is used for linear equivalence;
2. $\text{フ}_m \otimes \text{ヲ}_n \equiv \text{ヲ}_{m+n}$ with $m, n > 0$;



(a) Centralisation

(b) Decentralisation

Figure 81.1: Models of regulation for computational objects.

3. $\text{ℱ}_{n+2} \parr \text{ℌ} \equiv \text{ℌ}_n$ with $n < 0$ (negative integers are equivalent to some ℌ_n);
4. $\text{ℱ}_m \otimes \text{ℱ}_n \equiv \text{ℱ}_{m+n}$, $\text{ℱ}_m \parr \text{ℱ}_n \equiv \text{ℱ}_{m+n-2}$ and $\text{ℱ}_n^\perp \equiv \text{ℱ}_{2-n}$ (combinations of ℱ can be contracted);
5. $\text{ℌ}_m \otimes \text{ℌ}_n \equiv \text{ℌ}_m \parr \text{ℌ}_n \equiv \text{ℌ}_{m+n}$ and $\text{ℌ}_n^\perp \equiv \text{ℌ}_{-n}$ (combinations of ℌ can be contracted);
6. ℱ_n and ℌ_n are invisible for $n < 0$.

Proof. Stated in Girard's fourth paper on transcendental syntax [Gir20a, Section 4]. □

§80.6 **Arithmetic operations.** By relying on the properties of Proposition 80.5, it is possible to do elementary arithmetic operations on relative numbers:

- the constant ℱ corresponds to $[\![1]\!]$;
- $[\![n+1]\!] \equiv \text{ℱ} \otimes [\![n]\!]$ corresponds to the successor function;
- $[\![n-1]\!] \equiv \text{ℱ} \multimap [\![n]\!]$ corresponds to the predecessor function;
- $[\![n+m]\!] \equiv [\![n]\!] \otimes [\![m]\!]$ corresponds to addition;
- $[\![n-m]\!] \equiv [\![n]\!]^\perp \parr [\![m]\!] = [\![n]\!] \multimap [\![m]\!]$ corresponds to subtraction.

§80.7 **Example.** Equivalences are inferred from the properties of Proposition 80.5.

- $[\![2+3]\!] = [\![2]\!] \otimes [\![3]\!] = (\text{ℱ} \otimes \text{ℱ} \otimes \text{ℌ}) \otimes (\text{ℱ} \otimes \text{ℱ} \otimes \text{ℱ} \otimes \text{ℌ}) \equiv \text{ℌ}_2 \otimes \text{ℌ}_3 \equiv \text{ℌ}_5 \equiv \text{ℱ} \otimes \text{ℱ} \otimes \text{ℱ} \otimes \text{ℱ} \otimes \text{ℱ} \otimes \text{ℌ} = [\![5]\!]$.
- $[\![2-3]\!] = [\![2]\!]^\perp \parr [\![3]\!] = (\text{ℱ} \otimes \text{ℱ} \otimes \text{ℌ})^\perp \parr (\text{ℱ} \otimes \text{ℱ} \otimes \text{ℱ} \otimes \text{ℌ}) = (\text{ℱ} \parr \text{ℱ} \parr \text{ℌ}) \parr (\text{ℱ} \otimes \text{ℱ} \otimes \text{ℱ} \otimes \text{ℌ}) \equiv \text{ℌ}_{-2} \parr \text{ℌ}_3 \equiv \text{ℌ}_1 \equiv \text{ℱ} \parr \text{ℌ} = [\![-1]\!]$.

81 Discussion: anarchy

§81.1 Apodictics is a state where computational objects express logic without the need for external regulation or control. However, this does not mean that external control should

be excluded: it becomes a choice, something put over what is unregulated. This makes system-free formalisms such as apodictic transcendental syntax (Figure 81.1a) necessarily more flexible than system-bounded formalisms such as usual logical systems (Figure 81.1b). Personally, the apodictic part of logic reminds me of decentralised systems (typically, decentralised communication or cryptocurrencies).

§81.2 **Is apodictics sufficient.** Apodictics is about decentralised logic. Constellations live their live and the synthetics describes their behaviour with formulas or label them with tests. But is is sufficient to speak about logic? I do not have an answer. I believe that it may be sufficient but not satisfying. If logic is about an analysis and study of interaction then we first need a space of interacting entities then the ability to put words on it (the synthetics). But it is not all that matters. Typically, as imagined in Section 60, external control may be related to efficiency for computational power (algorithmic optimisation) but probably also for human readability and workability (tell me if I am wrong if I say that natural deduction is more convenient than reasoning with stellar resolution).

§81.3 Epidictics is exactly what apodictics lacks to express natural deduction. But adding epidictics is not exactly a "return to the outdated traditions". Instead, it is a layer *over* a space of freely interacting computational objects with freedom at the bottom and authority at the top.

Chapter 13

Epidictic experiments

Epidictics, discussed in Section 44, is the part of logic which is generic/substitutable. For instance, in proof-structures, atoms correspond to variables which can be replaced by other proof-structures. There is currently no true theory of epidictics in the context of transcendental syntax. However, it is possible to sketch and think about some ideas already presented by Girard [Gir18b].

82 Genericity of proof-structures

§82.1 **Variables.** Two things have to be distinguished. First, the proof-structures we defined in stellar resolution are *apodictic*. They are self-sufficient objects with formula leaves which are simply translated as rays such as $+1$ for an atom 1. There is nothing special about such rays. But it is not exactly the proof-structures which are defined in the usual theory of proof-nets (or even proofs in proof theory) which are more complex. We usually require that atoms are substitutable, which means that atoms should actually be seen as variables which are *universally* quantified. Hence, to speak about usual proofs, we must think about an interpretation of quantifiers.

§82.2 For conceptual reasons explained in the third article of transcendental syntax [Gir18b, Section 5.1], alternative rules for quantifiers are considered. These rules should *declare* variables and then rules only manipulate variables which are declared[1]. In particular, it

[1]Some people think that Girard attributes this trick to himself but this does not seem to be the case. He is probably aware that such hacks already existed but he rarely quote other people in his recent papers.

$$\frac{\beta_1, ..., \beta_n, \alpha \vdash \Gamma, A}{\beta_1, ..., \beta_n \vdash \Gamma, \forall \alpha.A}\ \forall \qquad\qquad \frac{\beta_1, ..., \beta_n \vdash \Gamma, \{\alpha := \sigma\}A}{\beta_1, ..., \beta_n \vdash \Gamma, \exists \alpha.A}\ \exists$$

Figure 82.1: Rules for quantifiers with declared variables. The left part of the sequent contains variables which are declared. We must have $\mathtt{vars}\sigma \subseteq \{\beta_1, ..., \beta_n\}$ in $\exists$ and $\alpha \notin \mathtt{vars}\Gamma$ in $\forall$.

is not allowed to use a variable as a mathematical entity by itself which is assumed to exist and have a content[2]. The rules for quantifiers are given in Figure 82.1.

§82.3 **Universal quantification.** In order to express the fact that a variable is generic, it must fit into the picture for any shape it can take. In the third article on transcendental syntax [Gir18b, Section 5.3], Girard suggests a definition in the case of MLL. A variable α is subject to three switchings $\forall_{\mathtt{id}}, \forall_{\otimes}$ and $\forall_{⅋}$ for all the shapes α and its dual can have: that is, either an irreducible atom, a par or a tensor. Since it seems that we cannot distinguish between an atom and its negation, it is considered in the transcendental syntax that it must be a "human" choice (in the first paper of transcendental syntax, Girard even propose switchings with tests in order to enforce the presence of duality). By imposing dual atoms, we say that cuts (for instance) cannot connect any atoms but that we want some *specific* connexions. We will hence have adapters allowing the connexion of these three $\forall$ tests with atoms and their dual.

§82.4 **Existential quantifiers.** A proof of existential formula turns a sequent $\vdash \Gamma, \{\alpha := \sigma\}A$ (where α is a variable and σ a term) into a sequent $\vdash \Gamma, \exists\alpha.A$. If we try to deconstruct such a proof, we obtain three parts: the specification $\vdash \Gamma, \exists\alpha.A$ which we want to prove, the proof itself which corresponds to a computational entity and the existential witness σ located in the proof. These three components are translated into: a set of tests, a vehicle containing the computational content of σ and another set of tests for σ. The existential witness (which cannot be inferred from the formula alone) comes with the vehicle together with its own tests. The materialisation of these tests for existential witness in stellar resolution is called *mould*. They are "pre-integrated" tests checking that the existential witness has the right shape.

§82.5 When translating application of existential rules in transcendental syntax, we must also handle the substitution used in the premise. This is done by a hack which encodes sort of explicit substitutions in stellar resolution [Gir18b, Section 5.2].

§82.6 **Encoding of terms.** As for the encoding of terms needed to represent existential witnesses. Girard's solution is to encode terms by multiplicative combinations of ⅂ and equality between terms by linear equivalence. Applications of functions such as $f(a)$ will be treated as a pair (f, a). We then require that the encoding is injective so that $f(t) = f(u)$ implies $t = u$. Actually, there are several solutions providing it satisfies the previous requirement. It does not matter which solution we choose since it is only a matter of representation[3].

[2] This is actually what is done in some programming languages such as the C language. Variables are first declared (they exist) then defined (we give them a value).

[3] As if we had the choice between a binary or hexadecimal encoding of integers.

83 Usage interpretation of second-order linear logic

§83.1 In the Usage interpretation, we are interested in the formation of behaviours. As in usual realisability interpretations, quantifiers can be handled by infinite unions and intersections (in Usage, we do not fear infinity). We allow variables $X, Y, Z, ...$ to appear in behaviours. A behaviour is *valid* only when it contains no variables. It is then possible to substitute a behaviour variable X for a behaviour $\mathbf{T}$.

§83.2 **Definition** (Epidictic architecture)**.** An *epidictic architecture* is a countable set of behaviours $\mathfrak{E}$ closed by a given set of connectives C, by adjunction and by composition, that is:

- for all $* \in C$ of arity n and $\mathbf{A}_1, ..., \mathbf{A}_n \in \mathfrak{E}$, we have $*(\mathbf{A}_1, ..., \mathbf{A}_n) \in \mathfrak{E}$ where $*(\mathbf{A}_1, ..., \mathbf{A}_n)$ is the application of the n-ary connective $*$ with premises $\mathbf{A}_1, ..., \mathbf{A}_n$;
- for all $\mathbf{F}, \mathbf{A}, \mathbf{B} \in \mathfrak{E}$, we have $\mathbf{F} \perp \mathbf{A} \otimes \mathbf{B} = \mathbf{F}(\mathbf{A}) \perp \mathbf{B}$ where

$$\mathbf{F}(\mathbf{A}) = \{\mathtt{Ex}(\Phi_F \uplus \Phi_A) \mid \Phi_F \in \mathbf{F}, \Phi_A \in \mathbf{A}\};$$

- if $\mathbf{A} \multimap \mathbf{B} \in \mathfrak{E}$ and $\mathbf{A} \in \mathfrak{E}$, then $\mathbf{B} \in \mathfrak{E}$.

We may need other conditions to obtain a satisfying definition but we will limit ourselves to those as we will not dive into the details.

§83.3 The definitions for universal and existential quantification are defined in Girard's fourth paper on transcendental syntax [Gir20a, Section 5.2].

§83.4 **Definition** (Behaviour for universal quantification)**.** Let $\mathfrak{E}$ be an epidictic architecture. The behaviour $\forall X.\mathbf{A}$ is defined by:

$$\forall X.\mathbf{A} := \bigcap_{\mathbf{T} \in \mathfrak{E}} \{X := \mathbf{T}\}\mathbf{A}$$

§83.5 **Definition** (Behaviour for existential quantification)**.** Let $\mathfrak{E}$ be an epidictic architecture. The behaviour $\exists X.\mathbf{A}$ is defined by:

$$\exists X.\mathbf{A} := \Big(\bigcup_{\mathbf{T} \in \mathfrak{E}} \{X := \mathbf{T}\}\mathbf{A}\Big)^{\perp\perp}$$

§83.6 Notice that we do not require a closure by bi-orthogonality. However, for the existential quantification which uses a union, we will need a closure as for the tensor of behaviours.

§83.7 Once we have quantifiers, it is possible to encode the part of logic we did not have in the previous chapters. This encoding has already been mentioned by Girard in the context of System F [Gir11a, Section 12.B.2]. These encodings are also extended (but unproven)

to neutral elements in Lafont's *"Linear logic pages"* [Laf99]. In Girard's recent paper, an encoding of full exponentials is also suggested from neutral elements [Gir17, Section 5.1].

§83.8 **Additive connectives.** The additive connectives which have been ignored until now can be encoded with second-order quantifiers. We start from the connective $\oplus$ corresponding to the intuitionistic $\vee$ in natural deduction. An informal way to think about how the encoding should look like, is to think about what *specification* $\oplus$ should satisfy or how its behaviour (use) may be described. If we look at the rule $\vee E$ (*cf.* Figure 9.3), it says that for any C, if A leads to C and B leads to C then we should obtain C. This can be described by the second-order formula $\forall C\ (A \multimap C) \Rightarrow (B \multimap C) \Rightarrow C$. We obtain the following encoding of additive connectives using behaviours $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{A} \oplus \mathbf{B} := \forall C.\ (\mathbf{A} \multimap C) \Rightarrow (\mathbf{A} \multimap C) \Rightarrow C \qquad\qquad \mathbf{A} \,\&\, \mathbf{B} := (\mathbf{A}^\perp \oplus \mathbf{B}^\perp)^\perp$$

§83.9 **Additive neutral elements.** The additive neutral elements can be encoded by the following behaviours:

$$\mathbf{0} := \forall X.X \qquad\qquad \top := \exists X.X.$$

If we consider that $\mathbf{0}$ represents contradiction and $\top$ absolute evidence, then $\exists X.X$ expressed the existence of a provable statement, *i.e.* of the possibility of an evidence. As for $\mathbf{0}$, in usual semantics, we usually do not want everything to be provable[4], hence it is interpreted by the absurd statement that every statement is provable.

§83.10 **Full exponentials.** In this thesis, we only defined limited exponentials corresponding to the intuitionistic arrow $\Rightarrow$ and its linear negation. If we would like to define full exponential modalities ! and ? in order to recover the full power of linear logic, then it is possible to use second-order quantifiers together with $\Rightarrow$ as follows:

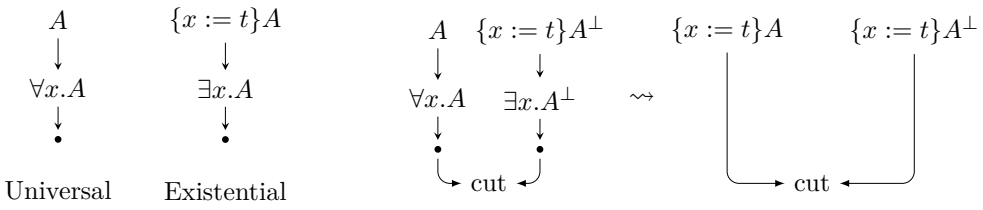
$$!\mathbf{A} := \forall X.\ (\mathbf{A} \Rightarrow X) \multimap X \qquad\qquad ?\mathbf{A} := (!\mathbf{A})^\perp.$$

The formula $\forall X.\ (\mathbf{A} \Rightarrow X) \multimap X$ expressed the fact that $\mathbf{A}$ can freely be subject to a non-linear behaviour (duplication and erasure) in any context of the epidictic architecture considered.

§83.11 **Multiplicative neutral elements.** There is a known encoding of multiplicative neutral elements based on exponentials and additive neutral elements:

$$\mathbf{1} := !\top \qquad\qquad \bot := ?\mathbf{0}.$$

[4]Note that I consider "usual logic" in this case since we are in the epidictics. In the transcendental syntax, when free from any logical system, "being provable" corresponds to the ability to materialise a statement by the constellations of some behaviour. This is usually not a problem.



(a) Links for quantifiers.

(b) Cut-elimination.

Figure 84.1: Proof-nets of predicate calculus. They are mentioned in Girard's blind spot [Gir11a, Section 11.C.3] and one of his article on quantifiers for linear logic [Gir91].

84 Usine in the case of predicate calculus: a sketch

§84.1 Unfortunately, no serious development of the Usine interpretation for second-order logic has been given yet. In order to illustrate the interpretation of quantifiers, Girard suggests to look at predicate calculus instead which is (interestingly) interpreted as a restriction of second-order logic [Gir18b, Section 5] (Recall that in the transcendental syntax, "first" and "second" order are different from usual first and second-order logic).

§84.2 In the third article of transcendental syntax [Gir18b], Girard states a conceptual problem with first-order individuals, which are objects of the universe on which some *properties* apply. For instance, natural numbers, persons, programs etc. The problem is that nothing justify their existence: they purely come from our intuitive perception of logic. They exist because they allow us to express what we have in mind (which can indeed be satisfying enough depending on what we do).

§84.3 As mentioned by Girard, when considering first-order proof-nets (*cf.* Figure 84.1), predicates such as P and individuals such as t in $\forall x.A$ and $P(t)$ play no role, computationally speaking. Actually, we could even remove predicates and individuals and only consider formulas. In other words, it is sufficient to consider everything as a property or statement. In transcendental syntax, since behaviours are considered, properties corresponding to individuals would express what kind of individual they are and this would be materialised by constellations corresponding to instance of that individual. An individual $f(a)$ would be a specification saying "I am $f(a)$" and instead of a property $P(f(a))$ saying "the individual $f(a)$ has the property P" we would have a statement saying "I'm an individual satisfying property P". To have becomes "to be". Predicates simply appear as an artificial way to manage allowed and forbidden interactions: social segregation again.

§84.4 **Quantification of predicate calculus.** We end up with a restriction of second-order logic in which quantification is restricted to a specific epidictic architecture encoding the set of all terms considered in a given universe. Hence, when considering tests for

the universal quantification $\forall x.A$, the shape of x is expected to be of a specific shape corresponding to an encoding of term. The same idea applies to existential witnesses.

§84.5 **Encoding of terms.** A constant is of type ⅂. But in order to encode a function symbol f, we must be able to encode pairs of symbols. The term $f(a)$ then corresponds to the encoding of (f, a). This encoding has to be *injective*, meaning that whenever $f(a) = f(b)$ then $a = b$ must follow. Girard suggested [Gir18b, Section 2.4] the following encoding inspired by set theory[5]:

$$< \mathbf{T}, \mathbf{U} >^{\bigstar} := (\mathbf{T} \parr \mathbf{U}) \otimes (\mathbf{T} \parr \mathbf{T} \parr \mathbf{U}).$$

§84.6 **Encoding of variables.** As for variables they should be handled with explicit substitutions [Gir18b, Section 5.2]. A variable α is treated with a ray $\alpha(X)$ (the X which is variable of stellar resolution should not be mistaken for a variable of predicate calculus). Variables occur either positively or negatively. Typically, when a variable α appears in the premise of an implication $A \multimap B$, then since $A \multimap B$ is implicitly $A^{\perp} \parr B$ hence A and the variable α appearing inside are negated (recall that terms are interpreted by formulas). We hence have addresses $\alpha(t)$ and $\alpha^{\perp}(t)$ for a variable α and its linear negation $\alpha^{\perp}$. In order to relate variables with a formula, Girard design addresses so that α and $\alpha^{\perp}$ are sort of left and right branches of a formula $\forall\alpha.A$ in which they appear. We then have three sort of addresses:

- $\alpha(x) := (\forall\alpha.A)(\mathtt{l} \cdot X)$ referring to positive occurrences of α in $\forall\alpha.A$;
- $\alpha^{\perp}(x) := (\forall\alpha.A)(\mathtt{r} \cdot X)$ referring to negative occurrences of α in $\forall\alpha.A$;
- $(\forall\alpha.A)(\mathtt{c} \cdot X)$ referring to the formula $\forall\alpha.A$ itself (as for the interpretation of MLL atoms).

These definitions are also extended to existential quantification. For instance, in the formula $\forall\alpha.P(\alpha) \multimap P(c)$, we expect that both $P(\alpha)$ and $P(c)$ are multiplicative formulas with their own sub-addresses. We will have axioms with rays $(\forall\alpha.P(\alpha) \multimap P(c))(t)$ where t an address referring to either an atom (as usual) or a variable of a term. In order to represent an explicit substitution $[\alpha := \sigma]$ where σ is a term, it is possible to design a star which can interact with the rays encoding a variable and replace them by rays encoding σ (such explicit substitutions can then be seen as constellations of adapters).

§84.7 **Equality.** An important predicate of predicate calculus is *equality* identifying individuals. Since everything boils down to formulas (there are no individuals), then equality between individuals (which are represented by multiplicative propositions) naturally becomes the linear equivalence $\equiv$. A famous criticism of Girard [Gir18b, Section 2] about the equality of predicate calculus, in its second-order form (*cf.* Paragraph 8.7), presupposes a specific space of properties, *i.e.* an implicit epidictic architecture. Indeed, when we say that $x = x$ when the two occurrences of x have the same properties, that is

[5] In set theory, the pair (x, y) can be represented by $\{\{x\}, \{x, y\}\}$.

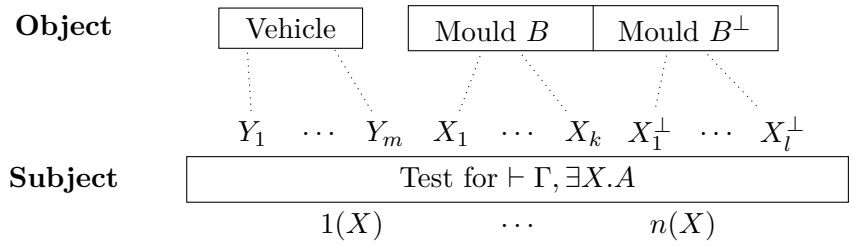


Figure 84.2: Vehicle and mould tested against a test for an existential formula. The conclusions of the test are $1, ..., n$. Variables Y_i can be either positive (a variable Z_i) or negative (a variable $Z_i^\perp$) and variables $X_i, X_i^\perp$ are occurrences of X and $X^\perp$.

$P(x) \Rightarrow P(x)$ for any predicate P, this looks absolutely trivial. However, imagine that we are in a computer in which the two copies of x used are located in two different places in the computer memory. It is common in programming to have variables at different addresses but with an equivalent content. A property of computers which would distinguish the two copies of x is to say that P is "being located at address α". This property is excluded because we consider that logic does not live in the computer world but in a logical world we made up. But it is actually reasonable to consider such properties as we always do in computer science. Hence, in the transcendental syntax, no epidictic architecture is assumed but if we need one (to consider the usual "logical properties") then we must make it explicit.

§84.8 **The mould.** A mould is a constellation-test specifying existential witnesses. If we look at the existential rule $\exists$ in Figure 82.1, we see that the existential witness σ is subject to a substitution $\{\alpha := \sigma\}$ replacing a variable α. Because variables appear either positively or negatively, so does the mould. Several choices of mould are possible. If B is the formula we associate with σ then the mould consists of tests for B and for $B^\perp$ that I call $\texttt{Tests}(B)$ and $\texttt{Tests}(B^\perp)$. Positive occurrences α will be linked to $\texttt{Tests}(B)$ and negative occurrences $\alpha^\perp$ to $\texttt{Tests}(B^\perp)$.

§84.9 As explained before, what is now tested against a specification/set of tests is not simply a vehicle (computational entity) but a mix of a vehicle together with a mould: the tested comes with its own certificate. We then need to check the vehicle and its mould passes the tests. It is like checking tickets for a concert; some tickets can be fake but still be considered valid. Given a mix of a vehicle with its mould, there is no guarantee that the two parts of the mould are negation of each other. We have two requirements:

1. the tested must present a clear separation between vehicle (fully objective) and mould (fully subjective), what Girard calls an *épure*;
2. two parts of the mould should be distinguished, each being the negation of the other.

A way to ensure that the two parts of the mould are negation of each other, a natural solution is to try to connect them with cuts and show a cut-elimination theorem. Although it should work in some cases with the right epidictic architecture, it is known that it is unprovable in general. The whole conception of the mould in the existential rule is illustrated in Figure 84.2.

85 Discussion: the theory of epidictic architectures

§85.1 In this chapter, I sketched how quantification could be developed in transcendental syntax in order to express second order logic and predicate calculus. However, we still have no true theory of epidictics. It is not even clear what such a theory should be. But it should at least provide a way to characterise what a generic proof-structure is. We are in a free space of construction where it is *possible* to build proof-structures what it can be useful to *limit* the space to those proof-structures so to make generic reasoning possible: since the world is limited, I can know what are the possible shape of the unknown.

§85.2 **A return to...the meta?.** If analytics and synthetics are respectively matter (where we construct programs) and meaning (where we construct formulas) then epidictics (formatting synthetics) should be the space we construct logical languages or logical systems. It is then a sort of *meta-language* except that unlike previous meta-languages for logic, its purpose is not to justify but to limit the potential for efficiency. Transcendental syntax works with only 3 layers (analytics, synthetics, epidictics) without the need for an infinite hierarchy of semantics. Apodictics is simply the absence of the third layer.

§85.3 In my opinion, a way to tackle this question is to ask ourselves how we would implement transcendental syntax on a computer. Imagine that we have a programming language in which we can define constellations. It is also possible to freely design and build tests in order to work with proof and type checking. It is possible to mix tests from different logics. However, we sometimes would like to work with a specific logic and enable generic reasoning but also in order to build tests automatically from a notion of formula, among other things. But we also would like to tell what is a proof-structure is, that a vehicle has to be made in a specific way: with binary stars of pairwise disjoint terms which are either polarised with + or unpolarised. We need a way to express such structural and interactional constraints over constellations in a convenient and generic way.

§85.4 **A theory of social control.** It can be even trickier if we add models of computation on the top. I believe that epidictics should be able to answer questions such as "what is an automata?" (interactionally and structurally speaking). Remark that structure is deeply related to interaction/computational behaviour. Interactions of an object are limited by its structure and structure puts control over interaction. For instance, we could put some structural layer over constellations so that execution of automata will necessarily start on initial states and do computation in the "right direction". We could forbid some interactions to happen for instance in circuits so that we first compute all

inputs before computing the output of gates. Such characterisations are what allows for generic reasoning. Although epidictics may look like an axiomatic theory of constraints, it is not exactly the case. The behaviour we force over the objects are only the behaviour we wish for because of some purpose in some system. It is a sort of *ethics*. We *want* our objects to behave in a specific way but they are free to interact in other ways when put against constellations outside of the system. Epidictics is a "guide" for computationals objects and not a system of constraints. Moreover, we can also change epidictics (or even remove it) without changing objects like how we can change a political regime.

§85.5 **Design plans.** Something I have in mind is that a potential epidictic language should probably be able to design "representative objects" from which constellations of a system are instantiated (it can be done with an homomorphism like how we extract diagrams from a dependency graph, in this case we would extract dependency graphs from an epidictic representative). Since only some specific constellations can be created, it induces a limitation of synthetics: only some behaviours can be considered, thus forming a logic. For instance, an epidictics which only allows typed polymorphic λ-terms to be created could be able to express natural deduction. It is like those design plans used to construct some objects or buildings. We need *concepts* and ways to produce those concepts. For instance, in the case of MLL proof-structures, the primitive concepts are axioms and formulas.

§85.6 **About type judgements.** As mentioned in the last part of the third article of transcendental syntax [Gir18b, Section 6.2], epidictics implicitly appears in Martin-Löf type theory (MLTT). In type theory, we usually have statements about types themselves. We have rules saying that $0 : \mathbf{N}$ (the term 0 is of type $\mathbf{N}$, the type of natural numbers) and if $n : \mathbf{N}$ then $s(n) : \mathbf{N}$. Now, we also have a rule concluding with $\mathbf{N} : Type$, *i.e.* that $\mathbf{N}$ is a valid type. For conjunction, we say that if $A : Type$ and $B : Type$ then $A \wedge B : Type$. This looks like a structuration of epidictics. One possible task for transcendental syntax is to give a clearer status for those type assertions in the light of the Usine/Usage interpretation and the computational justification of logical rules.

§85.7 Girard finally concludes with the following sentence describing the current status of epidictics: *"At the present moment, epidictics is but a name on a blank area of the logical charts; hence a sort of new frontier for logic."* [Gir18b, Section 6.2].

Conclusion

86 Summary and contributions

§86.1 I would say that the main purpose of this thesis was to fill the gap between logic as it is today and the latest developments in linear logic. The gap was big. The Curry-Howard correspondence established a formal correspondence between logic and computation. However, the developments of computer science, although recent, went very fast. Maybe even too fast for logic. Logic has a lot of catch up and transcendental syntax is one of the first bricks for something bigger.

§86.2 **Contextualisation.** The first notable thing this thesis provides is contextualisation. Transcendental syntax does not come out of nowhere. It is part of a long story of people and ideas. In this thesis, it has been put in context and is seen as the natural successor of Girard's geometry of interaction (itself being a successor of the theory of proof-nets). It is also connected to known fields such as (classical) realisability (with ideas already explored by Colin Riba and Emmanuel Beffara) or the broad field of program testing. I had to reconstruct the story behind transcendental syntax so that it becomes more natural and accessible. As for the philosophical side, transcendental syntax can be seen as an answer to old questions regarding the justification of logical rules (something already mentioned by Paolo Pistone) that I explained in Paragraph 12.4. Actually, the geometry of interaction alone could serve as a satisfying answer but the transcendental syntax goes further by introducing finiteness of the conditions of possibility of reasoning but also a focus on the primitive shape of objects (which extends the Church-style typing of λ-calculus).

§86.3 **Illustration.** There is a big chapter (*cf.* Chapter 8) dedicated to the illustration of stellar resolution, the model of computation behind transcendental syntax. Automata, logic programs, tile systems and models of circuits are illustrated with simulation proofs and examples. I believe these illustrations to be more than mere encodings since they are very faithful to the original computational mechanisms of these models. Stellar resolution computes by resolving structured relations of constraints creating a sort of flow of information. This can be seen as a very elementary and primitive mechanism which underlies most (if not all) models of classical computation (*cf.* end of Chapter 8).

§86.4 **Formalisation.** The main contribution of this thesis is that it is the first formalisation of transcendental syntax:

- stellar resolution is formalised (*cf.* Chapter 7) and properties (*cf.* Chapter 9) about it are proven (in particular properties of confluence which are essential);
- several ways to execute constellations are defined (*cf.* Section 49, Section 50 and Section 51) and in particular, interactive execution can lead to natural implementations of stellar resolution;
- a model of MLL is given in Chapter 10, inspired by the treatment of multiplicatives in the geometry of interaction (*cf.* Chapter 5);
- simulation of cut-elimination and of the Danos-Regnier correctness criterion are proven (*cf.* Section 68);
- classical results such as completeness and soundness of several models of MLL are stated (*cf.* Section 70).

§86.5 **Extensions.** Few ideas of transcendental syntax which have been sketched by Girard are interpreted and extended. Although it is still informal, it is meant to be a possible inspiration for future works extending the initial formalisation of transcendental syntax given in this thesis. In particular, I give my own interpretation of Girard's apodictic (*cf.* Chapter 12) and epidictic (*cf.* Chapter 13) which are very vague notions. On a more technical side, I give a basis for a formal definition of exponentials (*cf.* Chapter 11) in transcendental syntax, which allows the possibility of defining "alternative exponentials" such as Girard's expansionals (*cf.* Section 78).

87 Horizons

§87.1 When reading Girard's original articles on transcendental syntax, it may be difficult to understand what can be created out of it. The project may look rather shallow for some people. In this section, I expose several ideas of possible future works that I imagined without having the opportunity to develop them.

§87.2 **Remaining unsolved technical problems.** There are few completely technical problems left in this thesis, which could be solved but were not because of a lack of time or the presence of other priorities:

- there is no proper treatment of subjective rays. As shown in this thesis, internal polarities adds a complex behaviour to constellations. In particular, some unpolarised ray can become polarised. A proper treatment should include use cases for subjective rays together with an extended formal definition of execution and examples;
- as stated in Section 51, interactive execution with an interactive configuration $\Phi \vdash \Phi$ of reference constellation equal to its initial interaction space produce duplicates but the exact number of duplicates has not been characterised. The answer to this question (which is apparently a non-trivial problem on graphs) may allow us

to filter out duplicates and identify interactive and concrete execution (the latter being able to exactly simulate abstract execution);

- links with logic programming have not been properly defined although briefly discussed in Section 53. Execution is related to the resolution operator and stellar resolution is related to the semantics of logic programs (which attracted a lot of hype in the 90s). No formal links have been established. Interestingly, there are works of Wolfgang Bibel[6] such as the "connection method" [Bib13] which has been related to proof-structures with Bertram Fronhöfer's paper in honour of Wolfgang Bibel [Fro00]. There may also be things to say about constraint programming (stellar resolution is a system of constraints) and Andrew's refutation by matings [And76];
- there is not yet any alternative definition for λ-calculus with stellar resolution. When I talked about transcendental syntax with Lionel Vaux, he told me that it would be interesting to have a definition of untyped λ-calculus without going through an encoding using proof-structures. I tried to encode the Krivine Abstract Machine in Section 57 but I did not prove that the encoding is faithful and correctly simulate it. When I tried, I was under the impression that it was possible to define λ-calculus directly by encoding its term graph and using the mechanisms of stellar resolution (variable and constants) to simulate explicit substitutions;
- there is no computationally faithful definition of circuits in stellar resolution (*cf.* Section 58). Circuits (for instance boolean circuits) compute from inputs to outputs. In particular, the result of a gate can only be computed when its two inputs are defined. However, nothing specify this synchronised flow of computation in constellations. I thought of something like associating elements of an ordered set to rays but it was not a very natural thing to do (moreover, regular constellations become special case of those extended constellations with ordered rays);
- there is no complexity analysis for constellations. Some answers may be found in works on the complexity of logic programs [DEGV01]. The complexity of term unification is already well-documented but the execution of constellations raise new questions of complexity. In particular, the shape of terms in constellations has a direct influence on complexity. There may be formal links between class of shapes and complexity classes;
- there is no termination analysis for constellations. I tried to use works in programming logic [NGSKDS07] to state results but I omitted it since it was not complete nor useful for the results of this thesis. The idea was to use an analysis of dependency graphs to assert whether a constellation is terminating or not. The argument would rely on usual termination proofs of term rewriting theory [BN98, Section 2.3] by looking for size-decreasing cycles, a technique also used in functional programming [LJBA01];

[6] Who is explicitly mentioned in several recent articles on deep inference [GG07, Brü04, Gug].

- in Section 63, I defined some ideas of "stellar compression" in order to minimise constellations. However, since constellations can be seen as generalised automata, it may be possible to define more advanced simplification techniques and have a notion of "minimal constellation" *w.r.t.* some behaviour;
- there is a lack of proofs regarding exponentials in stellar resolution and a lack of evidences regarding the ability of stellar resolution to define interesting exotic exponentials (Girard's expansional are only sketched from Girard's original presentation);
- although I tried to explain Girard's notion of truth in Section 79, there is currently no serious purpose for it and there is no technical use case for an "axiom-free" Peano arithmetic. It is mostly interesting for philosophical purposes but may find some applications if one explores the subject;
- the interpretation of predicate calculus has only been roughly sketched in Section 84 and there is still a lot of work to do in order to achieve a full formalisation of it.

§87.3 **Extensions of stellar resolution.** Since the beginning of my thesis, Thomas Seiller and I thought about extending stellar resolution with coefficients (something already present in Seiller's work on interaction graphs and graphings). Personally, I had chemical reactions and Markov chains in mind and even tried to generalise stellar resolution using chemical models (without succeeding). Constellations can be seen as atoms or molecules interacting with each other to produce a reaction. After discussing with Seiller, we remarked that there was two possible distinct extensions:

- putting coefficients on stars. Fusion between stars then applies a monoid multiplication on coefficients. We would then have stars and constellations weighted in some given monoid. It corresponds to a generalisation of Seiller's weights on interaction graphs [Sei12a]. Girard also had similar ideas in his second article on transcendental syntax [Gir16b];
- another idea was to put coefficients on the edges of dependency graphs so that the sum of coefficients related to a ray is equal to 1. The point is that for each ray, there is some probability to connect to another specific ray. This is reminiscent of chemical reactions. In that case, coefficients are associated with pairs of rays.

The two extensions are actually compatible and generalise the stellar resolution defined in this thesis.

§87.4 **A new version of proof-nets.** An idea of Seiller was to use ideas of transcendental syntax to develop a sort of correction of proof-nets which would overcome the known problems of the theory of proof-nets (regarding local and global mechanisms appearing with exponentials and additives). It is true that it is not completely clear what a "good" proof-net should be. I personally do not support this direction since I believe that constellations are the next form of proof-nets and that they *are* what proof-nets

should be (one may say that I am avoiding the problem). Hence, stellar resolution should define logic by themselves without trying to look for connexions or updates of the original theory of proof-nets (which may be forgotten).

§87.5 **Parallel and concurrent computation.** Stellar resolution is an asynchronous and parallel model of computation. Parallel computation for term unification is a subject which has been explored quite a lot [HM89, Sib05, Kit91, VS84], probably because of the hype around logic programming in the 90s. An interesting direction is to connect stellar resolution to known works in parallel and concurrent computation. During my thesis, I had two ideas in mind which were left without conclusion:

- with Julien Marquet, we tried to think about an encoding of Lafont's interaction nets and in particular interaction combinators;
- with Félix Castro, we tried to think about an encoding of π-calculus. I wanted stellar resolution to replace process calculi by seeing how it could simulate some of them. However, it was less trivial than expected because of the name handling of π-calculus.

Both these ideas were not worth the time and efforts and I had other priorities so nothing has been realised so far. There are also subjective rays that I believe to be connected to synchronised computation and mechanisms of concurrent programming but I was never able to reach a conclusion (mostly because this consideration came very late in my thesis).

§87.6 **Generalised token machine.** One well-known application of the geometry of interaction is the token machine or interaction abstract machine (*cf.* Section 36). Since transcendental syntax extends and generalise the geometry of interaction, it is a natural direction to look for a generalised token machine. There are already such generalised machines in the literature [CC23, DLTY17, CVV21] but no link has been established so far. The idea of a generalised token machine is something which appeared very early in my thesis but which has finally been replaced by concrete and interactive execution (which came way later). The idea of what I called the "stellar machine" is that dependency graphs could be seen as an automaton. We put "tokens" (associated with a family, in order to represent the different diagrams being constructed) which travel through the dependency graph. At each step, a unification problem is solved by adding equations related to the traversed edges. A lot of non-trivial mechanisms have to be considered: non-determinism, colliding tokens/families etc. It seems to me that the treatment of those mechanisms may be similar to what is done in Chardonnet's works [CVV21].

§87.7 **Verified computation.** Transcendental syntax connects logic with the program testing of computer science. Correctness tests are seen as analysing the shape of constellations in order to assert that they have some computational behaviour. Although it makes sense, I never gave examples of verified computation. One could think of properties on automata or circuits, for instance. Automata would be encoded as constellations but also tests enduring that an automaton is part of some class of automata. Automata themselves

can be seen as finite tests asserting the membership of words in some language. There might also be connexion with model checking since constellations can be seen as sort of labelled transition systems.

§87.8 **Discrete complex systems and dynamical systems.** Seiller and I independently remarked that stellar resolution could be seen as a sort of tile system. I discovered Wang tiles at a summer school for young researchers in computer science (EJCIM) and Seiller had already heard about abstract tile assembly models before. At some point, while learning about tile systems, I had the hope of generalising stellar resolution with discrete dynamical systems (I tried to discuss with Benjamin Hellouin but without success) or models of complex system theory. It was very vague but I wanted to know if notions such as chaos and attractors had some sort of logical meaning. I did not have the mathematical background to seriously consider this direction.

§87.9 **Ludics.** This is something which I almost completely omitted in this thesis. Ludics can be seen as a abstraction of sequent calculus whereas geometry of interaction is an abstraction of proof-structures. In an informal with Seiller, we discussed about encoding ludics with stellar resolution so that transcendental syntax would subsumes both ludics and geometry of interaction. An idea was to put an order on constellations to internalise the sequentialisation of logical rules. This may be related to my idea of order and synchronisation for subjective rays. Moreover, I also discussed with Carlos Olarte and he told me that the control of constellations in epidictic (which may also be related to synchronisation/order in computation) reminded him of focalisation which appears in ludics but also in proof-search for linear logic.

§87.10 **Deep inference.** Deep inference [Gug, Brü04, GG07] is a very recent theory of logic which provides an abstraction of sequent calculus which is distinct from both ludics and geometry of interaction. It is an alternative new culture of logic. There is currently no research about how deep inference is different or close to transcendental syntax. After discussing with Pablo Donato and Adrien Ragot about the link between deep inference and transcendental syntax, I was under the (vague) impression that sequent calculi could be seen as recipes for the construction/generation of constellations/programs.

§87.11 **Descriptive complexity.** Connexions between transcendental syntax and computational complexity is really something I would like to see as I believe complexity problems are related to logic (once we have a more mature notion of logic – in my opinion logic is related to the shape of things and their computational potential). There are already works connecting ideas of geometry of interaction with implicit computational complexity (ICC) [Sei20a, BP99, AS16b] and transcendental syntax may provide results in this direction. However, I also hope for a new reading of descriptive complexity since types (either from the Usine or Usage point of view) have a *descriptive function*. They assert some computational potentiality. Can we characterise computational complexity classes by tests/behaviours? Constellations which can be typed in predicate calculus and second-order logic (and thus have a specific shape) would theoretically correspond to programs of complexity class P and NP [Imm12, Imm86].

§87.12 **The nature of programs and algorithms.** There is an article of Gurevich which asks what is an algorithm [Gur12]. Although seemingly innocent, this is a rather profound question. There is currently no satisfying formal definition of algorithm or even of a program. Thomas Seiller already discussed this matter several times in few talks. The subject also led to the ANR project "La géométrie des algorithmes" (GoA) coordinated by Alberto Naibo. Seiller tries to answer the question with his graphings but I believe the question could also be tackled only with stellar resolution which captures classical computation pretty well. As discussed in Section 19, algorithms may be seen as specifications. In our cases, it corresponds a Usine interpretation (since we would like finite checking that the programs is an implementation of some algorithm). Now, what is a *sequential* algorithm? I do not have an answer but I believe that we should have some sort of "sequential formulas" satisfied by sequential programs. Instead of extending the notion of test or program/constellation with ad-hoc external constructions, I rather believe in extending the "shapes" of constellations (plurality of individuals), so that it generalises the stellar resolution of this thesis.

§87.13 **Open proof assistant.** An idea which I find exciting but also vague for the moment. Proof assistants such as Coq relies on one specific system, for instance the calculus of inductive constructions which corresponds to some λ-calculus. No matter how convenient or inconvenient it is, we are limited to one logic and one model of computation to encode properties and models of computation (automata, circuits, programs etc). Using transcendental syntax, the core is minimal and boils down to the unification algorithm which is well-known and simple. Everything else is constructed within the language itself. In particular, typing, which is essential in proof assistants, is given by tests. A logic generates tests with formula labels (representing properties to be shown). Hence, several logic can coexist. Logical systems become sort of libraries/modules as in programming. Proving is not bound to a specific logic such as intuitionistic logic. Certainty relies on tests known/proven to be adequate (this cannot be directly proven in the system itself). Whether such an "open" proof assistant is good or not is still an open question. One may argue that functional systems are sufficient/convenient for most purposes. Anyway, potential proof assistants based on stellar resolution should not be understood as a potential replacement of other proof assistants but as another type of proof assistant.

§87.14 **Epidictic language as a script language for macros.** The epidictic is what allows stellar resolution to constructs closed systems. Such closed systems makes reasoning more efficient and "axiomatic" but also makes possible quantification (since the world is closed, I can make assumption of the shape of the possible objects on which I quantify). Typically, imagine that you would like to work with λ-calculus in stellar resolution. It would be too tedious and prone to errors to write the corresponding constellations and tests for each formulas each time. It actually took me a long time to figure out what could be a concrete realisation of Girard's epidictic. It is only late in my thesis that I considered that it could be... a macro system. The primitive system is stellar resolution which is very flexible and everything else is just syntactic sugar with complex macros. For instance, we would have a way to construct formulas as syntactic labels but also

macros to inductively associate formulas with test-constellations.

§87.15 **A realist thesis?.** I am not a philosopher at all and have absolutely no background in philosophy but during my thesis I had the hope that transcendental syntax would raise insights regarding scientific/semantic realism and essentialism in logic. If an "existentialist" approach to logic (as Girard calls it) consists in forgetting all primitive definitions so that types/formulas would be reconstructed from computational individuals, then it looks like transcendental syntax is a bit more than just an existentialist approach to logic. Transcendental syntax adds a Usine interpretation which was not present in ludics nor geometry of interaction. This interpretation shows that "the primitive shapes of things matters" and in particular that it is possible to anticipate the behaviour of individuals from their shape. I might be mistaking but it seems to me that the shape (of computational objects) becomes a *primitive essence* in this case. However, we know that behaviours cannot always be fully characterised by finite tests, hence this anticipation is indeed limited. There is also the problem of whether mathematical (or logical) entities exist in an external reality, independently of us. It seems to me that computational monist testing is a process of "reality-discovery" producing responses which are then interpreted/formatted by us. Although the medium of interaction (whether it is stellar resolution or something else) is relative/chosen, syntactic interactions occurring in mathematics says things about how reality is structured. This may explain the relevance of mathematics over the "real world" (and in particular the fact that mathematics says things about physics, thus allowing us to construct reliable bridges or reliable programs).

88 Limits of the current presentation of transcendental syntax

§88.1 **A too open theory.** Transcendental syntax is like a computational sandbox for logic. It is an alternative point of view on logic (like deep inference). Because it is so free, it also lacks direction. As shown in the previous section, there is a lot of potential applications but it remains to filter out the most interesting and relevant ones. Transcendental syntax is still a very speculative and experimental subject for the moment.

§88.2 **Complexity issues.** Concrete execution which is the implementable version of abstract execution (the more "semantical" or "denotational" way to evaluate constellations) has a horrible complexity. We constantly have to compute graph isomorphism which is known to be a difficult problem. We have no choice but to exclude concrete execution from real-world applications. Only interactive execution (which could be refined into a token machine) is left. Although no complexity results have been stated, it seems that the complexity of constellations representing some computational objects (automata, circuits, ...) is faithful to the original complexity of the model. All additional complexity added with constellations correspond to term unifications. Another problem of complexity related to logic is that the most natural correctness criterion is the Danos-Regnier correctness which is known to have an exponential complexity (there are 2^n tests for n par hyperedges). It is currently not known if other more efficient correctness criterion can be

easily and efficiently implemented in stellar resolution. It may be the case that adding an "epidictic layer" makes computation more efficient (we only look at external opaque interactions without considering the underlying micro-computational mechanisms – this is the point of axiomatic theories).

§88.3 **Limitation to classical computation.** The presentation of transcendental syntax in this thesis is obviously limited to classical computation. However, this is not an absolute limitation of transcendental syntax itself. Constellations could be extended with coefficients in order to interpret probabilistic or quantum computation (Chardonnet uses a token machine with complex coefficients). However, it is unsure that doing so would provide an interesting non-classical model of computation (when compared to models which purpose is to be relevant for non-classical computation).

§88.4 **Another blind spot.** As explained by Girard in the end of his third article on transcendental syntax [Gir18b], transcendental syntax still cannot give a "transcendental" status to type judgement appearing in (Martin-Löf) type theory. Moreover, there are also new works such as the so-called Homotopic Type Theory (HTT) with a new treatment of equality. There is also deep inference. Although all these theories speak about logic, they looks distinct from transcendental syntax. For the time being, transcendental syntax is not able to provide comments, analysis nor comparisons. Transcendental syntax, at least in its current form, does not look as universal as some would like it to be.

§88.5 **A materialist conception of logic.** After discussing with Baptiste Chanus and other people about transcendental syntax, Baptiste stated that despite the appearances of transcendental syntax and the hope of some people to see something more fundamental or universal than other conceptions of logic, transcendental syntax was, after all, a computational point of view of logic, hence a logical paradigm among other ones. It is true that transcendental syntax is deeply dependent of some material entities of a computational nature and that it does not take into account the social nature of the logical activity. In transcendental syntax, there is a strong belief that computation explains the whole logical activity. Whether it is right or wrong, this is out of my field of expertise. But if you still want to hear about my current belief, the natural and social aspect of logic may not be incompatible. There is the natural layer of stellar resolution, on which logical constructions can be made then the social/cultural layer correspond to the epidictic which corresponds to systems put over stellar resolution.

Appendix A

Mathematical conventions

A.1 General notations

◇ **Definition (:=) and equality (=)** I write $A := B$ when A is defined as B. It should not be mistaken with the *equality* $A = B$ saying that A and B are already defined and equal. It is similar to the difference between = (affectation) and == (boolean equality) in some programming languages such as the C language.

◇ **Confer/conferatur (*cf.*)** Used as reference to another thing (latin). For instance "*cf.* `A`" means that we can refer to `A` in case we want more information.

◇ **Domain of a function** Let $f : A \to B$ a function over sets. We define $\texttt{dom}(f) := A$, called the *domain* of f.

◇ **Exempli gratia (*e.g.*)** For example (latin).

◇ **Id est (*i.e.*)** That is (latin).

◇ **Images of a function** Let $f : A \to B$ a function over sets. We define $\texttt{img}(f) := B$, called the set of *images* of f.

◇ **Projections** Let $p = (a_1, ..., a_n)$ be a tuple. We define $\pi_i(p) = a_i$ as the i-nth projection of p.

◇ **Respectively (*resp.*)**

◇ **Sequence** By sequence, I mean a tuple $(a_1, ..., a_n)$ but more generally, it is defined as an indexed family indexed by natural numbvers (see Appendix A.2 below).

◇ **With respects to (*w.r.t.*)**

A.2 Set theory

Elementary set theory

§A.2.1 **Definition** (Cartesian product)**.** The *cartesian product* of two sets A and B is defined by $A \times B := \{(a, b) \mid a \in A, b \in B\}$.

§A.2.2 **Definition** (Disjoint sets)**.** Two sets A and B are *disjoint* if and only if $A \cap B = \emptyset$.

§A.2.3 **Definition** (Disjoint union)**.** Let A and B be two sets. Their *disjoint union* is defined by the set $A \uplus B := \{(a, 0) \mid a \in A\} \cup \{(b, 1) \mid b \in B\}$.

§A.2.4 **Example.** Let $A = \{a, b, c\}$ and $B = \{c, d\}$. We have

$$A \uplus B = \{(a, 0), (b, 0), (c, 0), (c, 1), (d, 1)\}.$$

Notice that the element $c \in A$ is distinguished from the element $c \in B$ by their associated index $i \in \{0, 1\}$.

When two sets are disjoint, we can omit the indexes 0 and 1 in their disjoint union since there is no ambiguity.

§A.2.5 **Definition** (Powerset)**.** The *powerset* of a set S is defined by $\mathcal{P}(S) := \{S' \mid S' \subseteq S\}$.

§A.2.6 **Definition** (Function application on sets)**.** Let $f : S \to S'$ be a function over two sets S and S'. If $S = \{a_1, ..., a_n\}$ then the application of f on S is defined by $f(S) = \{f(a_1), ..., f(a_n)\}$.

§A.2.7 **Definition** (Predicate associated to a set)**.** Let $P(x_1, ..., x_n)$ be a predicate with variables $x_1, ..., x_n$.

§A.2.8 **Definition** (Generalised union)**.** We define a *generalised iterative union* by

$$\bigcup_{i=n}^{n} := \emptyset \qquad \bigcup_{i=k}^{n} E_i := E_k \cup \bigcup_{i=k+1}^{n} E_i \quad \text{when } (k < n)$$

for sets $E_k, ..., E_n$, a counter i starting with $i = k$ and stopping at $i = n$.

We also define the alternative notations: $\bigcup_{k \leq i \leq n} E_i := \bigcup_{i=k}^{n} E_i$.

Let $P(x, S)$ be a predicate over a variable x and a set S ($x \in S$ and $S' \subseteq S$ are such predicates), and E an expression denoting a set where the variable x occurs. We define the following expression of *generalised union over a predicate* $P(x, S)$ on E:

$$\bigcup_{P(x,\emptyset)} E := \emptyset \qquad \bigcup_{P(x,\{e\}\cup S)} E := \{x := e\}E \cup \bigcup_{P(x,S)} E$$

We define the following notation for repeated generalised unions:

$$\bigcup_{B_1,...,B_n}^{A} E := \bigcup_{B_1}^{A} .... \bigcup_{B_n}^{A} E$$

Multisets

A multiset is a set in which a same element can appear several times. It is defined by a set and a map relating each elements of the set to its number of occurrence (called *multiplicity*).

§A.2.9 **Definition** (Multiset)**.** A *multiset* is a pair $(S, \texttt{nbOcc})$ of a set S and a total function $\texttt{nbOcc} : S \to \mathbf{N}$.

We use the notation $[a_1, ..., a_n]$ for finite multisets in which repetitions explictly appear.

§A.2.10 **Example.** Let $S := \{a, b, c\}$. We can define the multiset $(S, \texttt{nbOcc})$ such that $\texttt{nbOcc}(a) = 1$, $\texttt{nbOcc}(b) = 2$ and $\texttt{nbOcc}(c) = 3$. This multiset can be written $[a, b, b, c, c, c]$.

§A.2.11 **Definition** (Multiset induced by a set)**.** Let S be a set. It induces a multiset $\texttt{multiset}(S) := (S, \texttt{nbOcc})$ with $\texttt{nbOcc}(e) = 1$ for all $e \in S$.

§A.2.12 **Definition** (Multiset union)**.** Let $(A, \texttt{nbOcc}_A)$ and $(B, \texttt{nbOcc}_B)$ be two multisets. Their union $A \cup B$ is defined by the multiset $(A \cup B, \texttt{nbOcc}_{A\cup B})$ such that

$$\texttt{nbOcc}_{A\cup B}(e) = \texttt{nbOcc}_A(e) + \texttt{nbOcc}_B(e).$$

§A.2.13 **Example.** If we have two multisets $A = [a, b, b]$ and $B = [b, b]$, then their union is $(\{a, b\}, \texttt{nbOcc}_{A\cup B})$ with $\texttt{nbOcc}_{A\cup B}(a) = 1$ and $\texttt{nbOcc}_{A\cup B}(b) = 4$. The union can be written $[a, b, b, b, b]$.

Indexed families

Indexed families is an alternative definition for ordered multisets which I find more convenient. Instead of associating elements to a multiplicity, we start from a set of indexes and associate an element to each index.

§A.2.14 **Definition** (Indexed family)**.** An *indexed family* over I is given by a tuple $(S, I, \texttt{get})$ of a set S, a *set of indexes* I and an *index function* $\texttt{get} : I \to S$.

Indexed families with natural numbers as set of indexes correspond to sequences in mathematics and to arrays in programming.

§A.2.15 **Example.** We define an indexed family $(S, I, \texttt{get})$ with $S = \{a, b\}$, $N = \{0, 1, 2, 3\}$ and $\texttt{get}(0) = a$, $\texttt{get}(1) = a$, $\texttt{get}(2) = b$, $\texttt{get}(3) = a$. It corresponds to the array $\begin{bmatrix} 0 & 1 & 2 & 3 \\ a & a & b & a \end{bmatrix}$ where positions are given on the top with the corresponding integer index and the associated element on the bottom.

§A.2.16 **Notation.** Let $\varphi = (S, I, \texttt{get})$ be an indexed family. To make the notation closer to arrays in programming, we define the notation $\varphi[i] := \texttt{get}(i)$ for $i \in I$.

§A.2.17 **Definition** (Disjoint union of indexed families)**.** Let $\varphi_1 = (S_1, I_1, \texttt{get}_1)$ and $\varphi_2 = (S_2, I_2, \texttt{get}_2)$ be two indexed families. Their disjoint union $\varphi_1 \uplus \varphi_2$ is the indexed family $(S_1 \uplus S_2, I_{\varphi_1} \uplus I_{\varphi_2}, \texttt{get}_1 \uplus \texttt{get}_2)$ with $\texttt{get}_1 \uplus \texttt{get}_2 : I_{\varphi_1} \uplus I_{\varphi_2} \to S_1 \uplus S_2$ defined by $(\texttt{get}_1 \uplus \texttt{get}_2)(i) = \texttt{get}_k(i)$ when $i \in I_{\varphi_k}$ and $k \in \{1, 2\}$.

A.3 Language theory

§A.3.1 **Definition** (Alphabet and word)**.** An *alphabet* is a set of elements called *symbols*. A *word* over an alphabet Σ is an ordered sequence of symbols of Σ.

There is a distinguished symbol ε called *empty symbol*.

We often omit parentheses when writing words, *i.e.* we write abc instead of (a, b, c).

§A.3.2 **Example.** If $\Sigma = \{a, b, c\}$ then a, $aaaaa$, $acccb$, $aaabab$ are words over Σ.

§A.3.3 **Definition.** The *concatenation* of words is defined as follows:

- $\varepsilon \cdot w := w$ and $w \cdot \varepsilon := w$;
- $(a_1, ..., a_n) \cdot (b_1, ..., b_m) := (a_1, ..., a_n, b_1, ..., b_m)$.

§A.3.4 **Definition** (Kleene closure)**.** Let Σ be an alphabet, its *Kleene closure* written Σ^* is the set satisfying the following requirements:

- $\varepsilon \in \Sigma^*$
- if $c \in \Sigma$, then $c \in \Sigma^*$;
- if $w \in \Sigma^*$ and $w' \in \Sigma^*$ then $w \cdot w' \in \Sigma^*$;

§A.3.5 **Example.** If $\Sigma = \{a, b, c\}$ then $\{\varepsilon, a, b, c, aa, bb, cc, aaa, bbb, ccc, aba, abc, bac\} \subset \Sigma^*$.

§A.3.6 **Definition** (Notation)**.** Let $a_1, ..., a_n$ be symbols of some alphabet Σ. The set of *notations* over the *notation generators* $a_1, ..., a_n$ is the set N satisfying the following requirements:

- $\{a_1, ..., a_n\} \subset N$;

- if $a \in N$ and $x \in N$ then $a' \in N$, $a_x \in N$ and $a^x \in N$.

§A.3.7 **Example.** The set of notations over $\{a\}$ contains $a, a', a'', a''', a_1, a_2, a_3, a^1, a^2$ and a^3.

§A.3.8 **Definition** (Backus-Naur Form grammars)**.** The expression

$$A_1, ..., A_n ::= \qquad B_1 \mid ... \mid B_m \qquad \text{set } C \qquad \text{(D)}$$

corresponds to a notation for Backus-Naur Form (BNF) grammars. It says that the category or type called D, denoted by the set C, has elements represented by symbols $A_1, ..., A_n$ and constructed from one of the expressions in $B_1, ..., B_n$ (the vertical bar $\mid$ is then a disjunction of cases). The denotation D can be omitted.

We consider that all symbols of the set of notations over $\{A_1, ..., A_n\}$ can be used to represent elements of the set C.

It happens that some A_i occurs in some B_i and in this case, we have an *inductive type*.

§A.3.9 **Example.** Some typical examples are booleans, natural numbers, lists (parametrised by another type for elements) and trees (parametrised by another type for the content of nodes):

$a, b, c ::=$	$0 \mid 1$	set `Bool`	(Booleans)
$n, m, k, l ::=$	$0 \mid s(n)$	set `Nat`	(Natural numbers)
$l ::=$	$[] \mid e :: l$ with $e \in A$	set `List[A]`	(Lists)
$t, t_1, t_2 ::=$	$\emptyset \mid (t_1, e, t_2)$ with $e \in A$	set `Tree[A]`	(Trees)

Appendix B

Term unification

§B.0.1 Terms are used to represent statements. In this section, we are interested in solving equations $t \stackrel{?}{=} u$ between terms by searching for a way to replace variables by other terms in order to make t and u equal. Equations between terms are usually attributed to Robinson [R+65] but Herbrand [Her30] already studied term unification before in the context of his investigations on mathematical equations and proof theory.

§B.0.2 We refer the reader to the article of Lassez et al. [LMM88] for more details which are often omitted in the literature or Baader et al. [BN98] for a broader view.

B.1 Elementary definitions

§B.1.1 In this appendix I use a generalisation of term unification which is in not standard. In my definition of unification, I consider a binary relation $\supset$ over function symbols which tells us when two symbols are compatible. This will allows us to be more general when considering equations between terms.

§B.1.2 **Definition** (Signature)**.** A *signature* $\mathbb{S} = (V, F, \mathtt{ar}, \supset)$ consists of a countable set V of variables, a countable set F of function symbols whose arities are given by $\mathtt{ar} : F \to \mathbf{N}$ and a binary relation $\supset$: $F \times F$.

§B.1.3 **Convention.** In usual term unification theory and this appendix, we consider that $f \supset g$ if and only if $f = g$.

§B.1.4 **Definition** (First-order terms)**.** The set of (first-order) *terms* $\mathtt{Term}(\mathbb{S})$ over a signature $\mathbb{S} = (V, F, \mathtt{ar}, \supset)$ is inductively defined by the following grammar:

$$
\begin{aligned}
a, b, c, d, t, u, v, w ::= & \\
& \mid \quad X \\
& \mid \quad f(t_1, ..., t_n) \\
& \quad \text{for } X \in V, f \in F, \mathtt{ar}(f) = n \qquad \text{set } \mathtt{Term}(\mathbb{S})
\end{aligned}
\tag{Terms}
$$

§B.1.5 **Definition** (Set of variables)**.** The set of variables of a term is defined inductively as follows:

$$\texttt{vars}(X) = \{X\} \text{ (for} X \in V) \qquad \texttt{vars}(f(t_1, ..., t_n)) = \bigcup_{1 \leq i \leq n} \texttt{vars}(t_i)$$

§B.1.6 **Example.** Let $\mathbb{S} := (V, F, \texttt{ar}, \supset)$ be a signature such that $V = \{X, Y\}$, $F = \{c, f, g\}$ and $\texttt{ar}(c) = 0$, $\texttt{ar}(f) = 1$, $\texttt{ar}(g) = 2$. We can construct the following terms (among others): $c, X, Y, Z, f(c), f(X), g(c, c), g(X, Y), f(g(c, c)), g(f(X), f(f(Y)))$.

§B.1.7 **Definition** (Substitution)**.** A *substitution* is a function $\theta : V \to \texttt{Term}(\mathbb{S})$. Substitutions are extended from variables to terms by

$$\theta(f(u_1, ..., u_k)) = f(\theta(u_1), ..., \theta(u_k)).$$

The application $\theta(t)$ of a substitution θ on a term t can also be written θt. Substitutions will sometimes we explicitly written with a list of associations $\{X_1 \mapsto t_1, ..., X_n \mapsto t_n\}$ and sometimes simply $\{X := t\}$ if there is only one association.

A *renaming* is a substitution α such that $\alpha(X) \in V$ for all $X \in V$ and such that it is bijective. In particular, renamings are invertible and the inverse of a renaming α is written α^{-1}.

From two substitutions θ_1, θ_2, we can construct their composition $\theta_1 \circ \theta_2$ such that $(\theta_1 \circ \theta_2)t = \theta_1(\theta_2(t))$. The composition is associative [LMM88, Corollary 6]. It is then possible to write $\theta_1\theta_2\theta_3$ for either $(\theta_1 \circ \theta_2) \circ \theta_3$ or $\theta_1 \circ (\theta_2 \circ \theta_3)$.

§B.1.8 **Example.** Let $\theta := \{X \mapsto c, Y \mapsto c\}$ and $\psi := \{X \mapsto f(Y), Y \mapsto g(c, c)\}$. We have $\theta f(X) = f(\theta X) = f(c)$, $\psi g(X, Y) = g(\psi X, \psi Y) = g(f(Y), g(c, c))$ (subsitutions replace simultaneously and not sequentially, hence there is no clash between the two occurrences of Y) and $(\theta \circ \psi)g(X, Y) = g((\theta \circ \psi)X, (\theta \circ \psi)Y) = g((\theta \circ \psi)X, (\theta \circ \psi)Y) = g(\theta f(Y), \theta g(c, c)) = g(f(c), g(c, c))$.

§B.1.9 **Definition** (Equation and unification problem)**.** An *equation* is an unordered pair $t \stackrel{?}{=} u$ of terms in $\texttt{Term}(\mathbb{S})$.

A *unification problem* or simply *problem* is a set of equations $\{t_1 \stackrel{?}{=} u_1, ..., t_n \stackrel{?}{=} u_n\}$.

A *solution* or *unifier* for a problem P is a substitution θ such that for all $t \stackrel{?}{=} u \in P$, $\theta t = \theta u$. In this case, we say that the terms t and u are *unifiable* and that θ.

§B.1.10 **Example.** The problem $\{f(X, f(Y)) \stackrel{?}{=} f(g(c, c), Z)\}$ has for solution $\theta := \{X \mapsto g(c, c), Z \mapsto f(Y)\}$. The problems $\{f(X) \stackrel{?}{=} g(c, c)\}$ and $\{X \stackrel{?}{=} f(X)\}$ have no solution.

§B.1.11 **Definition** (Alpha-equivalence)**.** Two terms t and u are *α-equivalent*, written $t \approx_\alpha u$, if there exists a renaming α such that $t = \alpha(u)$.

§B.1.12 **Definition** (Alpha-unification)**.** An *α-unifier* for a problem $P = \{t_i \stackrel{?}{=} u_i\}_{1 \leq i \leq n}$ is a pair (θ, α) of a substitution θ and a renaming α such that θ is a solution for $\{t_i \stackrel{?}{=} \alpha u_i\}_{1 \leq i \leq n}$. Two terms t and u are *α-unifiable* if there exists an α-unifier for $\{t \stackrel{?}{=} u\}$.

§B.1.13 **Example.** A typical example is the problem $\{X \stackrel{?}{=} f(X)\}$ which has no unifier but has an α-unifier. A possible α-unifier is (θ, α) with $\theta := \{X \mapsto f(c), X' \mapsto c\}$ and $\alpha := \{X \mapsto X'\}$. We have $\theta X = f(c)$ and $\theta\alpha f(X) = \theta f(X') = f(c) = \theta X$.

§B.1.14 **Lemma** (Symmetry of α-unifiability)**.** Let t and u be two terms. We have that $\{t \stackrel{?}{=} u\}$ has an α-unifier if and only if $\{u \stackrel{?}{=} t\}$ has one.

Proof. To say that $\{t \stackrel{?}{=} u\}$ has an α-unifier means that there is some (θ, α) with θ a substitution and α a renaming such that $\theta t = \theta\alpha u$, meaning that t and u only differ by a renaming. We only show one implication and the other is similar. Assume $\theta t = \theta\alpha u$ (hypothesis H) for some (θ, α). We define $\theta' := \theta\alpha$ and would like to obtain $\theta'\alpha' t = \theta' u$ for some α'. We already know that $\theta t = \theta\alpha u$, hence we need to cancel the renaming α which is part of θ'. We define $\alpha' := \alpha^{-1}$. We finally have $\theta'\alpha' t = \theta\alpha\alpha^{-1} t = \theta t \stackrel{H}{=} \theta\alpha u = \theta' u$ (we used the fact that composition is associative, as stated in Definition B.1.7). □

§B.1.15 These definitions define a preorder on terms. A term t is lesser than another term u when it is more specialised or less general. In terms of substitutions, it means that t can be obtained by instantiating the variables of u with other terms.

§B.1.16 **Definition** (Generality relation on terms)**.** We define the following relation: given t, u two terms, $t \preceq u$ if and only if there exists a substitution θ such that $t = \theta u$.

§B.1.17 **Proposition.** The relation $\preceq$ defines a preorder.

Proof. Let t be a term. If θ is the identity substitution, we have $t = \theta t$. Let t_1, t_2, t_3 be terms. Assume $t_1 = \theta_a t_2$ and $t_2 = \theta_b t_3$. We can compose the two substitutions and obtain $\theta_a \circ \theta_b$. We have $(\theta_a \circ \theta_b) t_3 = \theta_a(\theta_b t_3) = \theta_a t_2 = t_1$. □

§B.1.18 The definition of α-unification comes from a simplification of Aubert and Bagnol's definition of *matching* [AB14, Definition 6] itself appearing in Girard's definitions [Gir13b, Section 1.1.2]. However, since *matching* already exists with a different definition in the literature, a different name is chosen. In the proposition below, the two variants are shown to be equivalent.

§B.1.19 **Proposition.** Two terms t_1 and t_2 are α-unifiable if and only if there exists two renamings α_1 and α_2 such that $\alpha_1 t_1$ and $\alpha_2 t_2$ are unifiable and that $\texttt{vars}(\alpha_1 t_1) \cap \texttt{vars}(\alpha_2 t_2) = \emptyset$.

Proof. ($\Rightarrow$) Assume that t_1 and t_2 are α-unifiable. Hence, we have $\theta\alpha t_1 = \theta t_2$ for some (θ, α). We have to find ψ, α_1 and α_2 such that $\psi\alpha_1 t_1 = \psi\alpha_2 t_2$ and $\texttt{vars}(\alpha_1 t_1) \cap \texttt{vars}(\alpha_2 t_2) = \emptyset$. We define $\alpha_2 := \alpha$. Providing there are enough variable symbols, it is possible to design α_1 so that $\alpha_1(X) \notin \texttt{vars}(\alpha_2 t_2)$ for all $X \in \texttt{vars}(t_1)$, *i.e.* $\texttt{img}(\alpha_1) \cap \texttt{vars}(t_2) = \emptyset$. We then have $\texttt{vars}(\alpha_1 t_1) \cap \texttt{vars}(\alpha_2 t_2) = \emptyset$. Now, $\alpha_1 t_1 \stackrel{?}{=} \alpha_2 t_2$ only differ from $t_1 \stackrel{?}{=} \alpha t_2$ by the fact that some variables of t_1 have been changed by α_1. We can define ψ so that it reverses the effect of α_1 without affecting $\alpha_2 t_2$. Hence, we have $\psi := \theta \circ \alpha_1^{-1}$. It has no effect on $\alpha_2 t_2$ because we assumed that $\texttt{img}(\alpha_1) \cap \texttt{vars}(t_2) = \emptyset$. ($\Leftarrow$) Assume that there exists two renamings α_1 and α_2 such that $\alpha_1 t_1$ and $\alpha_2 t_2$ are unifiable and that $\texttt{vars}(\alpha_1 t_1) \cap \texttt{vars}(\alpha_2 t_2) = \emptyset$. We have $\theta\alpha_1 t_1 = \theta\alpha_2 t_2$ for some θ. We can define the substitution $\psi := \theta \circ \alpha_2$ and the renaming $\alpha := \alpha_2^{-1} \circ \alpha_1$ such that $\psi\alpha t_1 = \theta\alpha_2\alpha_2^{-1}\alpha_1 t_1 = \theta\alpha_1 t_1 = \theta\alpha_2 t_2 = \psi t_2$. This shows that t_1 and t_2 are α-unifiable with (ψ, α). □

§B.1.20 It is also possible to define a relation $\leq$ on substitutions instead of terms. This relation is well-known as a preorder in the literature [BN98, Definition 4.5.1]. Its intuitive meaning is that $\theta \leq \psi$ is ψ is "more general" than θ.

§B.1.21 **Definition** (Generality relation on substitutions)**.** Let θ and ψ be two substitutions. We define the relation $\leq$ such that $\theta \leq \psi$ when there exists a substitution σ such that $\psi = \sigma\theta$.

§B.1.22 The problem of deciding if a solution to a given problem P exists is known to be decidable. Moreover, there exists a maximal solution $\texttt{solution}P$ *w.r.t.* the preorder $\preceq$ on substitutions, which is unique up to renaming. Several algorithms were designed to compute the unique solution when it exists, such that the Martelli-Montanari unification algorithm [MM82].

§B.1.23 **Definition** (Solved form)**.** A unification problem $P = \{t_i \stackrel{?}{=} u_i\}_{1 \leq i \leq n}$ is in *solved form* if:

- for $1 \leq i \leq n$, t_i is a variable $X_i \in V$ and
- no X_i appears in the right-hand side of an equation, *i.e.*

$$\{X_i\}_{1 \leq i \leq n} \cap \bigcup_{j=1}^{n} \texttt{vars}(t_j) = \emptyset.$$

The *underlying substitution* of P is defined by $\overrightarrow{P} := \{X_i \mapsto t_i\}_{1 \leq i \leq n}$.

B.2 Unification algorithm

§B.2.1 **Definition** (Rules for Martelli-Montanari algorithm)**.** The Martelli-Montanari algorithm is defined by a binary relation $\rightsquigarrow$ (in infix notation) over unification problems. It is defined as follows:

- ◇ **Clear** $P \cup \{t \stackrel{?}{=} t\} \overset{c}{\rightsquigarrow} P$;
- ◇ **Open** $P \cup \{f(t_1, ..., t_n) \stackrel{?}{=} g(u_1, ..., u_n)\} \overset{op}{\rightsquigarrow} P \cup \{t_1 \stackrel{?}{=} u_1, ..., t_n \stackrel{?}{=} u_n\}$ when $f \supset g$;
- ◇ **Orient** $P \cup \{t \stackrel{?}{=} X\} \overset{or}{\rightsquigarrow} P \cup \{X \stackrel{?}{=} t\}$ with $t \notin \mathtt{vars}(t)$;
- ◇ **Replace** $P \cup \{X \stackrel{?}{=} t\} \overset{r(X)}{\rightsquigarrow} \{X \mapsto t\}P \cup \{X \stackrel{?}{=} t\}$ with $X \in \mathtt{vars}(P)$ and $X \notin \mathtt{vars}(t)$.

We simply write $\rightsquigarrow$ when the rule is left implicit and we write $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$.

§B.2.2 **Definition** (Martelli-Montanari execution)**.** A (Martelli-Montanari) *(partial) execution* is a non-empty finite sequence of $\rho = (P_1, ..., P_{n+1})$ of unification problems such that $P_i \rightsquigarrow P_{i+1}$ for $1 \leq i \leq n$. The execution ρ is:

- *successful* if P_{n+1} is in solved form;
- *unsuccessful* if P_{n+1} is not in solved form;
- *full* if there is no P such that $P_{n+1} \rightsquigarrow P$.

§B.2.3 **Theorem** (Correctness and termination of unification algorithm)**.** There exists a full execution $\rho = (P_1, ..., P_n)$ with P_n in solved form if and only if P_1 has a solution. Otherwise, ρ is unsuccessful.

Proof. Proven in Lassez's article [LMM88, Theorem 3.1]. Other proofs are found in Baader and Nipkow's book [BN98, Lemma 4.6.5 & Lemma 4.6.7 & Lemma 4.6.10]. □

§B.2.4 **Theorem** (Confluence of the unification algorithm)**.** Let $\rho = (P_1, ..., P_n)$ be an execution starting from a solvable problem P_1. There exists an extension $\rho' = (Q_1, ..., Q_m)$ such that $\rho \cdot \rho' := (P_1, ..., P_n, Q_1, ..., Q_m)$ is successful.

Proof. This corresponds to a result of confluence in rewriting systems [BN98, Definition 2.1.3]. By Theorem B.2.3, we already know that the rewriting system associated to the Martelli-Montanari algorithm is terminating. In addition to this fact, we can use Newman's lemma [BN98, Lemma 2.7.2] by showing that the relation $\rightsquigarrow$ is locally confluent

(*cf.* [BN98, Lemma 2.7.1]) in order to show that it is confluent. Let P be a unification problem. We consider all the possible divergences of choices induced by the rules of Definition B.2.1.

◇ **Case of clear rule** Let P be a problem and x a rule such that $P \overset{x}{\rightsquigarrow} P'$. We have $P \cup \{t \overset{?}{=} t\} \overset{c}{\rightsquigarrow} P \overset{x}{\rightsquigarrow} P'$ and $P \cup \{t \overset{?}{=} t\} \overset{x}{\rightsquigarrow} P' \cup \{t \overset{?}{=} t\} \overset{c}{\rightsquigarrow} P'$.

◇ **Case of open rule** Let P be a problem and x a rule such that $P \overset{x}{\rightsquigarrow} P'$. We have

$$\begin{gathered} P \cup \{f(t_1, ..., t_n) \overset{?}{=} f(u_1, ..., u_n)\} \\ \overset{op}{\rightsquigarrow} P \cup \{t_1 \overset{?}{=} u_1, ..., t_n \overset{?}{=} u_n\} \\ \overset{x}{\rightsquigarrow} P' \cup \{t_1 \overset{?}{=} u_1, ..., t_n \overset{?}{=} u_n\} \end{gathered}$$

and

$$\begin{gathered} P \cup \{f(t_1, ..., t_n) \overset{?}{=} f(u_1, ..., u_n)\} \\ \overset{x}{\rightsquigarrow} P' \cup \{f(t_1, ..., t_n) \overset{?}{=} f(u_1, ..., u_n)\} \\ \overset{op}{\rightsquigarrow} P' \cup \{t_1 \overset{?}{=} u_1, ..., t_n \overset{?}{=} u_n\}. \end{gathered}$$

◇ **Case of orient rule** Let P be a problem and x a rule such that $P \overset{x}{\rightsquigarrow} P'$. We have $P \cup \{t \overset{?}{=} X\} \overset{or}{\rightsquigarrow} P \cup \{X \overset{?}{=} t\} \overset{x}{\rightsquigarrow} P' \cup \{X \overset{?}{=} t\}$ and $P \cup \{t \overset{?}{=} X\} \overset{x}{\rightsquigarrow} P' \cup \{t \overset{?}{=} X\} \overset{op}{\rightsquigarrow} P' \cup \{X \overset{?}{=} t\}$.

◇ **Case of replace** Let P be a problem and x a rule such that $P \overset{x}{\rightsquigarrow} P'$. We reason by case on x. For more clarity, we write θ for the substitution $\{X \mapsto t\}$.

- ◇ **Clear** We have $P \cup \{u \overset{?}{=} u, X \overset{?}{=} t\} \overset{c}{\rightsquigarrow} P \cup \{X \overset{?}{=} t\} \overset{r(X)}{\rightsquigarrow} \theta P \cup \{X \overset{?}{=} t\}$ and $P \cup \{u \overset{?}{=} u, X \overset{?}{=} t\} \overset{r(X)}{\rightsquigarrow} \theta P \cup \{\theta u \overset{?}{=} \theta u, X \overset{?}{=} t\} \overset{c}{\rightsquigarrow} \theta P \cup \{X \overset{?}{=} t\}$.
- ◇ **Open** We have

$$\begin{gathered} P \cup \{f(t_1, ..., t_n) \overset{?}{=} f(u_1, ..., u_n), X \overset{?}{=} t\} \\ \overset{op}{\rightsquigarrow} P \cup \{t_1 \overset{?}{=} u_1, ..., t_n \overset{?}{=} u_n, X \overset{?}{=} t\} \\ \overset{r(X)}{\rightsquigarrow} \theta P \cup \{\theta t_1 \overset{?}{=} \theta u_1, ..., \theta t_n \overset{?}{=} \theta u_n, X \overset{?}{=} t\} \end{gathered}$$

and

$$\begin{gathered} P \cup \{f(t_1, ..., t_n) \overset{?}{=} f(u_1, ..., u_n), X \overset{?}{=} t\} \\ \overset{r(X)}{\rightsquigarrow} \theta P \cup \{\theta f(t_1, ..., t_n) \overset{?}{=} \theta f(u_1, ..., u_n), X \overset{?}{=} t\} \\ = \theta P \cup \{f(\theta t_1, ..., \theta t_n) \overset{?}{=} f(\theta u_1, ..., \theta u_n), X \overset{?}{=} t\} \\ \overset{op}{\rightsquigarrow} \theta P \cup \{\theta t_1 \overset{?}{=} \theta u_1, ..., \theta t_n \overset{?}{=} \theta u_n, X \overset{?}{=} t\}. \end{gathered}$$

◇ **Orient** Assume that we have a term u which is not a variable. We have $P \cup \{u \stackrel{?}{=} Y, X \stackrel{?}{=} t\} \overset{or}{\rightsquigarrow} P \cup \{Y \stackrel{?}{=} u, X \stackrel{?}{=} t\} \overset{r(X)}{\rightsquigarrow} \theta P \cup \{\theta Y \stackrel{?}{=} \theta u, X \stackrel{?}{=} t\}$ and $P \cup \{u \stackrel{?}{=} Y, X \stackrel{?}{=} t\} \overset{r(X)}{\rightsquigarrow} \theta P \cup \{\theta u \stackrel{?}{=} \theta Y, X \stackrel{?}{=} t\} \overset{or}{\rightsquigarrow} \theta P \cup \{\theta Y \stackrel{?}{=} \theta u, X \stackrel{?}{=} t\}$. The orient rule in the second reduction works because the substitution θ does not change the nature of Y and u. The first is still a variable and the second still not a variable.

◇ **Replace** There are two cases. We write ψ for the substitution $\{Y \mapsto u\}$.

- Assume that $X = Y$. We have

$$P \cup \{Y \stackrel{?}{=} u, X \stackrel{?}{=} t\}$$
$$\overset{r(X)}{\rightsquigarrow} \theta P \cup \{\theta Y \stackrel{?}{=} \theta u, X \stackrel{?}{=} t\}$$
$$= \theta P \cup \{t \stackrel{?}{=} \theta u, X \stackrel{?}{=} t\}$$
and
$$P \cup \{Y \stackrel{?}{=} u, X \stackrel{?}{=} t\}$$
$$\overset{r(Y)}{\rightsquigarrow} \psi P \cup \{\psi X \stackrel{?}{=} \psi t, Y \stackrel{?}{=} u\}$$
$$= \psi P \cup \{u \stackrel{?}{=} \psi t, Y \stackrel{?}{=} u\}.$$

  It is then obvious that $t \stackrel{?}{=} u$ and $u \stackrel{?}{=} u$

- Assume that $X \neq Y$. We define $\psi' := \{Y \mapsto \theta u\}$ and $\theta' := \{X \mapsto \psi t\}$. We have

$$P \cup \{Y \stackrel{?}{=} u, X \stackrel{?}{=} t\}$$
$$\overset{r(X)}{\rightsquigarrow} \theta P \cup \{Y \stackrel{?}{=} \theta u, X \stackrel{?}{=} t\}$$
$$\overset{r(Y)}{\rightsquigarrow} \psi' \theta P \cup \{X \stackrel{?}{=} \psi' t, Y \stackrel{?}{=} \theta u\}$$
and
$$P \cup \{Y \stackrel{?}{=} u, X \stackrel{?}{=} t\}$$
$$\overset{r(Y)}{\rightsquigarrow} \psi P \cup \{X \stackrel{?}{=} \psi t, Y \stackrel{?}{=} u\}$$
$$\overset{r(X)}{\rightsquigarrow} \theta' \psi P \cup \{Y \stackrel{?}{=} \theta' u, X \stackrel{?}{=} \psi t\}.$$

□

§B.2.5 **Corollary.** Let $\rho = (P_1, ..., P_n)$ be an execution starting from a solvable problem P_1. There exists an extension $\rho' = (Q_1, ..., Q_m)$ such that $\rho \cdot \rho' := (P_1, ..., P_n, Q_1, ..., Q_m)$ is successful.

Proof. By correctness and termination of the unification algorithm (*cf.* Theorem B.2.3), since P_1 is solvable, there exists an execution from $P_1 \rightsquigarrow^* P_f$. Assume that we already started the execution with some arbitrary but valid steps $(P_1, ..., P_n)$. By confluence of the unification algorithm (*cf.* Theorem B.2.4) □

§B.2.6 **Theorem** (Unicity of solution)**.** If $\rho = (P_1, ..., P_n)$ is a full execution starting from a solvable problem P_1, then $\overrightarrow{P_n}$ is the unique solution of P_1 modulo $\approx_\alpha$.

Proof. Indirectly proven in Lassez's article [LMM88, Theorem 3.17]. The original statement says that any solvable problem P has a unique solution modulo $\approx_\alpha$ but the Martelli-Montanari algorithm computes such a solution by correctness of the algorithm. □

§B.2.7 **Corollary** (Execution lifting)**.** Let $(P, P_1, ..., P_n)$ be an execution. For all P' such that $P \subseteq P'$, there exists an execution $(P', P'_1, ..., P'_n)$ such that $P_i \subseteq P'_i$ for $1 \leq i \leq n$.

Proof. By confluence of the unification algorithm (*cf.* Theorem B.2.4), all paths of execution ultimately lead to a unique result modulo $\approx_\alpha$. We decompose P' into $P \cup P''$ (since it contains P). It is possible to focus only on the equation in P without impact on the result. We can then construct the execution $P \cup P'' \rightsquigarrow P_1 \cup P'' \rightsquigarrow^* P_n \cup P''$ corresponding to the execution $P' \rightsquigarrow P'_1 \rightsquigarrow^* P'_n$. □

Appendix C

Graph theory

C.1 Non-directed hypergraphs

Intuitively, hypergraphs are points in space called *vertices* such that some of them are linked together with *hyperedges*. An example of hypergraph is given in Figure C.1.1. Vertices are written v_i and hyperedges e_i for $i \in \mathbf{N}$.

§C.1.1 **Definition** (Hypergraph)**.** A hypergraph H is a tuple $(V, E, \texttt{end})$ of a set V of elements called *vertices*, a set E of elements called *hyperedges* and a function $\texttt{end} : E \to \mathcal{P}(V)$ mapping hyperedges to their associated *endpoints sets*, such that $\texttt{end}(e) \neq \emptyset$ for all $e \in E$.

§C.1.2 **Example.** The formal definition of the hypergraph illustrated in Figure C.1.1 is given by a tuple $H = (V, E, \texttt{end})$ such that $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, $E = \{e_1, e_2, e_3, e_4\}$ and $\texttt{end}(e_1) = \{v_1, v_2, v_3\}, \texttt{end}(e_2) = \{v_2, v_3\}, \texttt{end}(e_3) = \{v_3, v_5, v_6\}, \texttt{end}(e_4) = \{v_4\}$.

§C.1.3 **Definition** (Properties of hypergraphs)**.** Let $H = (V, E, \texttt{end})$ be a hypergraph.

- The *order* of H is defined by $|V|$;
- The *size* of H is defined by $|E|$;
- The *rank* of H, written $\texttt{rank}(H)$, is the maximal cardinality appearing in E, *i.e.* $\texttt{max}\{|\texttt{end}(e)| \text{ such that } e \in E\}$.

§C.1.4 **Definition** (Adjacency and incidence)**.** Let $H = (V, E, \texttt{end})$ be a hypergraph. Two vertices $x, y \in V$ are *adjacent* when there exists $e \in E$ such that $x, y \in \texttt{end}(e)$.

Two hyperedges $e, e' \in E$ are *incident* when $\texttt{end}(e) \cap \texttt{end}(e') \neq \emptyset$.

§C.1.5 **Example.** In Figure C.1.1, v_1 and v_2 are adjacent because $v_1, v_2 \in \texttt{end}(e_1)$. The hyperedges e_1 and e_2 are incident because $\texttt{end}(e_1) \cap \texttt{end}(e_2) = \{v_2, v_3\}$. However, v_1 and v_4 are not adjacent since no hyperedge link them and, e_1 and e_4 are not incident because they do not link common vertices.

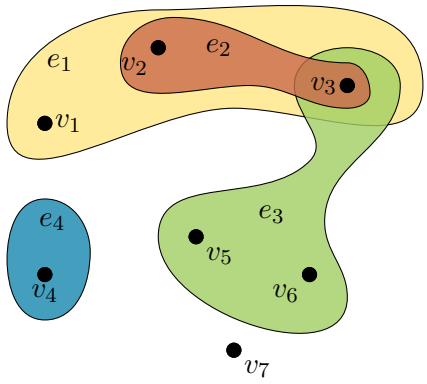


Figure C.1.1: Example of hypergraph taken from a StackExchange post: https://tex.stackexchange.com/questions/1175/drawing-a-hypergraph.

§C.1.6 **Definition** (Simple hypergraph)**.** A hypergraph $H = (V, E, \texttt{end})$ is *simple* when for all $e, e' \in E$ such that $\texttt{end}(e) \subseteq \texttt{end}(e')$, we have $i = j$.

§C.1.7 **Example.** The hypergraph of Figure C.1.1 is not simple because $\texttt{end}(e_2) \subseteq \texttt{end}(e_1)$ and $e_1 \neq e_2$. The point is to remove redundant links. Since e_1 already links v_2 and v_3 together, it is possible to forget the hyperedge e_2.

§C.1.8 **Definition** (Path)**.** Let $H = (V, E, \texttt{end})$ be a hypergraph. A *path* ρ in H from $v_1 \in V$ to $v_{n+1} \in V$ is an alternate sequence of vertices and hyperedges

$$\rho = (v_1, e_1, v_2, e_2, ..., v_n, e_n, v_{n+1})$$

where:

- the v_i are pairwise distinct with the exception of the pair (v_1, v_{n+1}), *i.e.* we can have $v_1 = v_{n+1}$;
- the e_i are all distinct;
- for all $1 \leq i \leq n$, we have $v_i, v_{i+1} \in \texttt{end}(e_i)$.

The *length* of the path is $|\rho| = n$. If $x_1 = x_{n+1}$, then ρ is called a *cycle*.

§C.1.9 **Example.** A possible path in the hypergraph of Figure C.1.1 from v_1 to v_3 is

$$\rho := (v_1, e_1, v_2, e_2, v_3).$$

A possible cycle (which is a loop) is (v_4).

§C.1.10 **Definition** (Cyclicity)**.** A hypergraph containing at least one cycle is called *cyclic*. Otherwise (if is has no cycles), it is *acyclic*.

There exists several notions of acyclicity (α-acyclicity and β-acyclicity for instance) but we will not need them in this thesis.

§C.1.11 **Definition** (Connectedness)**.** A hypergraph $H = (V, E, \texttt{end})$ is *connected* when there is a path between all pairs of vertices in V. Otherwise, it is *disconnected*.

§C.1.12 **Definition** (Hypergraph homomorphism)**.** A *homomorphism* f between two hypergraphs $H = (V_H, E_H, \texttt{end}_H)$ and $H' = (V_{H'}, E_{H'}, \texttt{end}_{H'})$ is given by the two following *underlying* maps with same name:

- $f : V_H \to V_{H'}$ (the vertex map) and
- $f : E_H \to E_{H'}$ (the edge map)

such that $f(\texttt{end}_H(e)) = \texttt{end}_{H'}(f(e))$ for all $e \in E_H$, *i.e.* it maps adjacent vertices to adjacent vertices.

§C.1.13 **Definition** (Injectivity and surjectivity of homomorphisms)**.** Let $f : H \to H'$ be a hypergraph homomorphism. It is:

- *injective* (or an injection) when its underlying maps are injective, *i.e.* $f(x) = f(x')$ implies $x = x'$ for x, x' edges or vertices of H;
- *surjective* (or a surjection) when its underlying maps are surjective, *i.e.* for every y (edge or vertex of H'), there is some x that generates it, *i.e.* there is some x such that $f(x) = y$.

§C.1.14 **Definition** (Hypergraph isomorphism)**.** Let f be a hypergraph homomorphism between two hypergraph homomorphism H and H'. It is a *hypergraph isomorphism* if and only if its underlying maps are bijective, *i.e.* both injective and surjective. We then say that H and H' are *isomorphic*, written $H \simeq H'$.

Since the underlying maps of an isomorphism are bijective, it induces an inverse isomorphism in the same way that bijections induce inverse functions.

§C.1.15 **Proposition.** A hypergraph homomorphism f between two hypergraphs

$$H = (V_H, E_H, \texttt{end}_H) \quad \text{and} \quad H' = (V_{H'}, E_{H'}, \texttt{end}_{H'})$$

is an isomorphism if and only if there exists a bijection f^{-1} such that $f \circ f^{-1} = f^{-1} \circ f = id_H$ where id_H is the identity morphism over H defined by the two identity maps $f : V_H \to V_H$ such that $f(v) = v$ and $f : E_H \to E_H$ such that $f(e) = e$.

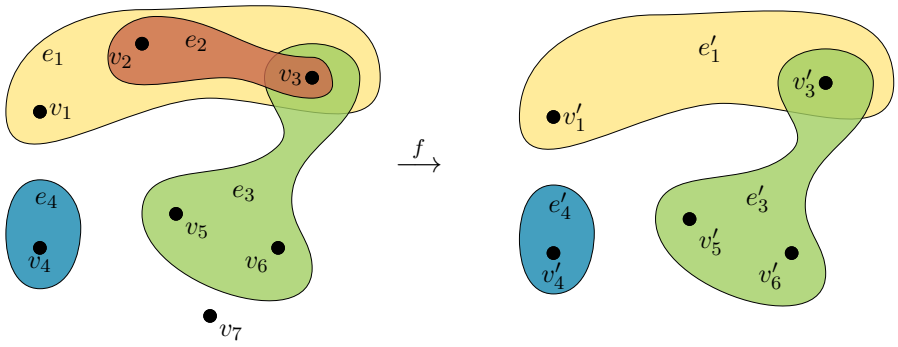


Figure C.1.2: A hypergraph homomorphism f relating two hypergraphs such that $f(v_1) = f(v_2) = v'_1$, $f(v_3) = v'_3$, $f(v_4) = f(v_7) = v'_4$, $f(v_5) = v'_5$, $f(v_6) = v'_6$, $f(e_1) = f(e_2) = e'_1$, $f(e_3) = e'_3$ and $f(e_4) = e'_4$.

§C.1.16 **Example.** A hypergraph homomorphism is illustrated in Figure C.1.2. It makes the hyperedge e_2 linking v_2 and v_3 disappear by merging it with e_1. We have $f(\texttt{end}e_2) = f(\{v_2, v_3\}) = \{v'_1, v'_3\} = \texttt{end}(e'_1) = \texttt{end}(f(e_2))$. The isolated vertex v_7 also disappears.

In the case of an isomorphisms, we expect H and $f(H)$ to be structurally equivalent. The typical example maps a hypergraph to the same graph up to renaming of vertices and hyperedges.

C.2 Directed hypergraphs

It is possible to consider directions in hypergraphs by considering that hyperedges have inputs and outputs.

§C.2.1 **Definition** (Directed hypergraph)**.** A *directed hypergraph* is a tuple

$$H = (V, E, \texttt{in}, \texttt{out})$$

where V is the set of vertices, E the set of hyperedges (called *hyperarcs*) and the two functions $\texttt{in} : E \to \mathcal{P}(V)$ and $\texttt{out} : E \to \mathcal{P}(V)$ are respectively the set of inputs and outputs associated to the hyperarc e.

Remark that we allow hyperarcs with no input or no output (for a hyperarc e, we may have $\texttt{in}(e) = \emptyset$ or $\texttt{out}(e) = \emptyset$).

§C.2.2 **Definition** (Directed path)**.** Let $H = (V, E, \texttt{in}, \texttt{out})$ be a directed hypergraph. A *path* ρ in H from $v_1 \in V$ to $v_{n+1} \in V$ is an alternate sequence of vertices and hyperedges

$$\rho = (v_1, e_1, v_2, e_2, ..., v_n, e_n, v_{n+1})$$

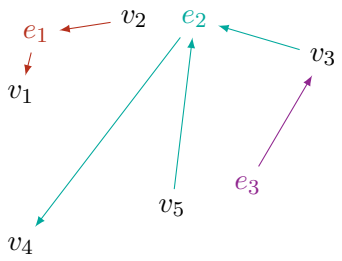


Figure C.2.1: Test.

where:

- the v_i are pairwise distinct with the exception of the pair (v_1, v_{n+1}), *i.e.* we can have $v_1 = v_{n+1}$;
- the e_i are all distinct;
- for all $1 \leq i \leq n$, we have $v_i \in \mathtt{in}(e_i)$ and $v_{i+1} \in \mathtt{out}(e_{i+1})$.

The *length* of the path is $|\rho| = n$. If $v_1 = v_{n+1}$, then ρ is called a *circuit* (or n-circuit if we want to make the length explicit).

It is possible to recover undirected graphs by considering symmetric directions.

§C.2.3 **Definition** (Symmetric hypergraph)**.** Let $H = (V, E, \mathtt{in}, \mathtt{out})$ be a directed hypergraph. It is *symmetric* if for each vertices v, v', there are two hyperedges e, e' such that $v \in \mathtt{in}(e)$, $v' \in \mathtt{out}(e)$, $v' \in \mathtt{in}(e)$ and $v \in \mathtt{out}(e)$.

A hypergraph can also be given an *orientation* which should be understood as a strict direction, *i.e.* edges go in one specific direction without leaving the possibility of going back.

§C.2.4 **Definition** (Oriented hypergraph)**.** A directed hypergraph is *oriented* when it contains no symmetry, *i.e.* there is no vertices v, v' and edges e, e' such that $v \in \mathtt{in}(e)$, $v' \in \mathtt{out}(e)$, $v' \in \mathtt{in}(e)$ and $v \in \mathtt{out}(e)$.

§C.2.5 **Example.** In Figure C.2.1, we have an example of directed hypergraph

$$H = (V, E, \mathtt{in}, \mathtt{out})$$

such that $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1, e_2, e_3\}$ with:

- $\mathtt{in}(e_1) = \{v_2\}$ and $\mathtt{out}(e_1) = \{v_1\}$;
- $\mathtt{in}(e_2) = \{v_3, v_5\}$ and $\mathtt{out}(e_2) = \{v_4\}$;
- $\mathtt{in}(e_3) = \emptyset$ and $\mathtt{out}(e_3) = \{v_3\}$.

Directed hypergraph homomorphisms must preserve inputs and ouputs.

§C.2.6 **Definition** (Directed hypergraph homomorphism)**.** A *homomorphism* f between two directed hypergraphs $H = (V, E, \texttt{in}_H, \texttt{out}_H)$ and $H' = (V_{H'}, E_{H'}, \texttt{in}_{H'}, \texttt{out}_{H'})$ is given by the two following maps with same name:

- $f : V_H \to V_{H'}$ (the vertex map) and
- $f : E_H \to E_{H'}$ (the edge map)

such that $f(\texttt{in}_H(e)) = \texttt{in}_{H'}(f(e))$ and $f(\texttt{out}_H(e)) = \texttt{out}_{H'}(f(e))$ for all $e \in E_H$, *i.e.* the functions f maps components (vertices and hyperarcs) of H to components of H' by preserving the linking of vertices by hyperarcs.

It is sometimes useful to consider an order in inputs and outputs by extending directed hypergraphs to *ordered* directed hypergraphs.

§C.2.7 **Definition** (Ordered directed hypergraph)**.** An *ordered directed hypergraph* $H = (V, E, \texttt{in}, \texttt{out}, \leq)$ is a directed hypergraph $(V, E, \texttt{in}, \texttt{out})$ extended with an order relation $\leq$ over $V \times V$.

§C.2.8 **Notation.** We write $\{u_1, ..., u_n\} \overset{e}{\mapsto} \{v_1, ..., v_m\}$ for an hyperedge e such that $\texttt{in}(e) = \{u_1, ..., u_n\}$ and $\texttt{out}(e) = \{v_1, ..., v_m\}$.

In case inputs and outputs are ordered (when we have an ordered hypergraph), then we write $[u_1, ..., u_n] \overset{e}{\mapsto} [v_1, ..., v_m]$ instead.

C.3 Special cases of hypergraphs

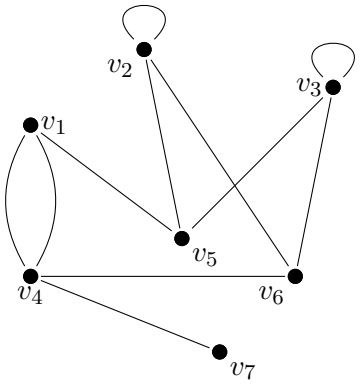
Multigraphs and graphs

§C.3.1 **Definition** (Multigraph)**.** A *multigraph* is a hypergraph H such that $\texttt{rank}(H) \leq 2$ (hyperedges only link two vertices or are loops over a vertex). Hyperedges are called *edges* in the case of a multigraph.

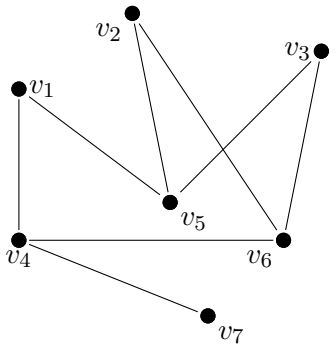
§C.3.2 **Definition** (Graph)**.** A loop in a hypergraph $H = (V, E, \texttt{end})$ is an edge e such that $|\texttt{end}(e)| = 1$. A *graph* is a simple multigraph H without loop.

§C.3.3 **Example.** An example of multigraph is illustrated in Figure C.3.1a and an example of graph is illustrated in Figure C.3.1b. The lines between vertices are *edges*. For instance, in Figure C.3.1a, v_1 and v_5 are linked by some hyperedge e such that $\texttt{end}(e) = \{v_1, v_5\}$, and v_2 is connected to itself by a loop, which is a hyperedge e' such that $\texttt{end}(e') = \{v_2\}$.

In the directed case, we simply write arrows instead of lines between vertices.



(a) Example of (connected) multigraph.



(b) Example of (connected) graph.

Figure C.3.1: Example of special cases of hypergraph.

§C.3.4 **Notation.** We write $u \overset{e}{\mapsto} v$ for a directed edge e such that $\texttt{in}(e) = u$ and $\texttt{out}(e) = v$.

Bipartite multigraph

A bipartite multigraph is a multigraph in which the set of vertices can be divided into two disjoint parts such that all edges link an element from one part to the other.

§C.3.5 **Definition** (Bipartite multigraph)**.** A *bipartite multigraph* $G = (V, E, \texttt{end})$ is a multigraph (V, E) such that $V = V_1 \uplus V_2$ and for each $e \in E$, we have $\texttt{end}(e) = \{v_1, v_2\}$ such that $v_1 \in V_1$ and $v_2 \in V_2$.

Square grid graph

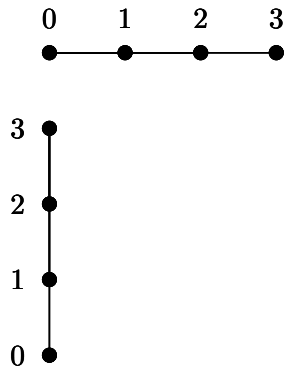
§C.3.6 **Definition** (Line graph)**.** A *line graph* or *path graph* is a graph $G = (V, E, \texttt{end})$ where V is an ordered sequence of vertices $[v_1, ..., v_n]$ with $n \geq 1$ and E is an ordered sequence of edges $[e_1, ..., e_{n-1}]$ such that $v_i, v_{i+1} \in \texttt{end}(e_i)$.

§C.3.7 **Definition** (Cartesian product of graphs)**.** Let

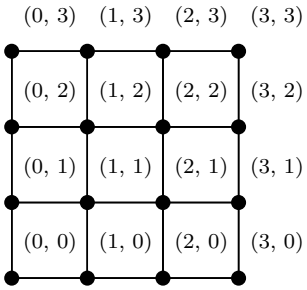
$$G = (V_G, H_G, \texttt{end}_G) \quad \text{and} \quad G' = (V_{G'}, H_{G'}, \texttt{end}_{G'})$$

be two graphs. Their *cartesian product* is the graph $G \square G' = (V_{G\square G'}, E_{G\square G'}, \texttt{end}_{G\square G'})$ such that $V_{G\square G'} = V_G \times V_{G'}$ and two vertices (u, v) and (u', v') are adjacent when either $u = u'$ and v is adjacent to v', or $v = v'$ and u is adjacent to u'.

§C.3.8 **Definition** (Square grid graph)**.** Let G be a line graph of size n. The associated *grid graph* is given by the graph product $G \square G$.



(a) Two line/path graphs.



(b) Result of the product of the two line graphs: a square grid graph.

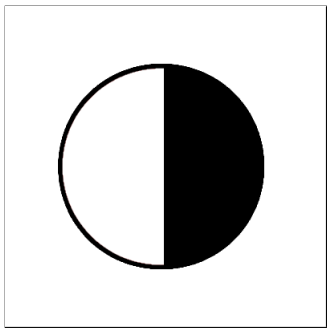
Figure C.3.2: Example of square grid graph.

An example of line graph and grid graph is given in Figure C.3.2. Grid graphs can be used to represent a finite subset of a set E^2 and more typically, the plane $\mathbf{Z}^2$.

Appendix D

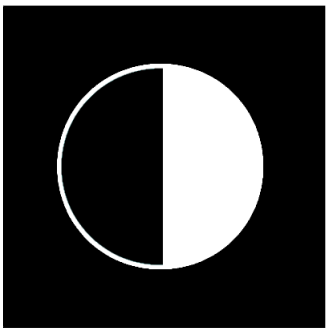
Transcendental aesthetics

In this section, I propose to revisit Kant's transcendental aesthetics in light of Girard's transcendental syntax. You will find a sequence of pictorial entities accessible through the pure intuition of (pixel) space and (waste of) time. It could be made into an NFT collection but I will freely share these pieces of art instead. It is inspired by Girard's "pure waste of paper" [Gir01, Appendix A]. This section itself may be seen as a pure waste of paper as well.



$[+a(X), +b(X)]$

Identity.

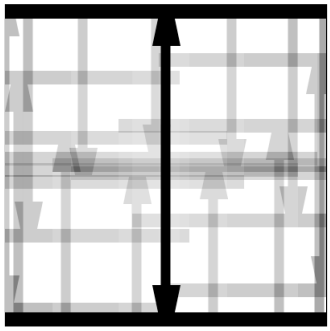


$[-a(X), a(X)] + [-b(X), b(X)]$

Test for identity.

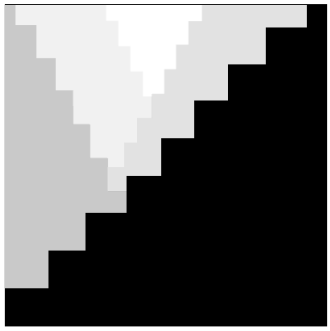
$[-\omega(X), +\omega(f(X))]$

Black hole.



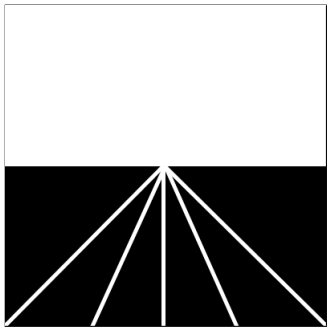
$[+a(X), +b(X)] + [-c(X), -d(X)] + [+e(X), -f(X)]$

Adapters.



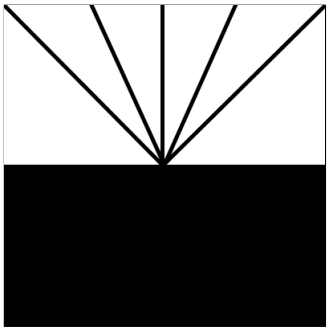
$[+nat(0)] + [-nat(X), +nat(s(X))] + [-nat(X), nat(X)]$

Natural numbers.



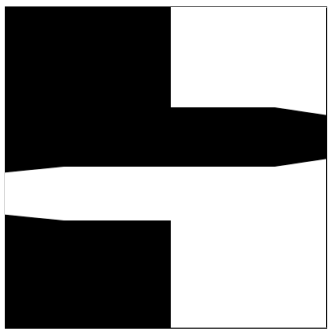
$[+a(X), +b(X)] + [-a(X), -b(X), 1]$

Rhizome.



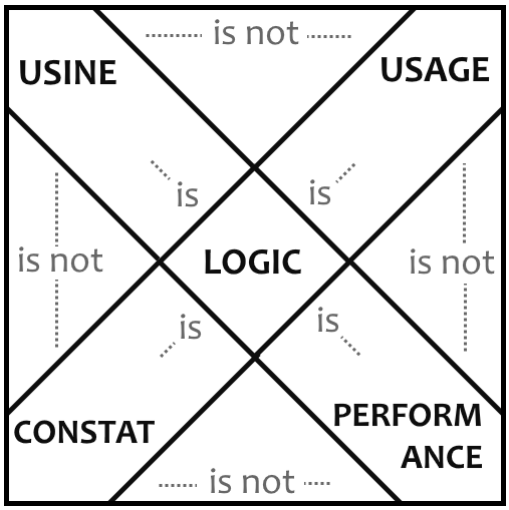
$[+a(+b(t))] + [-a(X), X]$

True digging.

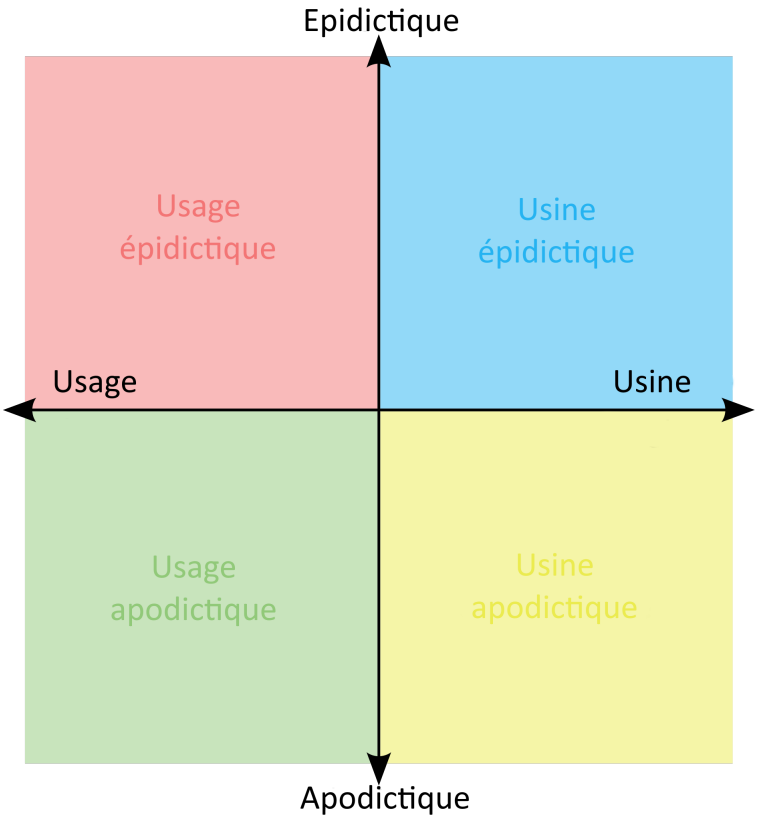


$[..., +a(X)] + [-a(X), -b(X)] + [+b(X), ...]$

Mutual understanding.



The Holy Quaternity of Logic.



The Synthetic Compass.

Menu de Maestracci

-10% de réduction pour les chercheurs en informatique théorique

Entrée – 4€

Pâtes salées avant ébullition ou
Salade de connecteurs Broccoli (sauce à la Tarski) ou
Camembert mou à la Frege

Plat – 14€

Viande au feu classique (à la moutarde de montre) ou
Steak tartare linéaire ($\star$ spécialité du chef)

Dessert (sorbet sémantique) – 5€

Parfum poire ou
Parfum poire+vanille+lait ou
Sorbet chaud à la tortue ou
Sorbet de petit salé aux lentilles

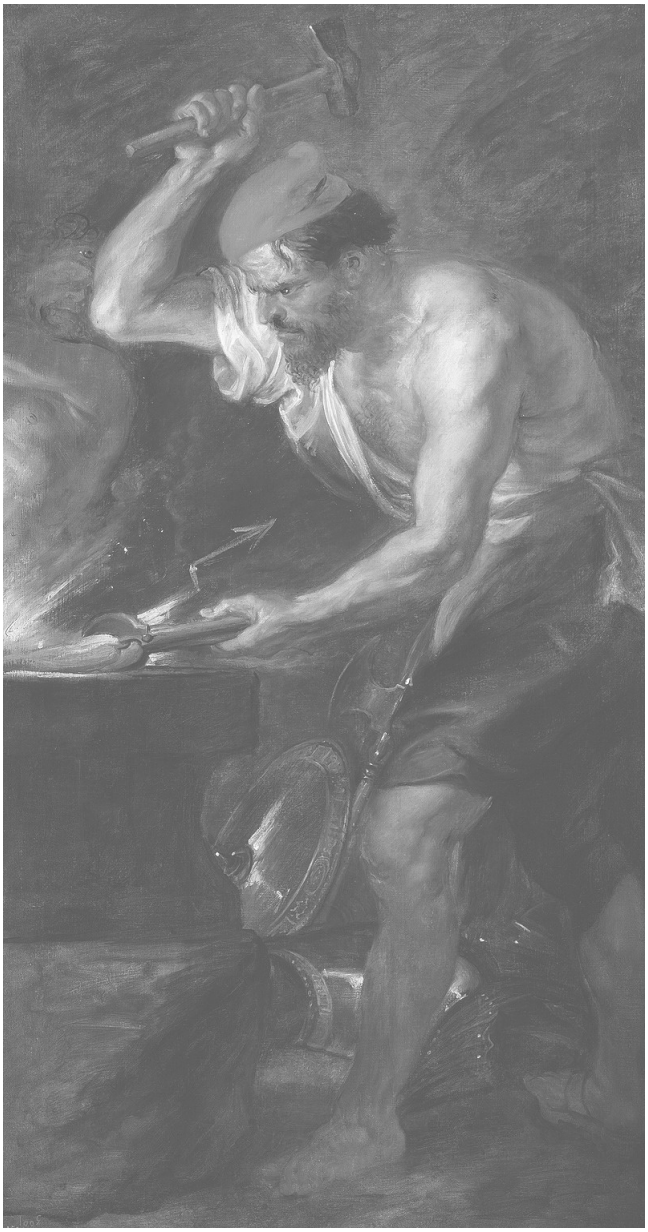
Boissons – Gratuit
(à volonté avec carte de fidélité sinon un seul service)

Bière 1L
Vin 1L

Maestracci's menu.

- Pâtes salées avant ébullition : *"Le fantôme de la transparence"*;
- Salade de connecteurs Broccoli : *"On the meaning of logical rules I : syntax vs. semantics"*;
- Camembert mou à la Frege : *"Proofs and Types"*;
- Moutarde de montre : *"Mustard watches, an integrated approach to time and food"*;
- Viande au feu classique : *"Shrodinger's cut"*;
- Steak tartare linéaire : *"Shrodinger's cut"*;
- Sorbet sémantique : *"La syntaxe transcendantale, un manifeste"*.

References for Maestracci's menu.



Vulcan forging the Thunderbolts of Jupiter (Rubens)

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. doi:10.1017/CBO9780511804090.

[AB14] Clément Aubert and Marc Bagnol. Unification and logarithmic space. In *Rewriting and Typed Lambda Calculi*, pages 77–92. Springer, 2014. doi:10.1007/978-3-319-08918-8_6.

[ABCJ98] David Albrecht, Frank A. Bäuerle, John N. Crossley, and John S. Jeavons. *Curry-Howard terms for linear logic*, volume 61. Springer, 1998. URL: http://www.jstor.org/stable/20016001.

[Abe06] Paul Abelson. *The seven liberal arts: A study in mediaeval culture*. Teachers' College, Columbia University, 1906.

[Abr91] V. Michele Abrusci. Phase semantics and sequent calculus for pure noncommutative classical linear propositional logic. *The Journal of Symbolic Logic*, 56(4):1403–1451, 1991. doi:10.2307/2275485.

[Abr94] Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994. URL: https://www.sciencedirect.com/science/article/pii/0304397594001030, doi:https://doi.org/10.1016/0304-3975(94)00103-0.

[Abr16] Samson Abramsky. Information, processes and games. *Philosophy of Information*, 2016. URL: https://arxiv.org/abs/1604.02603, doi:10.48550/arXiv.1604.02603.

[ABS16] Clément Aubert, Marc Bagnol, and Thomas Seiller. Unary resolution: Characterizing PTIME. In *International Conference on Foundations of Software Science and Computation Structures*, pages 373–389. Springer, 2016. doi:10.1007/978-3-662-49630-5_22.

[Acc18] Beniamino Accattoli. Proof nets and the linear substitution calculus. In *International Colloquium on Theoretical Aspects of Computing*, pages 37–61. Springer, 2018. doi:10.1007/978-3-030-02508-3_3.

[ACCL91] Martin Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of functional programming*, 1(4):375–416, 1991. URL: https://www.cambridge.org/

core/journals/journal-of-functional-programming/article/explicit-substitutions/C1B1AFAE8F34C953C1B2DF3C2D4C2125, doi:10.1017/S0956796800000186.

[ACJ97] David Albrecht, John N Crossley, and John S. Jeavons. New Curry-Howard terms for full linear logic. *Theoretical computer science*, 185(2):217–235, 1997. URL: https://www.sciencedirect.com/science/article/pii/S0304397597000443, doi:10.1016/S0304-3975(97)00044-3.

[AJ94] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *The Journal of Symbolic Logic*, 59(2):543–574, 1994. doi:10.2307/2275407.

[AK12] Jesse Alama and Johannes Korbmacher. The lambda calculus. 2012. URL: https://plato.stanford.edu/entries/lambda-calculus/.

[AL95] Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995. URL: https://www.sciencedirect.com/science/article/pii/0304397594002797, doi:10.1016/0304-3975(94)00279-7.

[AM20] Matteo Acclavio and Roberto Maieli. Generalized connectives for multiplicative linear logic. In *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. URL: https://drops.dagstuhl.de/opus/volltexte/2020/11649/, doi:10.4230/LIPIcs.CSL.2020.6.

[Ama16] Roberto Amadio. Operational methods in semantics. 2016. URL: https://hal.science/cel-01422101v2/.

[And76] Andrews. Refutations by matings. *IEEE transactions on computers*, 100(8):801–807, 1976. URL: https://ieeexplore.ieee.org/document/1674698, doi:10.1109/TC.1976.1674698.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of logic and computation*, 2(3):297–347, 1992. doi:doi.org/10.1093/logcom/2.3.297.

[AP14] V. Michele Abrusci and Paolo Pistone. On transcendental syntax: A kantian program for logic? 2014. URL: https://www.academia.edu/10495057/On_Trascendental_syntax_a_Kantian_program_for_logic.

[AR99] V. Michele Abrusci and Paul Ruet. Non-commutative logic I: the multiplicative fragment. *Annals of pure and applied logic*, 101(1):29–64, 1999. URL: https://www.sciencedirect.com/science/article/pii/S0168007299000147, doi:10.1016/S0168-0072(99)00014-7.

[AS16a] Clément Aubert and Thomas Seiller. Characterizing co-NL by a group action. *Mathematical Structures in Computer Science*, 26(4):606–638, 2016. URL: https://hal.science/hal-01005705/, doi:10.1017/S0960129514000267.

[AS16b] Clément Aubert and Thomas Seiller. Logarithmic space and permutations. *Information and Computation*, 248:2–21, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0890540115001364, doi:10.1016/j.ic.2014.01.018.

[Bag14] Marc Bagnol. *On the resolution semiring*. PhD thesis, Aix-Marseille Université, 2014. URL: https://www.normalesup.org/~bagnol/phd/these_screen.pdf.

[Bar91] Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1(2):159–178, 1991. doi:10.1017/S0960129500001274.

[Bar01] Chris Barker. Iota and Jot: the simplest languages? *The Esoteric Programming Languages Webring*, 2001. URL: http://semarch.linguistics.fas.nyu.edu/barker/Iota/.

[Bar21] Davide Barbarossa. *Towards a resource based approximation theory of programs*. PhD thesis, Université Sorbonne Paris Nord, 2021. URL: https://theses.hal.science/tel-03886068v1.

[BC06] Normand Baillargeon and Charb. *Petit cours d'autodéfense intellectuelle*. Lux, 2006.

[BC18] Mikołaj Bojańczyk and Wojciech Czerwiński. Automata toolbox. 2018.

[BDS15] Marc Bagnol, Amina Doumane, and Alexis Saurin. On the dependencies of logical rules. In *International Conference on Foundations of Software Science and Computation Structures*, pages 436–450. Springer, 2015. doi:10.1007/978-3-662-46678-0_28.

[Bef06] Emmanuel Beffara. A concurrent model for linear logic. *Electronic Notes in Theoretical Computer Science*, 155:147–168, 2006. URL: https://www.sciencedirect.com/science/article/pii/S1571066106001927, doi:10.1016/j.entcs.2005.11.055.

[Bel62] Nuel D. Belnap. Tonk, plonk and plink. *Analysis*, 22(6):130–134, 1962. doi:10.2307/3326862.

[Ber66] Robert Berger. *The undecidability of the domino problem*. American Mathematical Soc., 1966. doi:10.1090/MEMO/0066.

[BF97] Cesare Burali-Forti. Una questione sui numeri transfiniti. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 11(1):154–164, 1897. doi:10.1007/BF03015911.

[BF11] Michele Basaldella and Claudia Faggian. Ludics with repetitions (exponentials, interactive types and completeness). *Logical Methods in Computer Science*, 7, 2011. URL: https://lmcs.episciences.org/1095, doi:10.2168/LMCS-7(2:13)2011.

[Bib13] Wolfgang Bibel. *Automated theorem proving*. Springer Science & Business Media, 2013. doi:10.1007/978-3-322-90102-6.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[Bla92] Andreas Blass. A game semantics for linear logic. *Annals of Pure and Applied logic*, 56(1-3):183–220, 1992. URL: https://www.sciencedirect.com/science/article/pii/0168007292900739, doi:10.1016/0168-0072(92)90073-9.

[Bla98] Deborah L. Black. Logic in Islamic philosophy. *Routledge Encyclopedia of Philosophy*, 5:706–13, 1998. doi:10.4324/9780415249126-H017-1.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1998. doi:10.1017/CBO9781139172752.

[Boc61] Józef Maria Bochenski. *A history of formal logic*. University of Notre Dame Press, 1961. URL: https://circulosemiotico.files.wordpress.com/2012/10/historyofformall00boch.pdf.

[BP84] Paul Benacerraf and Hilary Putnam. *Philosophy of mathematics: Selected readings*. Cambridge University Press, 1984.

[BP99] Patrick Baillot and Marco Pedicini. Elementary complexity and geometry of interaction. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 25–33, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Brü04] Kai Brünnler. *Deep inference and symmetry in classical proofs*. Logos Verlag Berlin, 2004. URL: https://iccl.inf.tu-dresden.de/web/Phdthesis3005/en.

[BT09] Michele Basaldella and Kazushige Terui. On the meaning of logical completeness. In *International Conference on Typed Lambda Calculi and Applications*, pages 50–64. Springer, 2009. URL: https://arxiv.org/abs/1011.1625, doi:10.48550/arXiv.1011.1625.

[Bur00] Stanley Burris. The laws of Boole's thought. *Preprint*, 2000.

[CC23] Simon Castellan and Pierre Clairambault. The geometry of causality: Multi-token geometry of interaction and its causal unfolding. *Proceedings of the ACM on Programming Languages*, 7(POPL):689–717, 2023. URL: https://dl.acm.org/doi/abs/10.1145/3571217, doi:10.1145/3571217.

[CDG+97] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. *Available online since*, 1997. URL: https://inria.hal.science/hal-03367725.

[Chm09] Janusz Chmielewski. *Language and Logic in Ancient China: Collected Papers on the Chinese Language and Logic*. Warsaw: PAN, 2009.

[Chu32] Alonzo Church. A set of postulates for the foundation of logic. *Annals of mathematics*, pages 346–366, 1932. URL: https://www.jstor.org/stable/1968337, doi:10.2307/1968337.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940. doi:10.2307/2266170.

[CI96] Karel Culik II. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160(1-3):245–251, 1996. URL: https://www.sciencedirect.com/science/article/pii/S0012365X96001185, doi:10.1016/S0012-365X(96)00118-5.

[Col58] Henry Thomas Colebrooke. *Essays on the Religion and Philosophy of the Hindus*. Williams and Norgate, 1858.

[Con22] Sidney Congard. La logique face à l'arbitraire. 2022. URL: https://hal.science/hal-03689001/.

[CR96] Alain Colmerauer and Philippe Roussel. The birth of Prolog. In *History of Programming Languages—II*, page 331–367. Association for Computing Machinery, New York, NY, USA, 1996. URL: https://dl.acm.org/doi/10.1145/234286.1057820, doi:10.1145/234286.1057820.

[Cur34] Haskell B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584, 1934. URL: https://www.pnas.org/doi/pdf/10.1073/pnas.20.11.584, doi:10.1073/pnas.20.11.584.

[Cur03] Pierre-Louis Curien. Symmetry and interactivity in programming. *Bulletin of Symbolic Logic*, pages 169–180, 2003. URL: https://www.jstor.org/stable/3094788.

[Cur05a] Pierre-Louis Curien. Introduction to linear logic and ludics, part I. *arXiv preprint cs/0501035*, 2005. URL: https://arxiv.org/abs/cs/0501035, doi:10.48550/arXiv.cs/0501035.

[Cur05b] Pierre-Louis Curien. Introduction to linear logic and ludics, part II. *arXiv preprint cs/0501039*, 2005. URL: https://arxiv.org/abs/cs/0501039, doi:10.48550/arXiv.cs/0501039.

[CVV21] Kostia Chardonnet, Benoît Valiron, and Renaud Vilmart. Geometry of Interaction for ZX-Diagrams. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14470, doi:10.4230/LIPIcs.MFCS.2021.30.

[Dan90] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)*. PhD thesis, Paris 7, 1990. URL: https://perso.ens-lyon.fr/pierre.lescanne/PUBLICATIONS/DanosPhD.pdf.

[Dav58] Martin Davis. Computability and Unsolvability. 1982 ed, 1958. doi:10.2307/2273892.

[DC18] Daniel De Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, 2018. URL: https://arxiv.org/abs/0905.4251, doi:10.48550/arXiv.0905.4251.

[DCK97] Roberto Di Cosmo and Delia Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 35–46. IEEE, 1997. URL: https://ieeexplore.ieee.org/document/614927, doi:10.1109/LICS.1997.614927.

[DCKP03] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409, 2003. doi:10.1017/S0960129502003791.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, sep 2001. doi:10.1145/502807.502810.

[Des16] Joëlle Despeyroux. (mathematical) logic for systems biology. In *Computational Methods in Systems Biology*, 2016. URL: https://www.semanticscholar.org/paper/(Mathematical)-Logic-for-Systems-Biology-(Invited-Despeyroux/c4789e2f981ae3ad82f565cb3964993efccf5fe1, doi:10.1007/978-3-319-45177-0_1.

[DFLO19] Joëlle Despeyroux, Amy Felty, Pietro Liò, and Carlos Olarte. A logical framework for modelling breast cancer progression. In Madalena Chaves and Manuel A. Martins, editors, *Molecular Logic and Computational Synthetic Biology*, pages 121–141, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-19432-1_8.

[Dij01] Edsger W. Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001. doi:10.1145/365559.365617.

[DJS95] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Lkq and LKT: sequent calculi for second order logic based upon dual linear decompositions of classical implication. *Advances in Linear Logic*, 222:211–224, 1995. URL: https://www.semanticscholar.org/paper/LKQ-and-LKT%3A-sequent-calculi-for-second-order-logic-Danos-Joinet/20fb476610426c30608714e984a451cc1108c659.

[DLTY17] Ugo Dal Lago, Ryo Tanaka, and Akira Yoshimizu. The geometry of concurrent interaction: Handling multiple ports by way of multiple tokens. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017. URL: https://www.computer.org/csdl/proceedings-article/lics/2017/08005112/12OmNwE9Ozy, doi:10.1109/LICS.2017.8005112.

[DM60] Augustus De Morgan. *Syllabus of a proposed system of logic*. Walton and Maberly, 1860.

[dMDF+20] Elisabetta de Maria, Joelle Despeyroux, Amy Felty, Pietro Lió, Carlos Olarte, and Abdorrahim Bahrami. Computational logic for biomedicine and neurosciences. *ArXiV*, 2020. URL: https://arxiv.org/abs/2007.07571, doi:10.48550/arXiv.2007.07571.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960. URL: https://dl.acm.org/doi/10.1145/321033.321034, doi:10.1145/321033.321034.

[DR89] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989. doi:10.1007/BF01622878.

[DR95] Vincent Danos and Laurent Regnier. Proof-nets and the hilbert space. In Jean-Yves Girard, Yves Lafont, and LaurentEditors Regnier, editors, *Advances in Linear Logic*, London Mathematical Society Lecture Note Series, page 307–328. Cambridge University Press, 1995. doi:10.1017/CBO9780511629150.016.

[DR99] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227(1-2):79–97, 1999. URL: https://www.sciencedirect.com/science/article/pii/S0304397599000493, doi:10.1016/S0304-3975(99)00049-3.

[Duc09] Etienne Duchesne. *La localisation en logique: géométrie de l'interaction et sémantique dénotationnelle*. PhD thesis, Université Aix-Marseille II, 2009. URL: https://www.theses.fr/2009AIX22080.

[Dum91] Michael Dummett. *The logical basis of metaphysics*. Harvard university press, 1991.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web International Summer School*, pages 40–110. Springer, 2009. doi:10.1007/978-3-642-03754-2_2.

[EMJP19] Joanna Ellis-Monaghan, Nataša Jonoska, and Greta Pangborn. Tile-based DNA nanostructures: mathematical design and problem encoding. *Algebraic and Combinatorial Computational Biology*, pages 35–60, 2019. URL: https://www.sciencedirect.com/science/article/abs/pii/B9780128140666000027, doi:10.1016/B978-0-12-814066-6.00002-7.

[EO91] Norbert Eisinger and Hans Jürgen Ohlbach. Deduction systems based on resolution. 1991. URL: https://uir.unisa.ac.za/handle/10500/24123.

[F+79] Gottlob Frege et al. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. *From Frege to Gödel: A source book in mathematical logic*, 1931:1–82, 1879.

[Fac83] R. Lance Factor. What is the" logic" in buddhist logic? *Philosophy East and West*, 33(2):183–188, 1983. URL: https://www.jstor.org/stable/1399101, doi:10.2307/1399101.

[Fer01] José Ferreirós. The road to modern logic—an interpretation. *Bulletin of Symbolic Logic*, 7(4):441–484, 2001. doi:10.2307/2687794.

[FF03] John Fuegi and Jo Francis. Lovelace & Babbage and the creation of the 1843'notes'. *IEEE Annals of the History of Computing*, 25(4):16–26, 2003. URL: https://ieeexplore.ieee.org/document/1253887, doi:10.1109/MAHC.2003.1253887.

[FPQ21] Christophe Fouqueré, Jean-Jacques Pinto, and Myriam Quatrini. Incoherences in dialogues and their formalization focus on dialogues with schizophrenic individuals. In *(In) coherence of Discourse*, pages 91–115. Springer, 2021. URL: https://sorbonne-paris-nord.hal.science/hal-03660353/, doi:10.1007/978-3-030-71434-5_5.

[FR94] Arnaud Fleury and Christian Retoré. The mix rule. *Mathematical Structures in Computer Science*, 4(2):273–285, 1994. doi:10.1017/S0960129500000451.

[Fre82] Gottlob Frege. Ueber die wiffenfchaftliche Berechtigung einer Begriffsfchrift. *Zeitschrift für philosophie und philosophische Kritik: vormals Fichte-Ulricische Zeitschrift*, 81:48, 1882.

[Fri75] Harvey Friedman. Some systems of second order arithmetic and their use. In *Proceedings of the international congress of mathematicians (Vancouver, BC, 1974)*, volume 1, pages 235–242, 1975.

[Fri76] Harvey M. Friedman. Systems on second order arithmetic with restricted induction I, II. *J. Symb. Logic*, 41:557–559, 1976.

[Fro00] Bertram Fronhöfer. Proof structures and matrix graphs. *Intellectics and Computational Logic: Papers in Honor of Wolfgang Bibel*, pages 159–173, 2000. doi:10.1007/978-94-015-9383-0_10.

[FY19] Yosuke Fukuda and Akira Yoshimizu. A linear-logical reconstruction of intuitionistic modal logic S4. *ArXiV*, 2019. URL: https://arxiv.org/abs/1904.10605, doi:10.48550/arXiv.1904.10605.

[GAL92a] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 15–26, 1992. URL: https://dl.acm.org/doi/abs/10.1145/143165.143172, doi:10.1145/143165.143172.

[GAL92b] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. Linear logic without boxes. In *[1992] Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 223–234, 1992. doi:10.1109/LICS.1992.185535.

[Gan13] Jonardon Ganeri. The philosophy of the Hindus: On the Nyāya and Vaiśesika systems (1824). In *Indian Logic*, pages 34–66. Routledge, 2013.

[Gel08] Michael Gelfond. Answer sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 285–316. Elsevier, 2008. URL: https://www.sciencedirect.com/science/article/pii/S1574652607030076, doi:https://doi.org/10.1016/S1574-6526(07)03007-6.

[Gen35a] Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[Gen35b] Gerhard Gentzen. Untersuchungen über das logische Schließen. II. *Mathematische Zeitschrift*, 39(1):405–431, 1935. doi:10.1007/BF01201363.

[GG07] Alessio Guglielmi and Tom Gundersen. Normalisation control in deep inference via atomic flows. *ArXiV*, 2007. URL: https://arxiv.org/abs/0709.1205, doi:10.48550/arXiv.0709.1205.

[Gim09] Stéphane Gimenez. *Programmer, calculer et raisonner avec les réseaux de la logique linéaire*. PhD thesis, Université Paris-Diderot-Paris VII, 2009. URL: https://theses.hal.science/tel-00629013/.

[Gir87a] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987. URL: https://www.sciencedirect.com/science/article/pii/0304397587900454, doi:10.1016/0304-3975(87)90045-4.

[Gir87b] Jean-Yves Girard. Multiplicatives. In G. Lolli, editor, *Logic and Computer Science: New Trends and Applications*, pages 11–34. Rosenberg & Sellier, 1987.

[Gir88] Jean-Yves Girard. Normal functors, power series and λ-calculus. *Annals of Pure and Applied Logic*, 37(2):129–177, 1988. URL: https://www.sciencedirect.com/science/article/pii/0168007288900255, doi:https://doi.org/10.1016/0168-0072(88)90025-5.

[Gir89a] Jean-Yves Girard. Geometry of interaction I: interpretation of system F. In *Studies in Logic and the Foundations of Mathematics*, volume 127, pages 221–260. Elsevier, 1989. URL: https://www.sciencedirect.com/science/article/abs/pii/S0049237X08702714, doi:10.1016/S0049-237X(08)70271-4.

[Gir89b] Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.

[Gir91] Jean-Yves Girard. Quantifiers in linear logic II. *Nuovi problemi della logica e della filosofia della scienza*, 2:1, 1991.

[Gir95] Jean-Yves Girard. Geometry of interaction III: Accommodating the additives. In *Proceedings of the Workshop on Advances in Linear Logic*, page 329–389, USA, 1995. Cambridge University Press. URL: https://girard.perso.math.cnrs.fr/GOI3.pdf.

[Gir96] Jean-Yves Girard. *Proof-nets: the parallel syntax for proof-theory*, chapter 4, pages 95–123. Routledge, 1996. URL: https://girard.perso.math.cnrs.fr/Proofnets.pdf.

[Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998. URL: https://www.sciencedirect.com/science/article/pii/S0890540198927006, doi:https://doi.org/10.1006/inco.1998.2700.

[Gir99] Jean-Yves Girard. On the meaning of logical rules I: syntax versus semantics. In *Computational logic*, pages 215–272. Springer, 1999. URL: https://girard.perso.math.cnrs.fr/meaning1.pdf, doi:10.1007/978-3-642-58622-4_7.

[Gir00] Jean-Yves Girard. On the meaning of logical rules II: multiplicatives and additives. *NATO ASI Series F Computer and Systems Sciences*, 175:183–212, 2000. URL: https://girard.perso.math.cnrs.fr/meaning2.pdf.

[Gir01] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical structures in computer science*, 11(3):301, 2001. doi:10.1017/S096012950100336X.

[Gir06] Jean-Yves Girard. Geometry of interaction IV: the feedback equation. In Viggo Stoltenberg-Hansen and JoukoEditors Väänänen, editors, *Logic Colloquium '03*, Lecture Notes in Logic, page 76–117. Cambridge University Press, 2006. doi:10.1017/9781316755785.006.

[Gir07] Jean-Yves Girard. Truth, modality and intersubjectivity. *Mathematical structures in computer science*, 17(6):1153–1167, 2007. doi:10.1017/S0960129507006342.

[Gir11a] Jean-Yves Girard. *The Blind Spot: lectures on logic*. European Mathematical Society, 2011.

[Gir11b] Jean-Yves Girard. Geometry of interaction V: logic in the hyperfinite factor. *Theoretical Computer Science*, 412(20):1860–1883, 2011. URL: https://www.sciencedirect.com/science/article/pii/S0304397510007073, doi:10.1016/j.tcs.2010.12.016.

[Gir11c] Jean-Yves Girard. La syntaxe transcendantale, manifeste. 2011. URL: https://girard.perso.math.cnrs.fr/syntran.pdf.

[Gir13a] Jean-Yves Girard. Geometry of interaction VI: a blueprint for transcendental syntax. 2013.

[Gir13b] Jean-Yves Girard. Three lightings of logic (invited talk). In *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013. URL: https://drops.dagstuhl.de/opus/volltexte/2013/4185/, doi:10.4230/LIPIcs.CSL.2013.11.

[Gir16a] Jean-Yves Girard. *Le fantôme de la transparence*. Éditions Allia Paris, 2016.

[Gir16b] Jean-Yves Girard. Transcendental syntax II: non-deterministic case. 2016. URL: https://girard.perso.math.cnrs.fr/trsy2.pdf.

[Gir17] Jean-Yves Girard. Transcendental syntax I: deterministic case. *Mathematical Structures in Computer Science*, 27(5):827–849, 2017. URL: https://girard.perso.math.cnrs.fr/trsy1.pdf, doi:10.1017/S0960129515000407.

[Gir18a] Jean-Yves Girard. La logique 2.0. 2018. URL: https://girard.perso.math.cnrs.fr/logique2.0.pdf.

[Gir18b] Jean-Yves Girard. Transcendental syntax III: equality. 2018. URL: https://girard.perso.math.cnrs.fr/trsy3.pdf.

[Gir19] Jean-Yves Girard. Un tract anti-système. 2019. URL: https://girard.perso.math.cnrs.fr/systeme.pdf.

[Gir20a] Jean-Yves Girard. *Transcendental Syntax IV: Logic Without Systems*, pages 17–36. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-62077-6_2.

[Gir20b] Jean-Yves Girard. Un tract anti-système II: le monstre de Gila. 2020. URL: https://girard.perso.math.cnrs.fr/gila.pdf.

[Göd31] Kurt Gödel. Über formal unentscheidbare sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931. doi:10.1007/BF01700692.

[Gri89] Timothy G. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58, 1989. URL: https://dl.acm.org/doi/pdf/10.1145/96709.96714, doi:10.1145/96709.96714.

[Gue04] Stefano Guerrini. Proof nets and the lambda-calculus. *Linear logic in computer science*, pages 65–118, 2004. doi:10.1017/CBO9780511550850.003.

[Gug] Alessio Guglielmi. Deep inference and the calculus of structures. URL: https://people.bath.ac.uk/ag248/p/CalcStrPR.pdf.

[Gur12] Yuri Gurevich. What is an algorithm? In *SOFSEM 2012: Theory and Practice of Computer Science: 38th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlyn, Czech Republic, January 21-27, 2012. Proceedings 38*, pages 31–42. Springer, 2012. doi:10.1007/978-3-642-27660-6_3.

[GW04] Dov M. Gabbay and John Hayden Woods. *Handbook of the History of Logic*, volume 2009. Elsevier North-Holland, 2004.

[Ham21] Wendy Hammache. *Contrôle du calcul et limites du sens: fonctions, computation et types de G. Frege à A. Church*. PhD thesis, Université Lyon III, 2021. URL: https://scd-resnum.univ-lyon3.fr/out/theses/2021_out_hammache_w.pdf.

[Har09] John Harrison. *Handbook of practical logic and automated reasoning*. Cambridge University Press, 2009. URL: 10.1017/CBO9780511576430.

[Hed04] Shawn Hedman. *A First Course in Logic: An introduction to model theory, proof theory, computability, and complexity*, volume 1. OUP Oxford, 2004. doi:10.1093/oso/9780198529804.001.0001.

[Her30] Jacques Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, Université de Paris, 1930. URL: https://eudml.org/doc/192791.

[HH16] Dominic Hughes and Willem Heijltjes. Conflict nets: Efficient locally canonical MALL proof nets. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, 2016. URL: https://ieeexplore.ieee.org/document/8576483.

[HM89] Jochen Hager and Martin Moser. An approach to parallel unification using transputers. In *GWAI-89 13th German Workshop on Artificial Intelligence: Eringerfeld, 18.–22. September 1989*, pages 83–91. Springer, 1989. URL: 10.1007/978-3-642-75100-4_10.

[HMOS08] Matthew Hague, Andrzej S. Murawski, C. H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE, 2008. URL: https://ieeexplore.ieee.org/document/4557934, doi:10.1109/LICS.2008.34.

[Hor51] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951. URL: https://www.jstor.org/stable/2268661.

[How80] William A. Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[HS97] Jaakko Hintikka and Gabriel Sandu. Game-theoretical semantics. In *Handbook of logic and language*, pages 361–410. Elsevier, 1997. URL: https://www.sciencedirect.com/science/article/abs/pii/B9780444817143500096, doi:10.1016/B978-044481714-3/50009-6.

[HS03] Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. *Theoretical computer science*, 294(1-2):183–231, 2003. URL: https://www.sciencedirect.com/science/article/pii/S0304397501002419, doi:10.1016/S0304-3975(01)00241-9.

[HS06] Esfandiar Haghverdi and Philip Scott. A categorical model for the geometry of interaction. *Theoretical Computer Science*, 350(2-3):252–274, 2006. URL: https://www.sciencedirect.com/science/article/pii/S0304397505006808, doi:10.1016/j.tcs.2005.10.028.

[HVG03] D. Hughes and R. Van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic (extended abstract). In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, pages 1–10,

2003. URL: https://ieeexplore.ieee.org/document/1210039, doi:10.1109/LICS.2003.1210039.

[iln90] Robin ilner. *Functions as processes*, pages 167–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990. doi:10.1007/BFb0032030.

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1):86 – 104, 1986. doi:https://doi.org/10.1016/S0019-9958(86)80029-8.

[Imm12] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012. doi:10.1007/978-1-4612-0539-5.

[Jae96] Peter Jaenecke. Elementary principles for representing knowledge. *KO KNOWLEDGE ORGANIZATION*, 23(2):88–102, 1996. URL: https://kr.org/proceedings/KR-1991-proceedings-scanned.pdf.

[JM05] Nataša Jonoska and Gregory L. McColm. A computational model for self-assembling flexible tiles. In *International Conference on Unconventional Computation*, pages 142–156. Springer, 2005. doi:10.1007/11560319_14.

[JM06] Nataša Jonoska and Gregory L. McColm. Flexible versus rigid tile assembly. In *International Conference on Unconventional Computation*, pages 139–151. Springer, 2006. URL: https://link.springer.com/chapter/10.1007/11839132_12, doi:10.1007/11839132_12.

[JMS11] Natasha Jonoska, Gregory L. McColm, and Ana Staninska. On stoichiometry for the assembly of flexible tile DNA complexes. *Natural Computing*, 10(3):1121–1141, 2011. doi:10.1007/s11047-009-9169-1.

[Joi93] Jean-Baptiste Joinet. *Etude de la normalisation du calcul des séquents classique à travers la logique linéaire*. PhD thesis, Paris 7, 1993. URL: https://www.theses.fr/1993PA077066.

[JR15] Emmanuel Jeandel and Michael Rao. An aperiodic set of 11 Wang tiles. *ArXiV*, 2015. URL: https://arxiv.org/abs/1506.06492, doi:10.48550/arXiv.1506.06492.

[JS21] Jean-Baptiste Joinet and Thomas Seiller. From abstraction and indiscernibility to classification and types: revisiting Hermann Weyl's theory of ideal elements. *Kagaku tetsugaku*, 53(2):65–93, 2021. URL: https://hal.science/hal-03128018v1/document.

[KCHM09] Jean-Louis Krivine, Pierre-Louis Curien, Hugo Herbelin, and Paul-André Melliès. Interactive models of computation and program behavior. *Panorama et Synthèses*, 27, 2009.

[Kit91] Hiroaki Kitano. Unification algorithms for massively parallel computers. In *Proceedings of the Second International Workshop on Parsing Technologies*, pages 172–181. IEEE, 1991. URL: https://ieeexplore.ieee.org/document/77189, doi:10.1109/PARBSE.1990.77189.

[KK71] Robert Kowalski and Donald Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3-4):227–260, 1971. URL: https://www.sciencedirect.com/science/article/abs/pii/0004370271900129, doi:10.1016/0004-3702(71)90012-9.

[KKPS82] Norman Kretzmann, Anthony Kenny, Jan Pinborg, and Eleonore Stump. *The Cambridge history of later medieval philosophy: from the rediscovery of Aristotle to the disintegration of scholasticism, 1100-1600*. Cambridge University Press, 1982. doi:10.1017/CHOL9780521226059.

[Knu99] Simo Knuuttila. Medieval theories of modality. 1999. URL: https://plato.stanford.edu/entries/modality-medieval/.

[Koe01] Teun Koetsier. On the prehistory of programmable machines: musical automata, looms, calculators. *Mechanism and Machine theory*, 36(5):589–603, 2001.

[Kow74] Robert Kowalski. Predicate logic as programming language. In *IFIP congress*, volume 74, pages 569–544, 1974. URL: https://www-public.imtbs-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/Kowalski74.pdf.

[Kow75] Robert Kowalski. A proof procedure using connection graphs. *J. ACM*, 22(4):572–595, oct 1975. URL: https://dl.acm.org/doi/10.1145/321906.321919, doi:10.1145/321906.321919.

[Laf89] Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–108. ACM, 1989. URL: https://dl.acm.org/doi/10.1145/96709.96718, doi:10.1145/96709.96718.

[Laf95] Yves Lafont. *From proof nets to interaction nets*, page 225–248. London Mathematical Society Lecture Note Series. Cambridge University Press, 1995. doi:10.1017/CBO9780511629150.012.

[Laf97] Yves Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997. URL: https://www.sciencedirect.com/science/article/pii/S0890540197926432, doi:10.1006/inco.1997.2643.

[Laf99] Yves Lafont. Linear logic pages. 1999. URL: http://iml.univ-mrs.fr/~lafont/pub/llpages.pdf.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theoretical computer science*, 318(1-2):163–180, 2004. URL: https://www.sciencedirect.com/science/article/pii/S0304397503005231, doi:10.1016/j.tcs.2003.10.018.

[Lam89] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 16–30, 1989. URL: https://dl.acm.org/doi/pdf/10.1145/96709.96711, doi:10.1145/96709.96711.

[Lau99] Olivier Laurent. Polarized proof-nets: proof-nets for LC. In *International Conference on Typed Lambda Calculi and Applications*, pages 213–227. Springer, 1999. doi:10.1007/3-540-48959-2_16.

[Lau01] Olivier Laurent. A token machine for full geometry of interaction. In *International Conference on Typed Lambda Calculi and Applications*, pages 283–297. Springer, 2001. doi:10.1007/3-540-45413-6_23.

[Lau03] Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, 2003. URL: https://www.sciencedirect.com/science/article/pii/S0304397501002973, doi:10.1016/S0304-3975(01)00297-3.

[Lec11] Alain Lecomte. *Meaning, logic and ludics*. World Scientific, 2011. doi:10.1142/p670.

[Lei12] Alexander Leitsch. *The resolution calculus*. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-60605-2.

[Lév78] Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université Paris VII, 1978.

[LJBA01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–92, 2001. doi:10.1145/360204.360210.

[LLS09] James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410(4-5):384–405, 2009. URL: https://www.sciencedirect.com/science/article/pii/S030439750800724X, doi:10.1016/j.tcs.2008.09.062.

[LM08] Olivier Laurent and Roberto Maieli. Cut elimination for monomial MALL proof nets. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 486–497. IEEE, 2008. URL: https://ieeexplore.ieee.org/document/4557937, doi:10.1109/LICS.2008.31.

[LMM88] J-L. Lassez, Michael J. Maher, and Kim Marriott. Unification revisited. In *Foundations of logic and functional programming*, pages 67–113. Springer, 1988. doi:10.1007/3-540-19129-1_4.

[LP11] Giuseppe Longo and Thierry Paul. The mathematics of computing between logic and physics. In *Computability in Context: Computation and Logic in the Real World*, pages 243–273. World Scientific, 2011. doi:10.1142/9781848162778_0007.

[LQ09] Alain Lecomte and Myriam Quatrini. Ludics and its applications to natural language semantics. In *International Workshop on Logic, Language, Information, and Computation*, pages 242–255. Springer, 2009. doi:10.1007/978-3-642-02261-6_20.

[LR03] Olivier Laurent and Laurent Regnier. About translations of classical logic into polarized linear logic. In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, pages 11–20. IEEE, 2003. URL: https://ieeexplore.ieee.org/document/1210040, doi:10.1109/LICS.2003.1210040.

[LRM91] Jorge Lobo, Arcot Rajasekar, and Jack Minker. Semantics of horn and disjunctive logic programs. *Theoretical Computer Science*, 86(1):93–106, 1991. URL: https://www.sciencedirect.com/science/article/pii/030439759190006N, doi:10.1016/0304-3975(91)90006-N.

[LS10] Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *European Symposium on Programming*, pages 205–225. Springer, 2010. doi:10.1007/978-3-642-11957-6_12.

[Mar86] Christopher J. Martin. William's machine. *The Journal of Philosophy*, 83(10):564–572, 1986. URL: http://www.jstor.org/stable/2026432, doi:10.2307/2026432.

[Maz17] Damiano Mazza. *Polyadic Approximations in Logic and Computation (Habilitation thesis)*. PhD thesis, Université Paris 13, 2017. URL: https://www.lipn.fr/~mazza/papers/Habilitation.pdf.

[McL08] Colin McLarty. Theology and its discontents: the origin myth of modern mathematics. *wersja April*, 15:2008, 2008.

[Mer15] Lucius Gregory Meredith. Linear types can change the blockchain. *arXiv preprint arXiv:1506.01001*, 2015. URL: https://arxiv.org/abs/1506.01001, doi:10.48550/arXiv.1506.01001.

[Mil95] Dale Miller. A survey of linear logic programming. *Computational Logic: The Newsletter of the European Network of Excellence in Computational Logic*, 2(2):63–67, 1995. URL: https://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/ComputNet95/llsurvey.html.

[Mil21] Dale Miller. A survey of the proof-theoretic foundations of logic programming. *Theory and Practice of Logic Programming*, pages 1–46, 2021. URL: https://arxiv.org/abs/2109.01483, doi:10.1017/S1471068421000533.

[Min94] Jack Minker. Overview of disjunctive logic programming. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):1–24, 1994. doi:10.1007/BF01530759.

[Miq09] Alexandre Miquel. De la formalisation des preuves à l'extraction de programmes. *HdR thesis, Université Paris*, 7, 2009.

[Miq17] Étienne Miquey. *Classical realizability and side-effects*. PhD thesis, Université Sorbonne Paris Cité-Université Paris Diderot (Paris 7), 2017. URL: https://www.i2m.univ-amu.fr/perso/etienne.miquey/these/these.pdf.

[MLS84] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*, volume 9. Bibliopolis Naples, 1984.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982. URL: https://dl.acm.org/doi/10.1145/357162.357169, doi:10.1145/357162.357169.

[Moo97] Gregory H. Moore. The prehistory of infinitary logic: 1885–1955. In *Structures and Norms in Science*, pages 105–123. Springer, 1997. doi:10.1007/978-94-017-0538-7_7.

[MOTW99] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theoretical Computer Science*, 228(1-2):175–210, 1999. URL: https://www.sciencedirect.com/science/article/pii/S1571066104000222, doi:10.1016/S1571-0661(04)00022-2.

[Mou88] Guy Mourlevat. *Les machines arithmétiques de Blaise Pascal*, volume 51. Académie des sciences, lettres, arts, 1988.

[MP05] Roberto Maieli and Quintijn Puite. Modularity of proof-nets. *Archive for Mathematical Logic*, 44(2):167–193, 2005. doi:10.1007/s00153-004-0242-2.

[MT03] Harry G. Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. In *Italian Conference on Theoretical Computer Science*, pages 23–36. Springer, 2003. doi:10.1007/978-3-540-45208-9_4.

[MT15] Damiano Mazza and Kazushige Terui. Parsimonious types and non-uniform computation. In *International Colloquium on Automata, Languages, and Programming*, pages 350–361. Springer, 2015. doi:10.1007/978-3-662-47666-6_28.

[MW17] Pierre-Étienne Meunier and Damien Woods. The non-cooperative tile assembly model is not intrinsically universal or capable of bounded turing machine simulation. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 328–341, 2017. URL: https://dl.acm.org/doi/10.1145/3055399.3055446, doi:10.1145/3055399.3055446.

[NGSKDS07] Manh Thang Nguyen, Jürgen Giesl, Peter Schneider-Kamp, and Danny De Schreye. Termination analysis of logic programs based on dependency graphs. In *International Symposium on Logic-based Program Synthesis and Transformation*, pages 8–22. Springer, 2007. doi:10.1007/978-3-540-78769-3_2.

[Ngu21] Lê Thành Dũng Nguyễn. *Towards a resource based approximation theory of programs*. PhD thesis, Université Sorbonne Paris Nord, 2021. URL: https://theses.hal.science/tel-04132636v1/document.

[NS19] Lê Thành Dũng Nguyen and Thomas Seiller. Coherent interaction graphs. *ArXiV*, 2019. URL: https://arxiv.org/abs/1904.06849, doi:10.48550/arXiv.1904.06849.

[Par92] Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 190–201. Springer, 1992. doi:10.1007/BFb0013061.

[Pas97] Blaise Pascal. Machine d'arithmétique. *Modern Logic*, 7(1):56–66, 1997.

[Pat14] Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014. doi:10.1007/s11047-013-9379-4.

[Pei79] Charles S. Peirce. On junctures and fractures in logic. *Writings of Charles S. Peirce*, 1884:391, 1879. URL: https://www.jstor.org/stable/j.ctt16gz8j1.

[Pen92] Mati Pentus. Equivalent types in Lambek calculus and linear logic. *Mian Prepublication Series*, 1992. URL: https://www.semanticscholar.org/paper/Equivalent-Types-in-Lambek-Calculus-and-Linear-Pentus-Series/c8e6b6789cad6675bc73ac0a98c1014814feb90c.

[Pet08] Charles Petzold. *The annotated Turing: a guided tour through Alan Turing's historic paper on computability and the Turing machine*. Wiley Publishing, 2008.

[Pis15] Paolo Pistone. Rule-following and the limits of formalization: Wittgenstein's considerations through the lens of logic. In *From Logic to Practice*, pages 91–110. Springer, 2015. doi:10.1007/978-3-319-10434-8_6.

[PL09] Thierry Paul and Giuseppe Longo. Le monde et le calcul: réflexions sur calculabilité, mathématiques et physique. In *Logique & Interaction: Géométrie de la cognition, Actes du colloque et école thématique du CNRS" Logique, Sciences, Philosophie" a Cerisy, Hermann*, 2009.

[Pos36] Emil L. Post. Finite combinatory processes—formulation 1. *The journal of symbolic logic*, 1(3):103–105, 1936. doi:https://doi.org/10.2307/2269031.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946. URL: https://www.ams.org/journals/bull/1946-52-04/S0002-9904-1946-08555-9/S0002-9904-1946-08555-9.pdf, doi:10.1090/S0002-9904-1946-08555-9.

[R+65] John Alan Robinson et al. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965. URL: https://dl.acm.org/doi/10.1145/321250.321253, doi:10.1145/321250.321253.

[Rad62] Tibor Rado. On non-computable functions. *Bell System Technical Journal*, 41(3):877–884, 1962. doi:10.1002/j.1538-7305.1962.tb00480.x.

[Rea02] Stephen Read. Medieval theories: properties of terms. 2002. URL: https://plato.stanford.edu/entries/medieval-terms/.

[Reg92] Laurent Regnier. *Lambda-calcul et réseaux*. PhD thesis, Université Paris 7, 1992. URL: https://www.theses.fr/1992PA077165.

[Reg94] Laurent Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126(2):281–292, 1994. URL: https://www.sciencedirect.com/science/article/pii/0304397594900124, doi:10.1016/0304-3975(94)90012-4.

[Res64] Nicholas Rescher. Studies in the history of arabic logic. *Revue Philosophique de Louvain*, 76:669–670, 1964. doi:10.2307/2217894.

[Res02] Greg Restall. *An introduction to substructural logics*. Routledge, 2002.

[Res12] Nicholas Rescher. *Temporal modalities in Arabic logic*, volume 2. Springer Science & Business Media, 2012. URL: https://link.springer.com/book/10.1007/978-94-010-3523-1, doi:10.1007/978-94-010-3523-1.

[Rib07] Colin Riba. Strong normalization as safe interaction. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 13–22. IEEE, 2007. URL: https://www.computer.org/csdl/proceedings-article/lics/2007/29080013/12OmNxWui8x, doi:10.1109/LICS.2007.46.

[Rie14] Lionel Rieg. *On forcing and classical realizability*. PhD thesis, Ecole normale supérieure de lyon-ENS LYON, 2014. URL: https://theses.hal.science/tel-01061442/PDF/RIEG_Lionel_2014_These.pdf.

[Ros96] Kristoffer H. Rose. *Explicit substitution: tutorial & survey*. Computer Science Department, 1996. doi:https://www.researchgate.net/publication/228386201_Explicit_substitution_tutorial_survey.

[Rus03] Bertrand Russell. *The principles of mathematics*. Public Domain, 1903. URL: https://people.umass.edu/klement/pom/.

[Sch31] Arthur Schopenhauer. *Eristische Dialektik: Die Kunst, Recht zu Behalten*. Kein & Aber Verlag, 1831.

[Sch24] Moses Schönfinkel. Über die bausteine der mathematischen Logik. *Mathematische annalen*, 92(3):305–316, 1924. doi:10.1007/BF01448013.

[Sch56] Kurt Schütte. Ein system des verknüpfenden schliessens. *Archiv für mathematische Logik und Grundlagenforschung*, 2(2):55–67, 1956. doi:10.1007/BF01969991.

[Sch65] Thomas W. Scharle. Axiomatization of propositional calculus with Sheffer functors. *Notre Dame Journal of Formal Logic*, 6(3):209–217, 1965. URL: https://projecteuclid.org/journals/notre-dame-journal-of-formal-logic/volume-6/issue-3/Axiomatization-of-propositional-calculus-with-Sheffer-functors/10.1305/ndjfl/1093958259.full, doi:10.1305/ndjfl/1093958259.

[Sch94] Harold Schellinx. *The noble art of linear decorating*. University of Amsterdam, 1994.

[Sch06] Ulrich Schöpp. Space-efficient computation by interaction. In *International Workshop on Computer Science Logic*, pages 606–621. Springer, 2006. doi:10.1007/11874683_40.

[Sco82] Dana Scott. Domains for denotational semantics. In *International Colloquium on Automata, Languages, and Programming*, pages 577–610. Springer, 1982. URL: https://www.researchgate.net/publication/220897586_Domains_for_Denotational_Semantics, doi:10.1007/BFb0012801.

[See82] Nadrian C. Seeman. Nucleic acid junctions and lattices. *Journal of theoretical biology*, 99(2):237–247, 1982. URL: https://www.sciencedirect.com/science/article/abs/pii/0022519382900029, doi:10.1016/0022-5193(82)90002-9.

[Sei12a] Thomas Seiller. Interaction graphs: multiplicatives. *Annals of Pure and Applied Logic*, 163(12):1808–1837, 2012. URL: https://www.sciencedirect.com/science/article/pii/S0168007212000759, doi:10.1016/j.apal.2012.04.005.

[Sei12b] Thomas Seiller. *Logique dans le facteur hyperfini: géometrie de l'interaction et complexité*. PhD thesis, Aix-Marseille Université, 2012. URL: https://theses.hal.science/tel-00768403/.

[Sei16a] Thomas Seiller. Interaction graphs: additives. *Annals of Pure and Applied Logic*, 167(2):95–154, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0168007215000998, doi:10.1016/j.apal.2015.10.001.

[Sei16b] Thomas Seiller. Interaction graphs: Full linear logic. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. IEEE, 2016. URL: https://dl.acm.org/doi/10.1145/2933575.2934568, doi:10.1145/2933575.2934568.

[Sei17] Thomas Seiller. Interaction graphs: Graphings. *Annals of Pure and Applied Logic*, 168(2):278–320, 2017. URL: https://www.sciencedirect.com/science/article/pii/S0168007216301300, doi:10.1016/j.apal.2016.10.007.

[Sei18] Thomas Seiller. Interaction graphs: Non-deterministic automata. *ACM Transactions on Computational Logic (TOCL)*, 19(3):1–24, 2018. doi:10.1145/3226594.

[Sei20a] Thomas Seiller. Probabilistic complexity classes through semantics. *ArXiV*, 2020. URL: https://arxiv.org/abs/2002.00009, doi:10.48550/arXiv.2002.00009.

[Sei20b] Thomas Seiller. Zeta functions and the (linear) logic of Markov processes. *ArXiV*, 2020. URL: https://arxiv.org/abs/2001.11906, doi:10.48550/arXiv.2001.11906.

[Sha38] Claude E. Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, 1938. doi:10.1109/T-AIEE.1938.5057767.

[Sha84] Ehud Y. Shapiro. Alternation and the computational complexity of logic programs. *The Journal of Logic Programming*, 1(1):19–33, 1984. URL: https://www.sciencedirect.com/science/article/pii/0743106684900219, doi:10.1016/0743-1066(84)90021-9.

[Shi03] Masaru Shirahata. Geometry of interaction explained (algebra, logic and geometry in informatics). 数理解析研究所講究録, 1318:160–187, 2003. URL: https://www.kurims.kyoto-u.ac.jp/~hassei/algi-13/kokyuroku/19_shirahata.pdf.

[Sib05] Fadi N. Sibai. Parallel unification: Theory and implementations. In *Parallelization in Inference Systems: International Workshop Dagstuhl Castle, Germany, December 17–18, 1990 Proceedings*, pages 51–81. Springer, 2005. doi:https://doi.org/10.1007/3-540-55425-4_3.

[Sic76] Sharon Sickel. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers*, pages 823–835, 1976. URL: https://ieeexplore.ieee.org/document/1674701, doi:10.1109/TC.1976.1674701.

[Sim88] Stephen G. Simpson. Partial realizations of Hilbert's program. *The Journal of Symbolic Logic*, 53(2):349–363, 1988. doi:10.2307/2274508.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation (second edition)*. Thomsom, 2006.

[Smi13] Peter Smith. *An introduction to Gödel's theorems*. Cambridge University Press, 2013. URL: https://www.logicmatters.net/resources/pdfs/godelbook/GodelBookLM.pdf, doi:10.1017/CBO9781139149105.

[Spa02] Paul Vincent Spade. Thoughts, words and things: An introduction to late mediaeval logic and semantic theory. *Copyright by Paul Vincent Spade*, 10:2009, 2002. URL: http://pvspade.com/Logic/docs/thoughts1_1a.pdf.

[Ste61] J. T. Stevenson. Roundabout the runabout inference-ticket. *Analysis*, 21(6):124–128, 1961. URL: http://www.jstor.org/stable/3326421.

[Str06] Lutz Straßburger. Proof nets and the identity of proofs. *ArXiV*, 2006. URL: https://arxiv.org/abs/cs/0610123, doi:10.48550/arXiv.cs/0610123.

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Elsevier, 2006.

[Tai81] William W. Tait. Finitism. *The Journal of Philosophy*, 78(9):524–546, 1981. doi:10.2307/2026089.

[Tär77] Sten-Åke Tärnlund. Horn clause computability. *BIT Numerical Mathematics*, 17(2):215–226, 1977. doi:10.1007/BF01932293.

[Ten79] Neil Tennant. La barre de Scheffer dans la logique des séquents et des syllogismes. *Logique et Analyse*, 22(88):505–514, 1979. URL: http://www.jstor.org/stable/44085164.

[Ter04] Kazushige Terui. Proof nets and boolean circuits. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, pages 182–191. IEEE, 2004. URL: https://ieeexplore.ieee.org/document/1319612, doi:10.1109/LICS.2004.1319612.

[Ter11] Kazushige Terui. Computational ludics. *Theoretical Computer Science*, 412(20):2048–2071, 2011. URL: https://www.sciencedirect.com/science/article/pii/S0304397510007176, doi:10.1016/j.tcs.2010.12.026.

[TGBB95] Alan Mathison Turing, Jean-Yves Girard, Julien Basch, and Patrice Blanchard. *La machine de Turing*. Editions du seuil, 1995.

[Tho91] Wolfgang Thomas. On logics, tilings, and automata. In *International Colloquium on Automata, Languages, and Programming*, pages 441–454. Springer, 1991. doi:10.1007/3-540-54233-7_154.

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936. doi:10.1112/plms/s2-42.1.230.

[Vää19] Jouko Väänänen. Second-order and higher-order logic. 2019. URL: https://plato.stanford.edu/entries/logic-higher-order/.

[VS84] Jeffrey Scott Vitter and Roger A. Simons. Parallel algorithms for unification and other complete problems in P. In *Proceedings of the 1984 Annual Conference of the ACM on The Fifth Generation Challenge*, ACM '84, page 75–84, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800171.809607.

[Wad90] Philip Wadler. Linear types can change the world! In *Programming concepts and methods*, volume 3, page 5, 1990. URL: https://www.semanticscholar.org/paper/Linear-Types-can-Change-the-World!-Wadler/24c850390fba27fc6f3241cb34ce7bc6f3765627.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. *SIGPLAN Not.*, 38(9):189–201, aug 2003. doi:10.1145/944746.944723.

[Wad12] Philip Wadler. Propositions as sessions. *ACM SIGPLAN Notices*, 47(9):273–286, 2012. doi:10.1145/2398856.2364568.

[Wan57] Hao Wang. A variant to Turing's theory of computing machines. *Journal of the ACM (JACM)*, 4(1):63–92, 1957. URL: https://dl.acm.org/doi/pdf/10.1145/320856.320867, doi:10.1145/320856.320867.

[WGRO93] Anna Wierzbicka, Richard A. Geiger, and Brygida Rudzka-Ostyn. The alphabet of human thoughts. *Conceptualizations and Mental Processing in Language*, 3:23, 1993. doi:10.1515/9783110857108.23.

[Win98] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998. URL: https://thesis.library.caltech.edu/1866/.

[Wit10] Ludwig Wittgenstein. *Philosophical investigations*. John Wiley & Sons, 2010.

[Woo15] Damien Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373(2046):20140214, 2015. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.2014.0214, doi:10.1098/rsta.2014.0214.