



HAL
open science

Multiplicative linear logic from a resolution-based tile system

Boris Eng, Thomas Seiller

► **To cite this version:**

Boris Eng, Thomas Seiller. Multiplicative linear logic from a resolution-based tile system. 2022. hal-03725336

HAL Id: hal-03725336

<https://hal.science/hal-03725336v1>

Preprint submitted on 16 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MULTIPLICATIVE LINEAR LOGIC FROM A RESOLUTION-BASED TILE SYSTEM

BORIS ENG AND THOMAS SEILLER

LIPN - Université Sorbonne Paris Nord
e-mail address: engboris@hotmail.fr

CNRS / LIPN - Université Sorbonne Paris Nord
e-mail address: seiller@lipn.fr

ABSTRACT. We present the stellar resolution, a “flexible” tile system based on Robinson’s first-order resolution. After establishing formal definitions and basic properties of the stellar resolution, we show its Turing-completeness and to illustrate the model, we exhibit how it naturally represents computation with Horn clauses and automata as well as non-deterministic tiling constructions used in DNA computing. In the second and main part, by using the stellar resolution, we formalise and extend ideas of a new alternative to proof-net theory sketched by Girard in his transcendental syntax programme. In particular, we encode both cut-elimination and logical correctness for the multiplicative fragment of linear logic (MLL). We finally obtain completeness results for both MLL and MLL extended with the so-called MIX rule. By extending the ideas of Girard’s geometry of interaction, this suggests a first step towards a new understanding of the interplay between logic and computation where linear logic is seen as a (constructed) way to format computation.

1. INTRODUCTION

The evolution of the notion of proof. Among the different materialisations of logic, some remarkable and standard formalisms are Gentzen’s natural deduction and sequent calculus [39, 40] which are attempts at formally representing mathematical reasoning by means of logical rules one applies successively to construct proofs. These rules are represented by means of *sequents* which are expressions $\Gamma \vdash A$ stating that the conclusion A follows from a set of hypotheses Γ (Figure 1). Although intuitive and natural, logical rules seem rather arbitrary and for that reason, attempts at justifying them were made in philosophy [34, Chapter 8].

In the end of the 20th century, the so-called Curry-Howard correspondence was first discovered by Curry [23] but then clearly stated by Howard [62], during the rise of computer science. It shows a formal correspondence between proofs and programs but also between

Key words and phrases: Transcendental Syntax, Geometry of Interaction, proof-nets, linear logic, proof theory, model of computation, resolution, unification, realisability.

The authors would like to thank Valentin Maestracci for his useful comments.

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \wedge\text{L} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge\text{R}$$

FIGURE 1. Two inference rules for the conjunction in sequent calculus. The first one states that we can combine two hypotheses A and B to build the hypothesis $A \wedge B$ and the second one that proving A and B gives a proof of $A \wedge B$.

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \Rightarrow B}{\Gamma, \Delta \vdash B} \text{MD} \qquad \frac{\Gamma \vdash a : A \quad \Delta \vdash f : A \rightarrow B}{\Gamma, \Delta \vdash f(a) : B} \text{app}$$

FIGURE 2. Inference rule for modus ponens and the typing of function application where Γ is seen as a typing environment. The upper part corresponds to premises and the bottom to the conclusion. Giving an argument a of type A to a function f turning an element of type A to an element of type B indeed produces an element $f(a)$ of type B .

formulas and types in programming for some logical systems and some (functional) typed programming systems. A common illustration is the correspondence between natural deduction restricted to the implication (also called *implicative minimal logic*) and the simply typed λ -calculus [20]. The mysterious rules of logic were then given a computational meaning. For instance, the inference rule of *modus ponens* corresponds to the typing of function application (*cf.* Figure 2).

Although mathematical proofs are naturally thought to be purely static objects, this correspondence between proofs and programs shows that they also have a computational or dynamic aspect. The *cut rule* of sequent calculus, defined as follows¹:

$$\frac{\Gamma \vdash A \quad \Delta, A \vdash C}{\Gamma, \Delta \vdash C} \text{cut}$$

represents the use of intermediate lemma in a proof. The expression $\Gamma \vdash A$ states that if a statement A is provable from the hypotheses in Γ and that A together with some hypotheses Δ lead to some conclusion C then we can have a “shortcut” stating that C is a consequence of both Γ and Δ . Although essential in mathematical practice, Gentzen shows that it can, in fact, be removed without any loss of meaning. Similarly to how we can *inline* the code of function calls in the main body of a program, Gentzen’s cut-elimination theorem shows that there exists a procedure of explicitation of proofs which inlines lemmas within proofs. By the Curry-Howard correspondence, this corresponds to a logical counterpart of program execution: proofs are dynamic entities.

The implicit operations in reasoning and their structure. Inspired by the semantics of λ -calculus [43] and its role in the Curry-Howard correspondence, linear logic [41] was introduced by Girard as a refinement of intuitionistic logic (the underlying logic of Gentzen’s natural deduction). Linear logic can be simply presented as a sequent calculus with a control and explicitation over the duplication and erasure of formulas, making formulas handled as sort of limited resources. In particular, the famous decomposition of the implication presents implication are a composition of two operations: $A \Rightarrow B$ becomes $!A \multimap B$ where

¹Notice that the cut rule is very close to modus ponens.

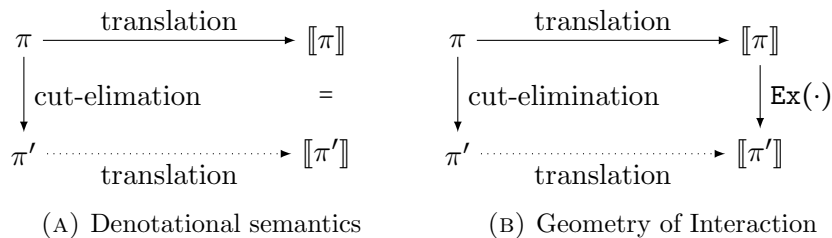


FIGURE 3. We associate a proof π to a mathematical object $\llbracket \pi \rrbracket$. In denotational semantics, we identify a proof π and its cut-elimination π' because we consider they have the same meaning, whereas in the GoI, they differ but are linked by computation. In particular, we are interested in simulating the computation linking them.

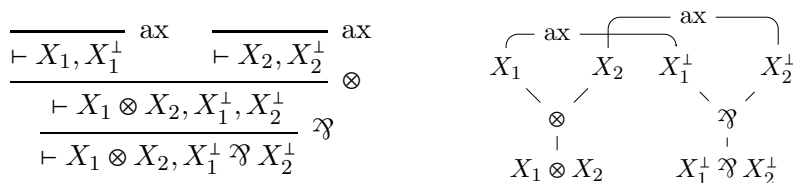


FIGURE 4. Example of sequent calculus proof and its corresponding proof-net in linear logic. Notice that the order of rules is forgotten and that we have a sort of “parallel” [47] syntax for proofs.

!A allows for an arbitrary use of A and \multimap is a linear implication using its premise exactly once. On top of that, linear logic enjoys a nice involutive linear negation breaking the separation between hypothesis and conclusion. This allows a nice compact sequent calculus which will be presented in 4.

Apart from defining a sequent calculus for linear logic, Girard was led to introduce an alternative syntax akin to natural deduction for linear logic: *proof-nets* (Figure 4), which exhibit a non-sequential structure in proofs [47]. Initially seen as a mere syntactic convenience, it led to a new understanding of proof theory with a fine-grained analysis of the mathematical meaning of proofs and of how they relate to computation. Danos [25, Chapter 9-11] and Regnier’s [89] thesis, but also other developments of linear logic that came after [32, 5] illustrate this analysis. Proof-nets are defined from more general alogical graphs called *proof-structures* which constitute a model of computation by themselves (where the graphs are reduced with cut-elimination). It is then possible to assert whether a proof-structure corresponds to a proof-net, *i.e.* is the translation of a sequent calculus proof, by using what we call a *correctness criterion* (*cf.* Section 4.3).

A semantic-free space for logic. Geometry of Interaction² (GoI) [45, 44, 42, 46, 49] was originally introduced as a mathematical analysis of cut-elimination for proof-nets. This gave rise to a dynamic semantics which became a major inspiration behind game semantics [17, 3, 64, 4, 2, 58], distinguishing itself from denotational semantics which usually identify

²Although the expression “Geometry of Interaction” often refers to methods of static execution of λ -terms by a token machine [28, 7] (inspired by a simplification of Girard’s first GoI [27]), we only refer to Girard’s original programme [44, 42, 46, 49, 50, 51] in this paper.

a proof and its reduction by cut-elimination, thus forgetting how they are computationally related, as explained in Figure 3 [101, Section 1.1].

Around the time of the fourth paper on the GoI [49], Girard concurrently introduced ludics [48] which instead of proof-nets, starts by forgetting all the inessential parts of sequent calculus in order to obtain very general and alogical objects called *designs*. What the latest works on the GoI and ludics have in common is that they start from alogical computational objects from which proofs and formulas are *defined*, and hence, are no more primitive. In particular, formulas are defined by set of computational objects from which we expect some shared common characteristics, and connectives are ways to construct new sets from other sets depending on how their objects interact with each other³. In particular, the (linear) negation \mathbf{A}^\perp is the set of all objects which interact well with the objects of \mathbf{A} , which respects to a binary orthogonality relation \perp opposing objects. Orthogonality relations should be understood as *point of view* on interaction, deciding what a *good* interaction is. Formulas are constructed by *interactive typing* which is reminiscent of realisability interpretations [74, 92] and Riba’s reconstruction of simple types from the untyped λ -calculus [91, Section 3].

In some sense, this leads to a sort of *semantic-free* approach to logic since the meaning of formulas is no more defined by an external semantics but rather by computational interaction between objects in a model of computation chosen as a ground for logic. The meaning of objects become their *possible uses*, which is *internal* since related to how they are shaped. This can be illustrated by a comparison between ludics and Schütte’s completeness proof [95, 16]: in the latter, we either have a proof (in the syntactic world) of a statement or we can construct a counter-model (in the semantic world) of its negation which refutes it, while in the former, we have a *counter-proof* (in the syntactic world) instead. Girard describes this situation as a *monism*⁴, meaning that logic lives in a self-contained homogeneous space where there is nothing but syntactic interaction. In terms of programming, it is like having both a program and its environment expressed as interactive entities of the same kind. This new framework should provide further developments of ideas presented by Curien [22] and Abramsky [1], which were already present in linear logic.

This idea of semantic-free typing allows us to define formulas which would be more general than the formulas of usual proof theory in the sense that the space of proofs of the conjunction $A \wedge B$ would be larger than in usual proof theory. Unusual computational entities may constitute a proof of $A \wedge B$. Hence, formulas become descriptions of computational behaviours in a chosen computational space.

Sufficient conditions for effective reasoning. This interactive interpretation of logic starts by choosing a model of computation but not all choices are equal. Several GoI models were defined using operator algebras [44, 42, 49, 50], term unification algebras [46], graphs [98, 99] and graphings [101, 103, 100]. Although all these models did define rich models, that were in particular used to study computational complexity [15, 10, 11, 102, 104], they had two main drawbacks.

- (1) The objects used to interpret even the most basic proofs were most of the time unnatural and infinite (as in Girard’s initial interpretation using operator algebras [44]).

³This way of typing computational entities is called *l’Usage* (the use) in Girard’s terminology.

⁴In opposition to *dualism*.

In particular, finite reasoning is no more possible⁵ because in order to assert that an object Φ is a proof of \mathbf{A} (which is a set), we need to *test* it (by checking orthogonality) against all elements of \mathbf{A}^\perp which may be infinite (because they represent all the potential partners of interaction). A solution would be to take inspiration from the correctness criterion of proof-nets which allows a finite and sufficient checking of logical correctness. Using term unification [46], a simplification with finite combinatorics has been suggested but was limited for a satisfying treatment of correctness criteria and fall into the same problems as for Seiller’s interaction graphs [98, 99] which are less expressive as they could not naturally express the standard Danos-Regnier correctness criterion [26]. Therefore, we need to find an appropriate model of computation to start with.

- (2) The obtained models did interpret soundly the fragments of linear logic considered, but no completeness results exist⁶.

Recently, Girard published a series of articles [51, 55, 53, 54, 57] sketching the main lines of a new kind of model called *transcendental syntax*. This kind of model would have the qualities of GoI models with more improvements. In particular, unlike previous models, the transcendental syntax allows for a satisfying treatment of correctness criterion for proof-nets which leads to completeness results by extending the model of flows [46]. Inspired by the correctness criterion, the new idea introduced by the transcendental syntax is that in order to check whether a computational object Φ is a proof of \mathbf{A} , we only test it against a primitive finite set $\mathbf{a} \subseteq \mathbf{A}^\perp$ which is sufficient to ensure what we wish for⁷. In some sense, it introduces in logic a notion of testing close to the usual program testing of software engineering. Once the objects are certified, we need to mathematically justify that the tests ensure a sound use, a property Girard calls *adequacy* which is similar to the adequacy property in Krivine realisability.

The main problem is that these articles are too inexact in form to serve satisfactorily as the basis of a mathematical theory⁸. The current work is the first step towards a proper formal account of the model.

1.1. Contributions of the paper. The contributions of this paper are the following:

- In Section 2, we formally describe a model of computation we call “stellar resolution”. It is based on Robinson’s first-order resolution [93] and extends/corrects the model of computation vaguely described in Girard’s transcendental syntax. In Section 3.4, we prove the main properties of the model and state its Turing-completeness (Proposition 3.7). In particular, while Girard claimed the failure of the Church-Rosser property for stellar resolution, we show it holds under some conditions (Theorem 3.18). We also relate the dynamics of our model to the construction of tiling-based computation (*cf.* Section 2.1).
- In Section 3, we encode few standard models of computation such as logic program and Turing machines. In Section 3.3, we relate our formalism to tiling-based models: Wang tiles [110] and the abstract tile assembly model [111, 87], which has applications in DNA

⁵This fact is the starting point of the philosophical motivation of the transcendental syntax: the search for the conditions of possibility of (logical) language.

⁶While this aspect is a failure somehow, it is also a feature as the models are very rich and open other paths of reflection.

⁷This way of typing computational entities is called *l’Usine* (the factory) in Girard’s terminology.

⁸The formulation is borrowed from Church’s criticism of von Mises’ notion of kollektiv [21].

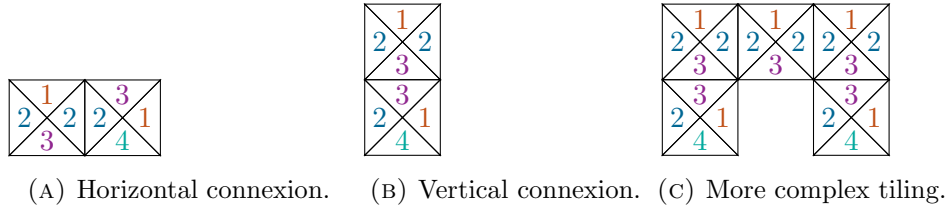


FIGURE 5. Tiling with two wang tiles.

computing [97, 113]. In particular, the stellar resolution is seen as a very general model of computation describing a dynamic exchange of information within a hypergraph-like structure, thus subsuming automata and tile systems.

- We explain how our model captures the dynamic of cut-elimination for the multiplicative fragment of linear logic (MLL) in Theorem 4.19, and the correctness criterion for proof-structures in Theorem 4.31. This implicitly relates MLL proofs (hence the linear simply typed λ -calculus) to the models of computation we encode in Section 3. We also correct some minor technical mistakes appearing in Girard's introducing paper on the transcendental syntax [55].
- In Section 5, we show how MLL formulas can be defined only from the execution of the stellar resolution and a binary orthogonality relation \perp opposing constellations (the objects of stellar resolution). This reconstruction of types suggests the possibility of speaking about type systems which could be applied to models such as automata, logic programs and tiling models as especially fine-grained specifications. Two typing methods are presented: types as labels in Section 5.1 defining types as set of computational entities passing some finite tests and types as computational behaviours in Section 5.2 representing types as sort of potentially infinite ideals. The section ends with a short discussion about multiplicative units in Section 5.3.
- We prove soundness and completeness of the model *w.r.t.* both MLL (*cf.* Theorem 6.12 and 6.14) and MLL+MIX (*cf.* Theorem 6.7 and 6.11), an extension of MLL with the so-called MIX rule [37]. A comment about Girard's adequacy property is given in Section 6.3.

2. STELLAR RESOLUTION

The stellar resolution is a new model of computation introduced in this paper as a computational ground for logic. For pedagogical purposes and for its proximity with these models, we present our model of computation as a tile system which can simulate logic programs by evaluating tilings and comment how it differs from other existing models appearing in the literature of logic programming at the end of Section 3.4.

We recall definitions of terms, unification and resolution in Appendix A which are essential.

2.1. From tile systems to logic programs. We start with the simple and common Wang tiles [110] which are very intuitive and present generalisations leading to our model of stellar resolution.

Wang tiles are square bricks with four sides containing a value (usually an integer or a colour when presented graphically). We can construct tilings by placing copies of tiles

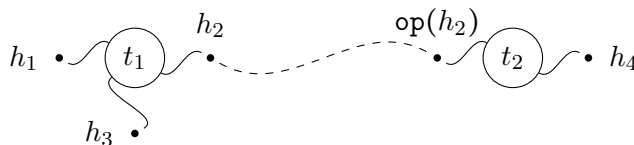


FIGURE 6. Tiling of flexible tiles connected by two complementary flexible arms.

on the plane \mathbf{Z}^2 . Two tiles can be connected along two opposite sides when they hold the same value. We give three examples of partial tilings in Figure 5. We usually require the tilings to be *maximal* (it cannot be extended with further copies of a tile) and *connected*. Moreover, the rotation of tiles is usually forbidden (although this condition can be relaxed in some cases).

A possible generalisation is the model of flexible tiles [68] used as a model for DNA computing with branched junction molecules [19]. It works with star-like tiles having flexible arms which contain values from a given set H . An involution⁹ $op : H \rightarrow H$ such that $op(h) \neq h = op(op(h))$ defines complementary values called *Watson-Crick complementaries*. Two flexible arms can be connected if they have complementary values (*cf.* Figure 6). Notice that the model is not limited to planarity, unlike Wang tiles. Surprisingly, this model can actually encode Wang tiles or other “planar” tiling-based models [69].

It is possible to generalise even more by considering polarised terms with a head symbol (*e.g.* $+c(X)$ but not $+X$) as values for flexible arms such that two terms can be connected when they are unifiable up to renaming, *i.e.* they can be made equal by a substitution from variables to terms (*cf.* Appendix A). It is more general than flexible tiles because terms can encode any set and term unification up to renaming potentially involves several possible partners. For instance, $+c(X)$ can be connected with $-c(t)$ for any term t since the substitution $\theta = \{X \mapsto t\}$ is a solution of the equation $X \stackrel{?}{=} t$ (with a renaming of t in order to avoid variable conflicts).

These flexible tile sets with terms are equivalent to first-order formulas in prenex conjunctive normal form [93]: flexible arms are first-order atoms $P(t), -P(t)$, tiles are disjunctive clauses $\{A_1, \dots, A_n\}$ with bound variables and tile sets are conjunctions of clauses. Robinson’s first-order resolution [93] induces a procedure of evaluation of tilings by successive contraction of links. For instance, the two connected clauses of Figure 7 merge and produce the new clause $[g(f(Y)), -b(f(Y)), +c(Y)]$ where the solution θ of the equation $a(X) \stackrel{?}{=} a(f(Y))$ associated to the link is propagated to the neighbours. The evaluation of a whole set of clauses is defined as follows:

- (1) construct all possible connected tilings by connecting clauses along unifiable terms of opposite polarity (non-determinism can happen);
- (2) eliminate the unwanted ones (typically, we need maximal tilings which cannot be extended);
- (3) try to contract all the tilings by using Robinson’s resolution rule:
 - if it fails, throw the tiling away;
 - if it works, it should give you a new star;

⁹A function f is an involution (or is involutive) when $f(f(x)) = x$

$$\theta = \frac{[g(X), +a(X), -b(X)]}{\{X \mapsto f(Y)\} \Big|} \quad \rightsquigarrow \quad \frac{[g(f(Y)), -b(f(Y)), +c(Y)]}{[-a(f(Y)), +c(Y)]}$$

FIGURE 7. Robinson's first-order resolution seen as a tiling model. The two terms $+a(X)$ and $-a(f(Y))$ are dual and then can be connected to form a tiling.

(4) at the end, you obtain a new constellation with stars coming from all the contracted tilings.

It corresponds to the computation of all new clauses we can infer from the available ones. At this point, our flexible tile system with terms is indeed a fancy way to present first-order resolution.

The model we call stellar resolution is a graph-theoretic variant of the above tile system where we allow additional features such as unpolarised terms and internal polarised terms. It is possible because we are not interested anymore in the logical meaning of our constructions.

The difference between logic programming and our approach is that our model is used with different motivations and with less constraints. In particular, it is allogical so it does not follow logical rules. More details about the stellar resolution and approaches of logic programming are detailed at the end of Section 3.4.

2.2. Stars and constellations. We use Girard's terminology of *stars* and *constellations* [55]. The tiles are called *stars* and their flexible arms are *rays*. A tile set is called a *constellation*. Rays can contain special polarised symbols called *colours* which are analogous to the colours of Wang tiles. For instance, if f is a symbol, then $+f$ and $-f$ are two *dual* colours. We then expect terms such as $+c(f(X))$, $f(X)$, Y , $+d(X, -e)$ and $+c(f(+f(X), Y))$ to be rays. The point lies in the technical ability to switch colours and play with the potential connexions of constellations, *e.g.* turning all colours $-c$ into $-d$.

Definition 2.1 (Coloured signature). A *coloured signature* is a tuple $\mathbb{C} = (V, C, F, \mathbf{ar}, \mathbf{op}, [\cdot])$ where V is a countable set of variables, C and F are disjoint countable set of function symbols such that C is called the set of *colours* and $\mathbf{ar} : C \uplus F \rightarrow \mathbf{N}$ associates an arity to function symbols. Colours in C are new function symbols $+f, -f$ constructed by juxtaposing a polarity in $\{-, +\}$ and a function symbol $f \in F$, and \mathbf{op} is an involution defined by $\mathbf{op}(+f) = -f$. The *underlying symbol* $[c]$ of a colour $c \in C$ such that $c \in \{+f, -f\}$ is defined by $[+f] = [-f] = f$ with f .

We assume the existence of a coloured signature $\mathbb{C} = (V, C, F, \mathbf{ar}, \mathbf{op}, [\cdot])$ unless we explicitly use a specific one.

Definition 2.2 (Rays). A *ray* on a signature $\mathbb{C} = (V, C, F, \mathbf{ar}, \mathbf{op}, [\cdot])$ is a term $r \in \mathbf{Terms}(\mathbb{C})$ constructed with variables in V and function symbols in $C \uplus F$ (*cf.* Appendix A).

A ray r is *coloured* if there is a function symbol f appearing in r such that $f \in C$ and it is *uncoloured* otherwise.

The underlying term of a ray is defined inductively as follows:

$$[x] = x \quad [c(r_1, \dots, r_n)] = [c]([r_1], \dots, [r_n]) \quad [f(r_1, \dots, r_n)] = f([r_1], \dots, [r_n])$$

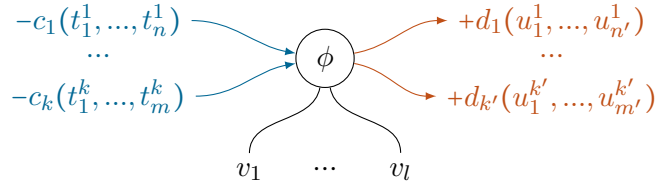


FIGURE 8. Star with prefixed rays seen as either input, output or unpolari-
sided. For general stars, internal colours are allowed as well.

with $x \in V, f \in F$ and $c \in C$.

Notice that the model is more complex than one would expect from the intuitions given in Section 2.1 because of the possible presence of internal colours. Actually, in this paper, only rays with colours as prefix will be used, hence rays $c(t_1, \dots, t_n)$ such that $c \in C$ and the terms t_i are uncoloured. Such terms look like atomic first-order formulas $P(t), -P(t)$ where P is a predicate. Although not used here, we choose to keep the more general definition of rays with internal colours in order to anticipate further works and extensions already described by Girard [54, Section 4.1].

Definition 2.3 (Star, Figure 8). A star ϕ over a coloured signature \mathbb{C} is a finite indexed family¹⁰ of rays, *i.e.* a finite set of indexes I_ϕ together with a map $\phi[\cdot] : I_\phi \rightarrow \text{IdRays}(\mathbb{C})$ which given an index gives the corresponding ray. The set of variables appearing in ϕ is defined by $\text{vars}(\phi) := \bigcup_{i \in I_\phi} \text{vars}(\phi[i])$ (*cf.* Appendix A). For convenience, stars will be written as a clause $[r_1, \dots, r_n]$.

The empty star is written $[\]$ and is defined by $I_{[\]} = \emptyset$.

Notation 2.4 (Substitutions extended to stars). A *substitution* (*cf.* Appendix A) is a function replacing variables by terms, within a term. Given a substitution θ , its action extends to stars by $\theta[r_1, \dots, r_n] = [\theta r_1, \dots, \theta r_n]$.

A *renaming* is a substitution replacing variables by other variables. We say that two stars ϕ_1, ϕ_2 are α -equivalent, written $\phi_1 \approx_\alpha \phi_2$, when there exists a renaming α such that $\alpha\phi_1 = \phi_2$.

Convention 2.5. In this paper, stars will be considered up to α -equivalence. We therefore define $\text{Stars}(\mathbb{C})$ as the set of all stars over a coloured signature \mathbb{C} , quotiented by \approx_α .

Definition 2.6 (Constellation). A *constellation* Φ is a countable indexed family of stars, *i.e.* a countable (possibly infinite) set I_Φ together with a map $\Phi[\cdot] : I_\Phi \rightarrow \text{Stars}(\mathbb{C})$. For convenience, a finite constellation will be written as a sum of stars $\Phi = \phi_1 + \dots + \phi_n$.

We define the set of rays of a constellation Φ by $\text{IdRays}(\Phi) = \{(i, j) \mid i \in I_\Phi, j \in I_{\Phi[i]}\}$ (we keep track of the star from which rays come) and $\pm\text{IdRays}(\Phi) := \{r \in \text{IdRays}(\Phi) \mid r \text{ is coloured}\}$ by its restriction to coloured rays.

The empty constellation is written \emptyset and is defined by $I_\emptyset = \emptyset$.

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.

¹⁰Which should be understood as an array indexed by a given set of indexes (typically, natural numbers).

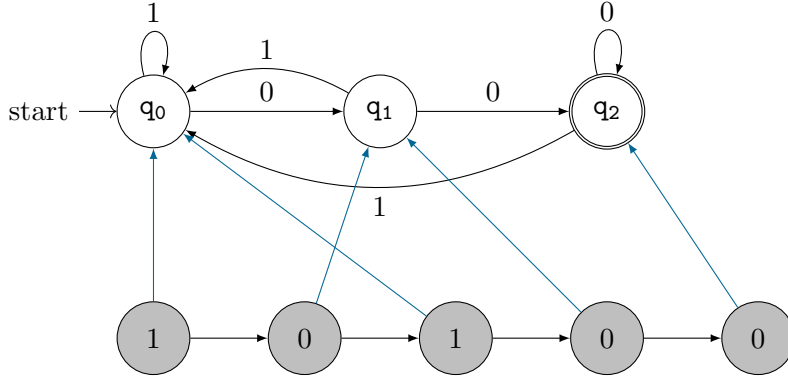


FIGURE 9. Example of finite deterministic automata with a mapping from a word graph to its state graph.

Notation 2.7 (Indexed set membership). We will sometimes write $e \in E$ for an indexed set E (a star or constellation in our case). The intended meaning is that there is an $i \in I_E$ such that $e = E[i]$.

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 through the paper and illustrate the model by some example of constellations.

Definition 2.8 (Encoding of natural numbers). We define the function symbol \bar{n} for a natural number $n \in \mathbf{N}$ by $\bar{0} = 0$ and $\overline{n+1} = s(\bar{n})$ for a unary symbol s and a constant 0 .

Examples 2.9. We give examples of finite and infinite constellations:

- $\Phi_{BH}^c := [-c(X), +c(X)]$;
- $\Phi_{\mathbf{N}}^+ := [+add(\bar{0}, Y, Y)] + [-add(X, Y, Z), +add(s(X), Y, s(Z))]$ (logic program for addition);
- $\Phi_{\mathbf{N}}^{n+m} := \Phi_{\mathbf{N}}^+ + [-add(\bar{n}, \bar{m}, R), R]$ (query for the computation of $n + m$);
- $\Phi_{\mathbf{N}}$ is defined by $I_{\Phi_{\mathbf{N}}} = \mathbf{N}$ and $\Phi_{\mathbf{N}}[i] := [-nat(\bar{i}), +nat(\overline{i+1})]$ (infinite chain)

over the signature defined by the variables $V = \{X, Y, Z, R\}$, the colours $C = \{\pm add, \pm nat\}$, and $F = \{add, nat, s, 0\}$, $\text{ar}(add) = 3$, $\text{ar}(nat) = \text{ar}(s) = 1$, $\text{ar}(0) = 0$. The constellation $\Phi_{\mathbf{N}}^{n+m}$ corresponds to the following Horn clauses [107] where $Add(X, Y, Z)$ states that $X + Y = Z$: $Add(0, Y, Y)$ and $Add(X, Y, Z) \Rightarrow Add(s(X), Y, s(Z))$.

Notation 2.10 (Disjoint union of constellations). Let Φ_1 and Φ_2 be two constellations. Their disjoint union $\Phi_1 \uplus \Phi_2$ is a constellation defined by $I_{\Phi_1 \uplus \Phi_2} := I_{\Phi_1} \uplus I_{\Phi_2}$ and the associated copairing $\Phi_1[\cdot] \uplus \Phi_2[\cdot] : I_{\Phi_1} \uplus I_{\Phi_2} \rightarrow \text{Stars}(C)$.

2.3. Evaluation of diagrams and execution of constellations. We are now interested in the formation of tilings we call *diagrams*. Unlike tilings with Wang tiles or flexible tiles, it is possible to evaluate these diagrams by contracting them with Robinson's resolution rule [93].

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 (*cf.* Appendix A). The *evaluation* of a diagram will correspond to solving its associated unification problem and producing a new star.

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 automata of final state q_2 is given in Figure 9 (on the top with vertices q_0, q_1, q_2). A word can be represented as a linear graph (on the bottom of Figure 9 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 (blue links).

The idea is that the state graph of the automata shows the allowed transitions (where loop can appear) and the word graph is a tiling of states or a traversal of graph which follows those allowed transitions (sometimes by *unfolding* loops).

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, a diagram will be associated to 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.

Definition 2.11 (Duality between rays). Let $\text{op}(r)$ the inverse of a ray r defined by invert-polarities, *i.e.* $\text{op}(r)$ is r where all colours c are replaced by $\text{op}(c)$.

Two rays r and r' are *dual w.r.t.* a set of colours $A \subseteq C$, written $r \varkappa_A r'$, when both have at least one colour $c \in A$ and $\{r \stackrel{?}{=} \text{op}(r')\}$ has a solution.

Proposition 2.12. *The relation \varkappa_A is symmetric but not reflexive nor transitive.*

Proof. Notice that only symmetric relations are used in the definition (equality and unification). Hence, it follows that duality is also symmetric. The failure of reflexivity and transitivity comes from the requirement of opposite colours. A ray cannot be dual to another ray where two identical polarities face each other. \square

Example 2.13. We have $+c(X) \varkappa_{\{c\}} -c(0)$ and $-d(X) \varkappa_{\{d\}} +d(f(X))$ but not $+c(X) \varkappa_A f(Y)$ (presence of unpolarised ray), $+c(X) \varkappa -d(X)$ (different head symbol), $+c(X) \varkappa_A +c(f(Y))$ (polarities are not opposite) nor $+c(f(X)) \varkappa_A -c(g(Y))$ (terms are not α -unifiable) for any A .

Definition 2.14 (Dependency graph). The *dependency graph* of a constellation Φ *w.r.t.* a set of colours $A \subseteq C$ is the undirected labelled multigraph¹¹ $\mathfrak{D}[\Phi; A] := (V, E, \ell)$ where $V := I_\Phi$ and for each $(i, j), (i', j') \in \pm\text{IdRays}(\Phi)$ such that $\Phi[i][j] \varkappa_A \Phi'[i'][j']$, we have $\{i, i'\} \in E$ and the edge labelling $\ell(i, i') = (j, j')$. We simply write $\mathfrak{D}[\Phi]$ when links for all colours appearing in Φ are allowed.

Example 2.15. Two examples of dependency graphs for the two constellations $\Phi_{\mathbb{N}}^{n+m}$ and $\Phi_{\mathbb{N}}$ of Example 2.9 are presented in Figure 10.

¹¹A multigraph is a graph with possibly several edges between two same vertices.

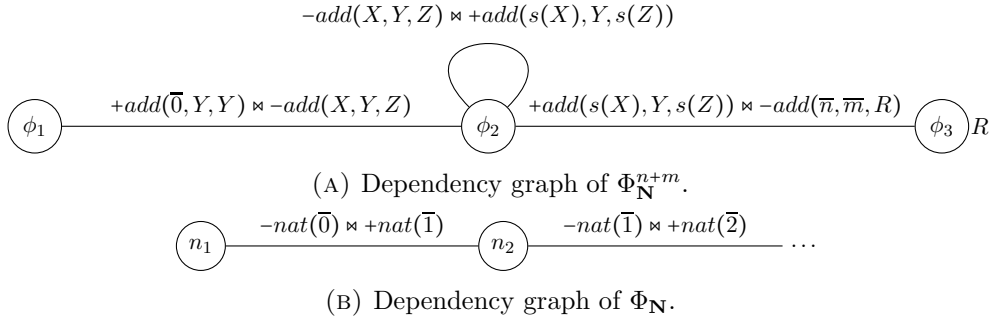


FIGURE 10. Examples of dependency graphs for constellations of Example 2.9.

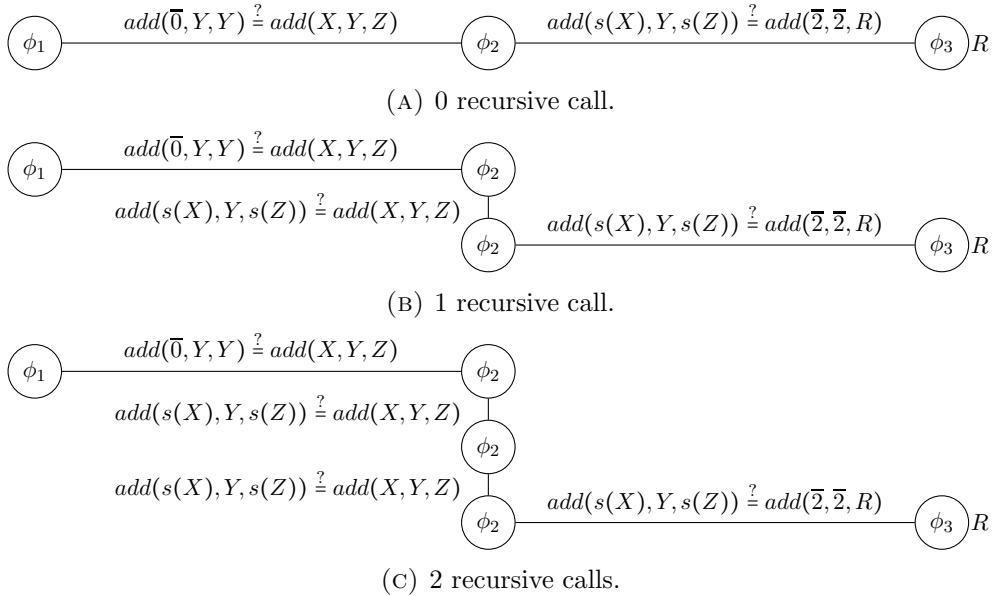
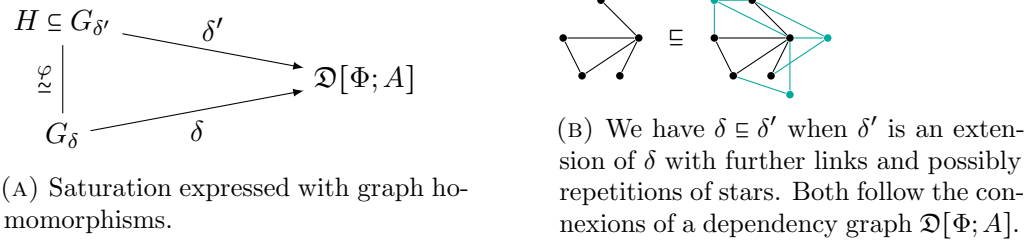


FIGURE 11. Examples of diagrams for the constellation $\Phi_{\mathbf{N}}^{2+2}$. The number of occurrences of ϕ_2 corresponds to the number of recursive calls. They correspond to unfolding of the loop of Figure 10 corresponding to the possibility of recursive call.



FIGURE 12. Ray linking graph for the first diagram of Figure 11 corresponding to the case with no recursive call.

Definition 2.16 (Diagram). An *A*-*diagram* (or simply *diagram* when working with all colours) δ on a set of colours $A \subseteq C$ over a constellation Φ is a graph homomorphism $\delta : G_\delta \rightarrow \mathcal{D}[\Phi; A]$ from a non-empty finite connected multigraph G_δ .


 FIGURE 13. Order \sqsubseteq on diagrams representing an idea of saturation.

We define the set of rays of δ by $\text{IdRays}(\delta) := \{(\delta(x), j) \mid x \in V^{G_\delta}, j \in I_{\delta(x)}\}$ (we keep track of the stars from which the rays come) and extend the definition to the set $\pm \text{IdRays}(\delta)$ of its coloured rays.

The labelling function ℓ of $\mathfrak{D}[\Phi; A]$ is extended to a diagram δ by $\ell(e) := \ell(\delta(e))$ for $e \in E^{\mathfrak{D}[\Phi; A]}$.

The *ray linking graph* $\text{RLG}(\delta)$ of δ is a graph showing how rays are linked (instead of stars). It is a graph (V, E) defined with $V := \text{IdRays}(\delta)$ such that $(j, j') \in E$ when j and j' are linked in G_δ , *i.e.* when there is some $e \in E^{G_\delta}$ of label (j, j') .

Finally, we require that a diagram has a ray linking graph which is a simple graph (A graph without loop on vertices and without multiple edges between two vertices).

The graph G_δ is considered up to renaming of the vertices and edges and for convenience, we will often have $V \subseteq \mathbf{N}$ in practice.

Example 2.17. 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 2.9) is given in Figure 11. An example of ray linking graph for the first diagram is given in Figure 12.

Notation 2.18 (Free rays and closed diagrams). Given an A -diagram δ of a constellation Φ , we define its set of *free* (unconnected) rays $\text{free}(\delta) \subseteq V^{\text{RLG}(\delta)}$ by the set of isolated rays index in $\text{RLG}(\delta)$. If $\text{free}(\delta) = \emptyset$, we say that δ is *closed*.

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.

Definition 2.19 (Saturated diagram). We define a binary relation \sqsubseteq (illustrated in Figure 13) on A -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 A -diagram *w.r.t.* \sqsubseteq is called *saturated*.

Proposition 2.20. *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. \square

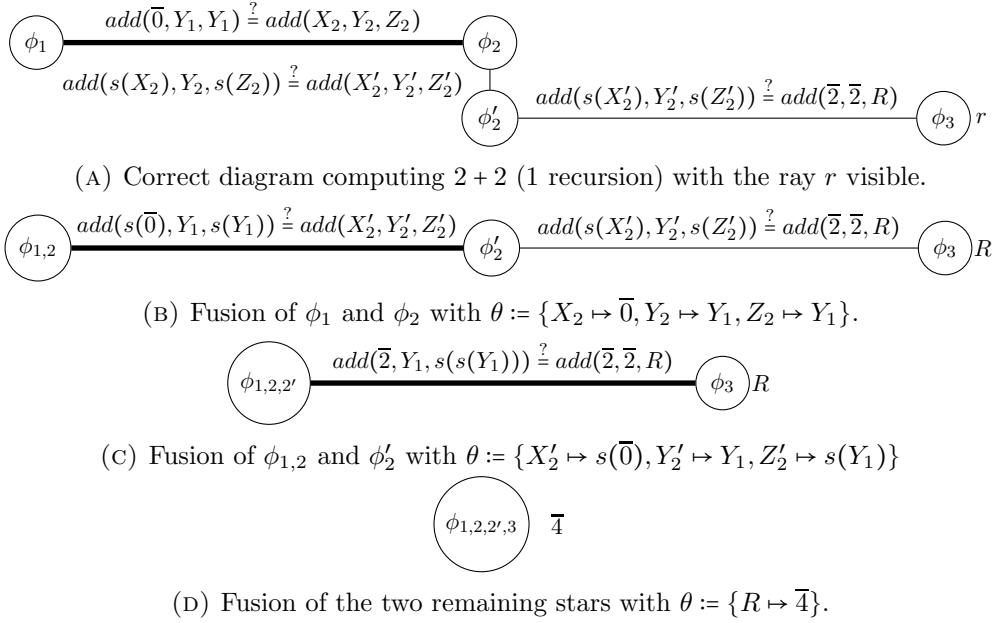


FIGURE 14. Fusion of the diagram from Figure 11b.

Links in a diagram have an underlying equation. It follows that a whole diagram is associated to a unification problem (*cf.* Appendix A). 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 .

Definition 2.21 (Underlying equation and problem). Let δ be an A -diagram of a constellation Φ . We define a canonical family of renamings for variables defined by $\alpha_v(x) = x_v$ for $v \in V^{G_\delta}$ and any variable x .

The *underlying equation* of a link $e = (v, v')$ of label (j, j') in E^{G_δ} is defined by $\mathbf{eq}(e) := \alpha_v[\Phi[v][j]] \stackrel{?}{=} \alpha_{v'}[\Phi'[v'][j']]$ and the *underlying problem* of δ is defined by $\mathcal{P}(\delta) = \{\mathbf{eq}(e) \mid e \in E^{G_\delta}\}$.

In Girard's original paper [55, Section 2.3], the evaluation of diagrams is defined as an edge contraction we call *fusion*. An edge e between two stars ϕ and ϕ' contain equations which are resolved and then the associated solution is propagated to both ϕ and ϕ' . The two connected rays associated to 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.

We define this step-by-step procedure of fusion but also an alternative and equivalent notion of evaluation we call *actualisation* which evaluates a diagram by solving the whole unification problem associated. It is similar to how small step evaluation differ to big step evaluation in the theory of programming languages [6, Section 1.1].

Definition 2.22 (Fusion, Figure 15). Let $\delta : G_\delta \rightarrow \mathfrak{D}[\Phi; A]$ be an A -diagram of a constellation Φ . We define the *fusion* of δ along a link $e = (v, v')$ in E^{G_δ} of label (j, j') as a new diagram $\delta' : G_{\delta'} \rightarrow \mathfrak{D}[\Phi + \phi; A]$ where $G_{\delta'}$ is G_δ such that:

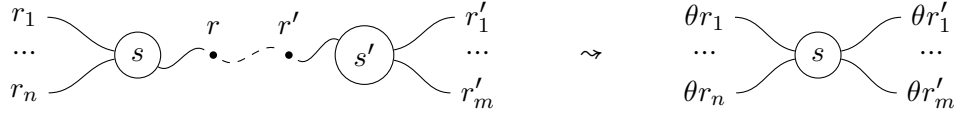


FIGURE 15. Illustration of a step of fusion where θ is the principal unifier of the underlying unification problem of the pair of rays (r, r') . The fusion of the two stars s and s' along the rays r and r' produces a new star s .

- (1) we compute $\theta := \mathbf{solution}(\{\mathbf{eq}(e)\})$;
- (2) we define $\phi_1 := \Phi[\delta(v)]$ and $\phi_2 := \Phi[\delta(v')]$;
- (3) we define ϕ'_1 by $I_{\phi'_1} := I_{\phi_1} \setminus \{j\}$ and ϕ'_2 by $I_{\phi'_2} := I_{\phi_2} \setminus \{j'\}$ and $\phi[i]$ behaves like $\phi'[i']$;
- (4) we define a new star $\phi := \theta\phi'_1 \uplus \theta\phi'_2$;
- (5) v and v' merge and are replaced by ϕ , i.e. $\delta(v') = \delta(v) = \phi$ and some $x \in V^{G_\delta}$ is linked with v if and only if it is connected to v' .

We use the notation $G_\delta \rightsquigarrow_e G_{\delta'}$ for a step of this procedure resulting in $G_{\delta'}$, $G_\delta \rightsquigarrow_e^* G_{\delta'}$ for its reflexive transitive closure and $G_\delta \rightsquigarrow_e^n G_{\delta'}$ for the reachability of $G_{\delta'}$ from G_δ in $n \in \mathbf{N}$ steps. We leave the reduced edge e implicit when obvious or not important.

Definition 2.23 (Correct diagrams and their actualisation). An A -diagram δ of a constellation Φ is *correct* if $\mathcal{P}(\delta)$ has a solution.

The *actualisation* of a correct diagram δ is the star $\Downarrow \delta$ defined by $I_{\Downarrow \delta} := \mathbf{free}(\delta)$ such that $(\Downarrow \delta)[(i, j)] = (\psi \circ \theta)(\Phi[i][j])$, where $\psi = \mathbf{solution}(\mathcal{P}(\delta))$ and $\theta := \alpha_{v_1} \circ \dots \circ \alpha_{v_n}$ with $V^{G_\delta} = \{v_1, \dots, v_n\}$ is the composition of renamings of Definition 2.21.

There are several ways to compute the solution of a unification problem. In this paper we use the Martelli-Montanari algorithm [80]. We call *partial execution* of a problem P an arbitrary sequence of steps of the algorithm applied on P . Further details are given in Appendix A.

In the following proof, we treat $\mathbf{free}(\delta)$ as a star made of the free rays of δ for readability.

Lemma 2.24 (Equivalence of diagram reduction). *For all diagram δ , there exists δ' such that $G_\delta \rightsquigarrow_e G_{\delta'}$ if and only if there exists a partial execution from $\mathcal{P}(\delta)$ to $\mathcal{P}(\delta') \cup \{X_1 \stackrel{?}{=} t_1, \dots, X_k \stackrel{?}{=} t_k\}$ with $\{X_1, \dots, X_k\} \cap \bigcup_{j=1}^k \mathbf{fv}(t_j) = \emptyset$ and $\mathbf{free}(\delta') = \{X_1 \mapsto t_1, \dots, X_k \mapsto t_k\} \mathbf{free}(\delta)$. It means that a step of fusion corresponds to some steps of the unification algorithm (cf. Appendix A).*

Proof. We show the two implications.

- (\Rightarrow) Assume the fusion succeeds and produces a graph $G_{\delta'}$ by using the substitution $\theta := \{X_1 \mapsto t_1, \dots, X_k \mapsto t_k\}$ corresponding to $\mathbf{solution}(\{\mathbf{eq}(e)\})$. We now consider the actualisation of δ which corresponds to solving the equation associated to δ . By confluence of the unification algorithm (cf. Appendix A), we can focus on e and isolate the result in order to obtain $\mathcal{P}(\delta') \cup \{X_1 \stackrel{?}{=} t_1, \dots, X_k \stackrel{?}{=} t_k\}$ with $\{X_1 \stackrel{?}{=} t_1, \dots, X_k \stackrel{?}{=} t_k\}$ in solved form (notice that $\mathcal{P}(\delta') = \mathcal{P}(\delta) \setminus \{e\}$). Then we have $\{X_1 \mapsto t_1, \dots, X_k \mapsto t_k\} \mathcal{P}(\delta') \cup \{X_1 \stackrel{?}{=} t_1, \dots, X_k \stackrel{?}{=} t_k\}$ by application of the unification algorithm (“replace” rule) on the equations $X_i \stackrel{?}{=} t_i$ we previously isolated and “stored”. We obtain the equations corresponding to

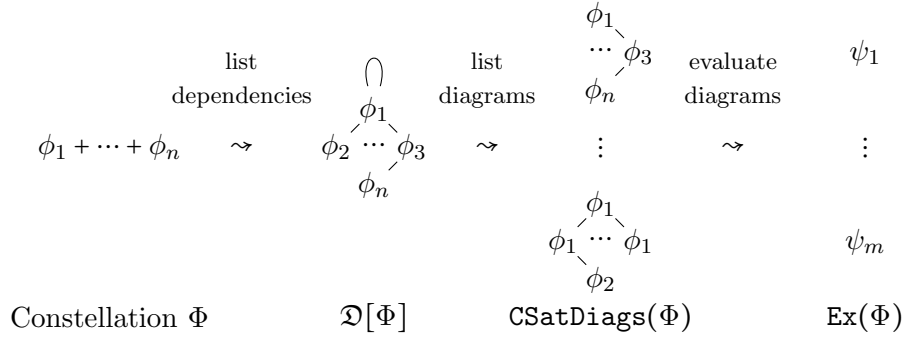


FIGURE 16. Illustration of the execution of a finite and strongly normalising constellation. Notice that diagrams can be thought of as sort of runs in the dependency graph seen as a generalised automaton.

a new diagram δ'' such that $\mathcal{P}(\delta'') = \{X_1 \mapsto t_1, \dots, X_k \mapsto t_k\} \mathcal{P}(\delta')$. After application of $\text{solution}(\{\text{eq}(e)\})$ on $\mathcal{P}(\delta')$, the variables X_1, \dots, X_k are “fixed”, *i.e.* they appear nowhere else, which prevents them to be altered during the execution of the algorithm and hence the substitutions of $\{X_1 \mapsto t_1, \dots, X_k \mapsto t_k\}$ will appear in the last substitution applied on the free rays. It shows that we will have $\text{free}(\delta'') = \{X_1 \mapsto t_1, \dots, X_k \mapsto t_k\} \text{free}(\delta)$ if we consider a notion of partial actualisation. This corresponds to δ' , hence $\delta' = \delta''$.

- (\Leftarrow) The previous point defines a correspondence between a step of fusion and some steps of the unification algorithm. By confluence of the unification algorithm, it is always possible to reorganise the order of resolution of equation so that the first step will correspond to a step of fusion, without any effect on the result. □

Theorem 2.25 (Equivalence between fusion and actualisation). *For all diagram δ , we have $G_\delta \rightsquigarrow^n G_{\Downarrow \delta}$ for $n = |\mathcal{P}(\delta)|$.*

Proof. By induction on n . Assume we have 0 links, hence G_δ does not reduce and has no edges. The only connected graph with no edge is a single vertex. This indeed corresponds to $G_{\Downarrow \delta}$, as expected. For the inductive case, we show that there exists a diagram δ' such that $G_\delta \rightsquigarrow G_{\delta'} \rightsquigarrow^n G_{\Downarrow \delta}$ knowing $G_{\delta'} \rightsquigarrow^n G_{\Downarrow \delta}$ (by induction hypothesis). The simulation of fusion (*cf.* Lemma 2.24) tells us that a step of fusion exactly corresponds to some steps of the unification algorithm. Consider a full application of the unification algorithm on $\mathcal{P}(\delta)$. By the confluence of the algorithm (*cf.* Appendix A), we can reorganise the computation of $\Downarrow \delta$ so that some steps correspond to $G_\delta \rightsquigarrow G_{\delta'}$ and the remaining ones to $G_{\delta'} \rightsquigarrow^n G_{\Downarrow \delta}$. Hence, we necessarily have a step of fusion $G_\delta \rightsquigarrow G_{\delta'}$. □

The *execution* of a constellation Φ (Figure 16) consists in computing all the correct saturated diagrams of Φ and actualising them. In appearance, it is very similar to the resolution operator [78, Chapter 3] which is analogous to the consequence operator [29, Section 2.2] of logic programming. 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 in Section 3.1, allowing cyclic diagrams still preserves the interpretation of logic programs since cyclic diagrams are often wrong for logic programs: for the

constellation $\Phi_{\mathbf{N}}^{n+m}$ of Example 2.9, 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.

Notation 2.26 (Set of correct saturated diagrams). We write $\text{SatDiags}_A(\Phi)$ for the *set of all saturated diagrams* obtained from $\mathfrak{D}[\Phi; A]$ for a constellation Φ and a set of colours $A \subseteq C$. We omit the set of colours and simply write $\text{SatDiags}(\Phi)$ when $A = C$.

We write $\text{CSatDiags}_A(\Phi)$ for the set of all diagrams in $\text{SatDiags}_A(\Phi)$ which are correct.

Definition 2.27 (Execution and normal form). The *execution* of a constellation Φ *w.r.t.* a set of colours $A \subseteq C$ is defined by $\text{Ex}_A(\Phi) := \Downarrow \text{CSatDiags}_A(\Phi)$, where $\Downarrow \text{CSatDiags}_A(\Phi) := \{\Downarrow \delta \mid \delta \in \text{CSatDiags}_A(\Phi)\}$. We write $\text{Ex}(\Phi)$ when all colours in Φ participate in the execution.

We discuss some design choices. Notice that the definition of diagram allows duplication of a same star, hence this apparently makes no difference whether or not a constellation is defined as a set or multiset. The purpose of defining constellations as multiset (actually indexed families) is to allow a quantitative analysis in the normal form. For instance, it would be possible to count how many times a given star appeared in the normal form.

Saturated diagrams of a constellation Φ , although impossible to extend, may have free coloured rays which can be connected to the rays of another constellation Φ' when computing the interaction $\text{Ex}(\Phi \uplus \Phi')$. This is necessary in order to consider composition in logic and prove the associativity of execution (*cf.* Theorem 5.20). However, this definition is different from Girard's original definition [55, Section 2.3] which erases stars containing free coloured rays for technical reasons explicited in our interpretation of MLL (*cf.* Section 4). We instead split Girard's execution by defining more primitive operations which can be combined to our execution.

We define an operation of *concealing* which violently mutes the constellation by removing stars containing polarised rays, thus forbidding any communication with other stars.

Definition 2.28 (Concealing). Let Φ be a constellation. The *concealing* of Φ is the constellation $\cancel{\Phi}$ defined by $I_{\cancel{\Phi}} := \{i \in I_{\Phi} \mid \phi := \Phi[i], \forall j \in I_{\phi}, \phi[j] \text{ is uncoloured}\}$.

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 set in the interpretation of MLL (*cf.* Section 4).

Definition 2.29 (Noise filtering). Let Φ be a constellation. The *noise filtering* of Φ is the constellation $\flat \Phi := \{i \in I_{\Phi} \mid \Phi[i] \neq []\}$.

3. COMPUTATIONAL ILLUSTRATIONS AND PROPERTIES OF EXECUTION

We illustrate how several common kinds of computation can be implemented in our model as certain classes of constellations. It shows that the stellar resolution is a very general and modular model of computation which expresses computation by transmission of data within a hypergraph structure. In particular, this generalises various classes of automata and Horn clauses used in logic programming and various tile systems. Although not explicitly shown in this paper, the stellar resolution should also represent a computational version of labelled transition systems which are commonly used in model checking [14, Chapter 2].

3.1. Logic programs. 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.

First, it is possible to do programming with predicate calculus [71]. It is then known that formulas of predicate calculus can be normalised so that formulas are represented only by conjunction of disjunctions (called clauses) with only universal quantifiers appearing as prefix [59, Section 3.2]. Formulas are then of the shape $\forall x_1, \dots, x_n. (A_1^1 \vee \dots \vee A_n^1) \wedge \dots \wedge (A_1^m \vee \dots \vee A_k^m)$ where every A_y^x is an atomic formula. 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 .

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 $A_1, \dots, A_n \vdash B$ called *Horn clauses* [61, 107]. 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.

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

$$P(t_1, \dots, t_n)^\star := [+P(t_1, \dots, t_n)].$$

For a rule, the translation is defined by:

$$(\wedge_{i=1}^m P_1(t_1^i, \dots, t_n^i) \vdash Q(u_1, \dots, u_k))^\star := (\bigcup_{i=1}^m \{-P_1(t_1^i, \dots, t_n^i)\}) \cup \{+Q(u_1, \dots, u_k)\}.$$

Finally, for a query, we have:

$$(?P(t_1, \dots, t_n))^\star := [-P(t_1, \dots, t_n), r_1, \dots, r_m]$$

with $\{r_1, \dots, r_m\} = \bigcup_{i=1}^n \text{vars}(t_i)$ which represent the information we would like to make visible in the output (see Example 2.9 where $[-add(\bar{n}, \bar{m}, r), r]$ is the query). A logic program $P := \bigcup_{i=1}^n \{C_i\}$ becomes $P^\star := \bigcup_{i=1}^n \{C_i^\star\}$.

The set of answers for a query q on a program P is defined by a set of substitutions $A_P^q = \{\theta_1, \dots, \theta_k\}$ such that for all $\theta \in A_P^q$, we have θq logically satisfied by P , written $P \models \theta q$. The answers are usually computed by iteratively applying the resolution rule (cf. Definition A.7 in Appendix A) 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 [71, 73] for more details about the computation of answers.

Theorem 3.1 (Simulation of logic programs). *Let P be a logic program with query q and P^\star and q^\star be their translation. We have $\text{Ex}(P^\star + q^\star) = \{\theta_1 \phi_1, \dots, \theta_k \phi_k\}$ if and only if for all θ_i , $P \models \theta_i q$.*

Proof. The proof relies on the fact that the execution reproduces the SLD-resolution [71, 73]. The satisfiability of a query is linked to the idea of “proof-search”: it is satisfiable when proved by facts, themselves proved by other facts and so on until nothing is left unproven. SLD-resolution tries to satisfy a query by matching it with the available facts and rules. Stars can actually be seen as first-order disjunctive clauses. Looking for justifications of

facts corresponds to the construction of diagrams and the fact of leaving nothing unproven corresponds to the saturation of diagrams. In particular, in presence of k possible choices of rules, k answers are computed independently in parallel and we obtain saturated diagrams $\delta_1, \dots, \delta_k$.

The fact of matching a query against available facts and rules can be seen as constructing a saturated and correct diagram. In the absence of error, we indeed obtain an instantiation of the variables $\mathbf{vars}(q)$ through a substitution θ_i . This exactly coincides with the actualisation of correct diagrams. By Lemma 2.25 this is equivalent to a full fusion. Having number of correct diagrams corresponds to the number of answers.

Remark that in the case of diagrams, only the free rays survive in the output, hence we have to add uncoloured rays corresponding to $\mathbf{vars}(q)$ to correctly simulate logic programs. We could also add rays $x \cdot X$ where x is a constant representing 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 $\theta \in A_P^q$.

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 $A_1, A_2, \dots, A_n \vdash B$, the equation $t \stackrel{?}{=} t$ associated to 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. \square

An example of logic program computing unary addition and its evaluation for the case of $2 + 2$ is illustrated in Figure 11 and 14.

3.2. Non-deterministic Turing machines. Definitions of Turing machines are taken for Sipser's introduction to the theory of computation [106, Section 3.1 in Second Edition].

Links between tile systems and automata have already been studied [81, 108] by considering recognisability on graphs, inducing sort of *asynchronous automata*. The encoding of automata in the stellar resolution follows a similar idea: the construction of diagrams simulates a run on a given word. It is possible to encode directed graphs by translating edges (e, e') with binary stars $[-g(e), +g(e')]$. It is then possible to encode an automata transitions by first encoding their state graph then extending the rays so that the fusion triggers a transmission of information (the remaining characters to be read). 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 acceptance a word.

In this section, we suggest an encoding of non-deterministic Turing machines. We use the fact 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 (push or pop of a symbol).

Definition 3.2 (Encoding of words). If $w = c_1 \dots c_n$ then $w^\star = [+i(c_1 \cdot \dots \cdot c_n \cdot \sqcup)]$ with the binary function symbol \cdot which is considered right-associative, *i.e.* $a \cdot b \cdot c = a \cdot (b \cdot c)$ and a constant \sqcup for the empty character.

Different encodings of words are possible. For instance, in Aubert and Bagnol's works [8, Definition 24][9, Definition 10], the characters are encoded with objects called *flows* (which can be seen as binary stars) forming a cyclic chain of α -unifiable terms which interact with the encoding of an automaton. This defines a characterisation of logspace computation where the input is explored with pointers.

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, $\delta : Q \times \Gamma_{\perp} \rightarrow \mathcal{P}(Q \times \Gamma_{\perp} \times \{1, \mathbf{r}, \mathbf{s}\})$ is the transition function, q_0 is the initial state and finally, q_a and $q_r \neq q_a$ are respectively the state of acceptance and rejection. We write Γ_{\perp} for $\Gamma \cup \{\perp\}$.

A configuration is a triple (l, q, r) where $q \in Q$ and l, r are tapes. It represents the position of the head 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 $w = c_1 \dots c_n$ is accepted by M , written $M(w) = 1$, when there is a sequence of configurations C_1, \dots, C_n such that:

- (1) $C_1 = (\perp, q_0, w)$;
- (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 , we write $M(w) = \infty$. We require that an NTM necessarily ends on q_a or q_r when it stops.

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.

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.

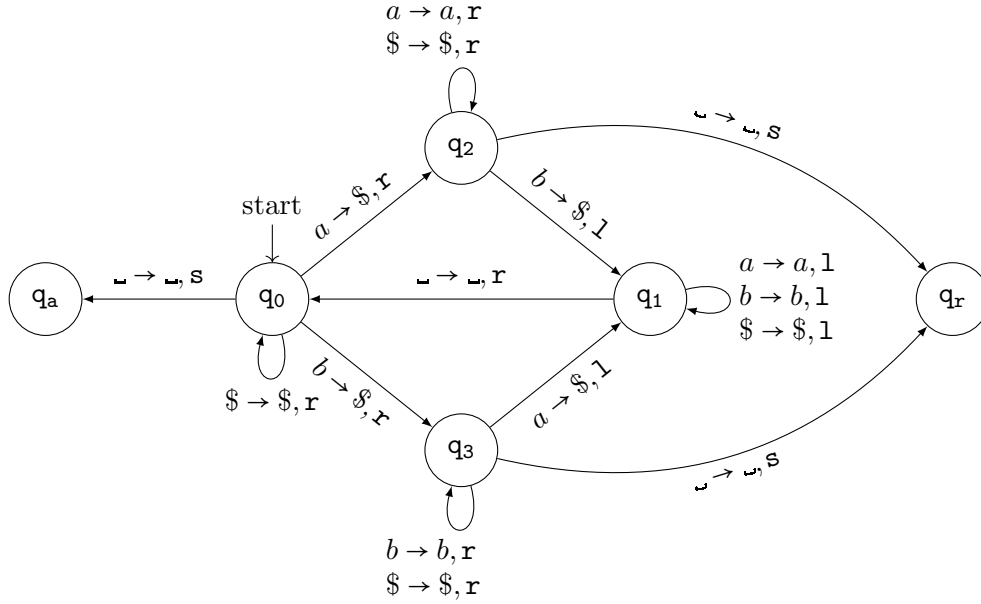
Definition 3.3 (Encoding of non-deterministic Turing machines). The encoding of an NTM $M = (Q, \Gamma, \delta, q_0, q_a, q_r)$ is defined by a constellation M^{\star} such that:

- q_0 is translated into $[-i(C \cdot W), +m(\perp, q_0, C, W)] + [-i(\perp), +m(\perp, q_0, \perp, \perp)]$;
- q_a is translated into $[-m(L, q_a, X, R), \mathbf{accept}]$;
- q_r is translated into $[-m(L, q_r, X, R), \mathbf{reject}]$;
- for each $q \in Q$ and $c \in \Gamma_{\perp}$ such that $(q', c', d) \in \delta(q, c)$:
 - if $d = 1$ (going left) then we have $[-m(L \bullet X, q, c, R), +m(L, q', X, c' \circ R)]$;
 - if $d = \mathbf{r}$ (going right) then we have $[-m(L, q, c, X \circ R), +m(L \bullet c', q', X, R)]$;
 - if $d = \mathbf{s}$ (staying still) then we have $[-m(L, q, c, R), +m(L, q', c', R)]$;
- we add two additional “memory allocation stars”:

$$[-m(\perp, Q, C, R), +m(\perp \bullet \perp, Q, C, R)] + [-m(L, Q, C, \perp), +m(L, Q, C, \perp \circ \perp)].$$

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 $[-m(L, q_a, X, R), \mathbf{accept}(L, X, R)]$.

Theorem 3.4 (Simulation of non-deterministic Turing machines). *Let M be an NTM such that q_a and q_r have no outgoing transitions and w a word. We have:*



$$\begin{aligned}
 M^\star &= [-i(C \cdot W), +m(\sqcup, q_0, C, W)] + [-i(\sqcup), +m(\sqcup, q_0, \sqcup, \sqcup)] + \\
 &\quad [-m(L, q_0, \sqcup, R), +m(L, q_a, \sqcup, R)] + [-m(L, q_2, \sqcup, R), +m(L, q_r, \sqcup, R)] + \\
 &\quad [-m(L, q_0, \$, C \circ R), +m(L \bullet \$, q_0, C, R)] + [-m(L, q_2, \$, C \circ R), +m(L \bullet \$, q_2, C, R)] + \\
 &\quad [-m(L, q_0, a, C \circ R), +m(L \bullet \$, q_2, C, R)] + [-m(L, q_2, a, C \circ R), +m(L \bullet a, q_2, C, R)] + \\
 &\quad [-m(L, q_0, b, C \circ R), +m(L \bullet \$, q_3, C, R)] + [-m(L \bullet C, q_2, b, R), +m(L, q_1, C, \$ \circ R)] + \\
 &\quad [-m(L, q_1, \sqcup, C \circ R), +m(L \bullet \sqcup, q_0, C, R)] + [-m(L, q_3, \sqcup, R), +m(L, q_r, \sqcup, R)] + \\
 &\quad [-m(L \bullet C, q_1, \$, R), +m(L, q_1, C, \$ \circ R)] + [-m(L, q_3, \$, C \circ R), +m(L \bullet \$, q_3, C, R)] + \\
 &\quad [-m(L \bullet C, q_1, a, R), +m(L, q_1, C, a \circ R)] + [-m(L \bullet C, q_3, a, R), +m(L, q_1, C, \$ \circ R)] + \\
 &\quad [-m(L \bullet C, q_1, b, R), +m(L, q_1, C, b \circ R)] + [-m(L, q_3, b, C \circ R), +m(L \bullet b, q_3, C, R)] + \\
 &\quad [-m(L, q_a, X, R), \text{accept}] + [-m(L, q_r, X, R), \text{reject}] + \\
 &\quad [-m(\sqcup, Q, C, R), +m(\sqcup \bullet \sqcup, Q, C, R)] + [-m(L, Q, C, \sqcup), +m(L, Q, C, \sqcup \circ \sqcup)]
 \end{aligned}$$

FIGURE 17. A Turing machine accepting words containing as many symbols a as symbols b where $a \rightarrow b, d$ from a state q to q' corresponds to a transition $\delta(q, a) = (q', b, d)$. When computing $\text{Ex}(M^\star + a^\star)$, we plug the input with the correct initial star and obtain $[+m(\sqcup, q_0, a, \sqcup)]$. No star can be connected, hence we have to connect to the right allocation star and obtain $[+m(\sqcup, q_0, a, \sqcup \circ \sqcup)]$. We can use the star corresponding to $a \rightarrow \$, r$ and obtain $[+m(\sqcup \bullet \$, q_2, \sqcup, \sqcup)]$. Since we read \sqcup , we use the star corresponding to the transition $\sqcup \rightarrow \sqcup, s$ and obtain $[+m(\sqcup \bullet \$, q_r, \sqcup, \sqcup)]$. We can only use the star corresponding to q_r and obtain $[\text{reject}]$. If we had a character b next to a , we would reach $[\text{accept}]$.

- (1) $M(w) = 1$ if and only if $[\text{accept}] \in \text{b} \not\downarrow \text{Ex}(M^\star + w^\star)$;
- (2) $M(w) = 0$ if and only if $([\text{accept}] \notin \text{b} \not\downarrow \text{Ex}(M^\star + w^\star) \text{ and } \text{b} \not\downarrow \text{Ex}(M^\star + w^\star) \neq \emptyset)$.

Proof. By design, $\mathfrak{D}[M^\star + w^\star]$ is isomorphic to the state graph of M (that is, there is a link between two rays if and only if the two corresponding states are adjacent) and each

run is isomorphic to some linear correct saturated diagram (because transitions correspond to binary stars).

A major difference with finite automata is the possibility of infinite computation. Such infinite computation made by constantly going from one state to the another (which can be the same one) corresponds to the existence of a diagram which cannot be saturated and hence does not appear in the normal form. Since q_a and q_r have no outgoing edges, it is impossible to have the non-deterministic choice of either stopping on a terminal state or continuing. If it was possible, we would obtain a misleading `[accept]` or `[reject]` in the output.

We give arguments showing that the dynamics of Turing machine (the transition function) is correctly simulated. We show that the fusion of stars correctly simulates the composition of transitions. Assume we have a star $[+m(l, q, c, r)]$ representing a configuration of the Turing machine. We have three cases depending on the direction:

- if we are going left, we have a transition $[-m(L \bullet X, q, c, R), +m(L, q', X, c' \circ R)]$. The fusion is successful only when l is of the shape $l' \bullet k$. In this case, by unification we have $l' = L$ and $X = c$ which identifies a next symbol on the left. The evaluation produces the star $[+m(L, q', k, c' \circ r)]$ which corresponds to writing c' after reading c and placing it on the right part of the tape to read the symbol k on the left. This indeed corresponds to “going on the left” in the tape;
- if we are going right, the reasoning is similar;
- if we stop the head, we have a transition $[-m(L, q, c, R), +m(L, q', c', R)]$ and the fusion only moves from a state q (when reading a symbol c) to another state q' (after writing the symbol c').

We now check the limit cases when the machine is out of memory (not enough space on the tape to apply a transition). These cases happen because Turing machines have potentially infinite tapes but we only manipulate finite extensible tapes. When going on the left, it happens that the left part of the tape l is not of the shape $l' \bullet k$. The typical case is when we have $l = \perp$. In this case, we can arbitrarily use the left allocation star and freely obtain $\perp \bullet \perp$. Remark that it is impossible to allocate too much space because the allocation stars require that we have a tape equal to \perp . 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.

Using all the previous arguments, we check the statements (1) and (2) of the simulation theorem for Turing machines.

- (1) Assume that $w \in \mathcal{L}(M)$ and there is a non-deterministic run reaching either q_a . By correspondence between runs and diagram for Turing machines, we must have a saturated and correct linear diagram reaching the ray `accept`. Since stars are binary and q_a is terminal, this ray can only be reached once in a diagram. Hence, such a diagram actualises into `[accept]`. Therefore, we have `[accept]` $\in \mathcal{b} \not\downarrow \text{Ex}(M \star + w \star)$. The converse implication uses the same argument with the remark that uncomplete runs correspond to diagrams which are erased by the operator $\not\downarrow$. Notice that we can reject sometimes but what matters is the existence of at least one non-deterministic run which reach q_a .
- (2) We show the two implications for the second statement.
 - (\Rightarrow) Assume that $w \notin \mathcal{L}(M)$ and that M terminates, meaning that M rejects w . Similarly to the proof of statement (1), we reach `reject` and never `accept`. We indeed have `[accept]` $\notin \mathcal{b} \not\downarrow \text{Ex}(M \star + w \star)$ and $\mathcal{b} \not\downarrow \text{Ex}(M \star + w \star) \neq \emptyset$.

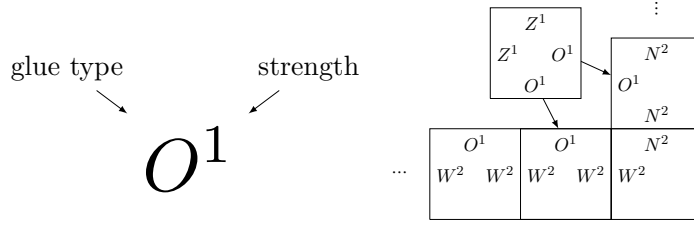


FIGURE 18. Illustration of an assembly in an aTAM. Assume we are at temperature $\tau = 2$. We can connect a new tile to an assembly because the glue types match and the sum of strengths involved is $1 + 1 \geq \tau$.

- (\Leftarrow) Assume that $[\text{accept}] \notin \flat \not\text{Ex}(M^\star + w^\star)$ and $\flat \not\text{Ex}(M^\star + w^\star) \neq \emptyset$. The only possibility is to have n occurrences of $[\text{reject}]$ in $\flat \not\text{Ex}(M^\star + w^\star)$. By correspondence between runs and diagram, we have a run reaching a rejecting state, hence $w \notin \mathcal{L}(M)$ and M terminates on w .

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 is erased 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 will reduce into the empty star $[\]$ which is erased by the operator \flat . Hence, it has no effect on the statement. \square

Although we have shown an example of *deterministic* Turing machine in Figure 17, 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.

Corollary 3.5 (Halting problem). *The problem of determining if $\flat \not\text{Ex}(M^\star + w^\star) \neq \emptyset$ is undecidable.*

Proof. By the simulation of non-deterministic Turing machines, a Turing machine M terminates if and only if $\flat \not\text{Ex}(M^\star + w^\star) \neq \emptyset$ since an infinite computation is either represented as the emptiness of the output (in case we only have diagrams impossible to saturate) or as the production of the empty star (e.g. $[-m(L, q, c, R), +m(L, q, x, R)]$) which is considered as a trivial loop. This is a known undecidable problem [30]. \square

3.3. Abstract tile assembly model. The abstract tile assembly [111, 87] (aTAM) is a tile system used in DNA computing [97] which extends Wang tiles (*cf.* Section 2.1). We present the idea without too much formality and refer to Lathrop et al. [77] for more details¹².

We define a *tile type* by $t_i = (g_w^i, g_e^i, g_s^i, g_n^i)$ for some i in a finite set of indexes I as objects intuitively corresponding to squares with each sides associated to a *glue type* $gl(g_d^i)$ for a direction $d \in \{\mathbf{w}, \mathbf{e}, \mathbf{s}, \mathbf{n}\}$ (for west, east, south and north) and a natural number $\text{str}(g_d^i)$ called its *strength*. The idea is that we have a global variable $\tau \in \mathbf{N}$ called the temperature

¹²However, we use a variant without seed assembly σ because it is more natural in our case.

and that a tile can be connected to other ones if the sum of strength involved in the connexion is at least τ . This phenomenon is known as *cooperation*.

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 $\alpha : \mathbf{Z}^2 \rightarrow T$ pasting tiles to the plane. It is associated to a (connected) grid graph G_α with vertices $V^{G_\alpha} := \text{dom}(\alpha)$ and there is an edge between two vertices representing tiles t_i, t_j with $i \neq j$ when $\text{gl}(g_d^i) = \text{gl}(g_{d'}^j)$ for $d = \text{op}(d')$ where op is the involution defined by $\text{op}(\mathbf{e}) = \mathbf{w}$ and $\text{op}(\mathbf{n}) = \mathbf{s}$.

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 from a given set of tile types). An example is given in Figure 18.

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. Tile types $t_i = (g_{\mathbf{w}}^i, g_{\mathbf{e}}^i, g_{\mathbf{s}}^i, g_{\mathbf{n}}^i)$ are encoded by a star t_i^\star :

$$\begin{aligned} & [-\overset{\bullet}{h}(\text{gl}(g_{\mathbf{w}}^i)(X), X, Y), -\overset{\bullet}{v}(\text{gl}(g_{\mathbf{s}}^i)(Y), X, Y), \\ & +\overset{\circ}{h}(\text{gl}(g_{\mathbf{e}}^i)(s(X)), s(X), Y), +\overset{\circ}{v}(\text{gl}(g_{\mathbf{n}}^i)(s(U)), X, s(Y))] \end{aligned}$$

where $\text{gl}(g)(X) := g(X) \cdot \overline{\text{str}(g)}$ for $\text{str}(g) \in \mathbf{N}$. 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 , the tiles cannot connect directly but has to use an intermediary star checking that the connexion is possible that we need to define.

The *environment constellation* for a temperature $\tau \in \mathbf{N} \setminus \{0\}$ is defined by $\Phi_{env}^\tau :=$

$$[+temp(\bar{\tau})] + \begin{bmatrix} +\overset{\bullet}{v}(g_1(X_1) \cdot N_1, X_1, Y_1), & -\overset{\circ}{v}(g_2(X_3) \cdot N_2, X_3, Y_3), \\ +\overset{\bullet}{h}(g_3(X_5) \cdot N_3, X_5, Y_5), & -\overset{\circ}{h}(g_4(X_7) \cdot N_4, X_7, Y_7), \\ -\overset{\circ}{v}(g_1(X_2) \cdot N_1, X_2, Y_2), & +\overset{\bullet}{v}(g_2(X_4) \cdot N_2, X_4, Y_4), \\ -\overset{\circ}{h}(g_3(X_6) \cdot N_3, X_6, Y_6), & +\overset{\bullet}{h}(g_4(X_8) \cdot N_4, X_8, Y_8), \\ -add(N_1, N_2, R_1), & -add(N_3, N_4, R_2), \\ -add(R_1, R_2, R), & -geq(R, T, \bar{1}), -temp(T) \end{bmatrix}$$

$$\begin{aligned} & +[-\overset{\bullet}{v}(g(X) \cdot \bar{0}, X, Y)] + [+ \overset{\circ}{v}(g(X) \cdot \bar{0}, X, Y)] + [-\overset{\bullet}{h}(g(X) \cdot \bar{0}, X, Y)] + [+ \overset{\circ}{h}(g(X) \cdot \bar{0}, X, Y)] \\ & + [+ \overset{\circ}{v}(g(X) \cdot \bar{0}, X, Y)] + [-\overset{\bullet}{v}(g(X) \cdot \bar{0}, X, Y)] + [+ \overset{\circ}{h}(g(X) \cdot \bar{0}, X, Y)] + [-\overset{\bullet}{h}(g(X) \cdot \bar{0}, X, Y)] \\ & + [+geq(\bar{0}, \bar{0}, \bar{1})] + [+geq(s(X), s(Y), R), -geq(X, Y, R)] + [+geq(s(X), \bar{0}, \bar{0})] + \\ & [+geq(\bar{0}, s(Y), \bar{0})] + [+add(\bar{0}, Y, Y)] + [-add(X, Y, Z), +add(s(X), Y, s(Z))] \end{aligned}$$

We define the translation of a set of tile types T as the constellation $T^\star := \sum_{t_i \in T} t_i^\star$.

Theorem 3.6 (Simulation of the aTAM). *Let $\mathcal{T} = (T, \tau)$ be a TAS. We have*

$$\text{CSatDiags}(T^\star + \Phi_{env}^\tau) \simeq \mathcal{A}_\square[\mathcal{T}].$$

Proof. It is sufficient to show that the computation of $\text{CSatDiags}(T^\star + \Phi_{env}^\tau)$ behaves like the construction of tilings in aTAM.

Direct connexions between tiles without using Φ_{env}^τ 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_{env}^τ as an intermediate for the connexion of two tile sides. We consider a tile $t_i \in \text{dom}(\alpha)$. We start with t_i^\star . Assume t_i can be connected to k other tiles in $\text{dom}(\alpha)$. They can only be connected through Φ_{env}^τ 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_{env}^τ . 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 using symbols *add* and *geq* are common logic programs, hence their correctness is assumed.

Since all $t_i \in \text{dom}(\alpha)$ 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 $\text{CSatDiags}(T^\star + \Phi_{env}^\tau)$. \square

3.4. Properties of constellations and their execution. In this section, few results of the execution are detailed. Firstly, 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 through Horn clauses [61, 107]) but also able to simulate the aTAM which is also Turing-complete [111, Section 3.2.5][112, Section 2].

Borrowing terminology from rewriting and the λ -calculus, we define the strong normalisation which corresponds to termination of the execution and the confluence asserting that it is possible to focus on a specific set of colours during the execution with no impact on the result, *i.e.* order is irrelevant.

Proposition 3.7 (Turing-completeness). *The stellar resolution is Turing-complete.*

Proof. Consequence of Theorem 3.4. Although we can encode Turing machines, the stellar resolution 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 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. \square

Definition 3.8 (Strong normalisation). A constellation Φ is *strongly normalising w.r.t.* a set of colours $A \subseteq C$ if and only if $\text{Ex}_A(\Phi)$ is a finite constellation (or equivalently that $\text{CSatDiags}_A(\Phi)$ is finite). We write $|\text{Ex}_A(\Phi)| < \infty$ (or $|\text{CSatDiags}_A(\Phi)| < \infty$) in this case. When $A = C$, we simply say that Φ is strongly normalising and omit to write A .

The shape of $\mathfrak{D}[\Phi; A]$ for a constellation Φ contains a lot of information about $\mathbf{Ex}_A(\Phi)$. By observing the shape of constellations in Section 3, we observe that only cycles make iteration possible and that several rays α -unifiable with the same single ray are linked to a non-determinism creating diagrams in parallel. However, duplication of stars can still occur without cycle nor non-determinism. Since the relationship between the structure of $\mathfrak{D}[\Phi; A]$ and the computational behaviour of $\mathbf{Ex}_A(\Phi)$ is a bit complex, we suggest few structural classes of constellations and establish theorems which will be useful to reason with constellations.

Definition 3.9 (Properties of constellation). A constellation Φ is:

- *exact* if all equations induced by the edges of $\mathfrak{D}[\Phi]$ are of the shape $t \stackrel{?}{=} t$;
- *acyclic* when $\mathfrak{D}[\Phi]$ is acyclic and otherwise it is *cyclic*;
- *connected* when $\mathfrak{D}[\Phi]$ is connected;
- *ambivalent* if the ray linking graph $\mathbf{RLG}\Phi$ (cf. Definition 2.16) has *ambivalent links* which are links between several vertices v_1, \dots, v_n (with $n > 1$) and a same vertex v . Otherwise, it is *monovalent*. In case it is ambivalent it is called:
 - *replicating* if we can only construct saturated diagram containing v_1, \dots, v_n , meaning that any duplications of stars remains in the same diagram;
 - *branching* or *non-deterministic* otherwise, meaning that duplications occur in “parallel universes”;
- *deterministic* when it is either monovalent or replicating.

All the definitions can be naturally parametrised with a set of colours $A \subseteq C$.

Examples 3.10. We illustrate the properties defined above.

- The constellation $\Phi_{\mathbf{N}}^{n+m}$ of Example 2.9 is connected, cyclic and 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 ambivalent.
- $[X, -c(X)] + [+c(f(Y))] + [+c(g(Y))]$ is acyclic, connected, and non-deterministic. The ray $-c(X)$ has two independent choices and leads to the formation of two diagrams.
- $[+a(1), +a(\mathbf{r})] + [+b(1), +b(\mathbf{r})] + [-a(X), -b(X)]$ is connected, cyclic and replicating. Two choices are possible for the negative rays but all the stars can appear in the same diagram by duplicating $[-a(X), -b(X)]$ and connecting the 1 (*resp.* \mathbf{r}) together.
- The disjoint union of two above constellations gives a disconnected constellation.

Lemma 3.11 (Termination of acyclic constellations). *If a constellation Φ is acyclic w.r.t. $A \subseteq C$ then $|\mathbf{CSatDiags}_A(\Phi)| < \infty$.*

Proof. Assume $\mathfrak{D}[\Phi; A]$ is acyclic and consider a diagram $\delta : D_\delta \rightarrow \mathfrak{D}[\Phi; A]$. It must be injective on the vertices, *i.e.* for $v, v' \in D_\delta$ if $v \neq v'$ then $\delta(v) \neq \delta(v')$, meaning that v and v' do not correspond to duplications of some star in $\mathfrak{D}[\Phi; A]$. Hence, the vertices of V^{D_δ} are uniquely taken from $V^{\mathfrak{D}[\Phi; A]}$ 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 $\mathbf{CSatDiags}_A(\Phi)$ is finite. \square

Lemma 3.12 (Uniqueness). *Let Φ be a constellation and $A \subseteq C$ a set of colours. If Φ is acyclic, connected and deterministic constellations w.r.t. A then $|\mathbf{SatDiags}_A(\Phi)| = 1$ (and $|\mathbf{Ex}_A(\Phi)| \leq 1$).*

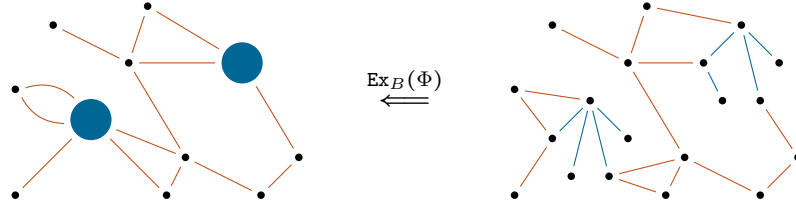


FIGURE 19. Partial execution acts as a partial diagram contraction, which is only possible when Φ and Φ' do not act on a same variable. The “blow-up” obtained by inverting the execution preserves the connexions between rays.

Proof. Since Φ is acyclic, by Lemma 3.11, we have $|\text{CSatDiags}_A(\Phi)| < \infty$ and there is no loop in $\mathfrak{D}[\Phi; A]$ and since it is connected, it has the shape of a tree. Because $\mathfrak{D}[\Phi; A]$ is a tree, a saturated diagram for a connected constellation must be maximal and include all vertices and rays of $\mathfrak{D}[\Phi; A]$. A deterministic constellation is either monovalent or replicating. If it is monovalent then there is at most one choice of connexion for a ray. By choosing all these unique connexions we obtain a unique diagram. If it replicating, there is few duplications of stars but the whole forms a unique diagram as well (it is only finite duplication and the only way to duplicate because there is no loop). Hence, we have $|\text{SatDiags}_A(\Phi)| = 1$. Depending on if this diagram obtained in both cases is correct or not there is at most one correct saturated diagram. Hence $|\text{CSatDiags}_A(\Phi)| = 1$ and $|\text{Ex}(\Phi)| \leq 1$. \square

Lemma 3.13 (Exactness). *Let Φ be a constellation which is exact w.r.t. a set of colours $A \subseteq C$. We have $\text{Diags}_A(\Phi) = \text{CSatDiags}_A(\Phi)$.*

Proof. Let Φ be an exact constellation. By the definition of diagram (cf. Definition 2.16), all equations $t \stackrel{?}{=} u$ induced by diagrams are renamings of equations induced by the edges $e \in V^{\mathfrak{D}[\Phi; A]}$, i.e. they are of the shape $\alpha_v t' \stackrel{?}{=} \alpha_{v'} u'$ for $v, v' \in V^{G_\delta}$. Assume $t' = u'$. For any renamings α_1, α_2 we have $\alpha_1 t$ which is α -unifiable with $\alpha_2 t$. We remark that $\text{solution}(\mathcal{P}(\delta))$ must always be a renaming. This makes δ correct. Hence, we have $\text{Diags}_A(\Phi) \subseteq \text{CSatDiags}_A(\Phi)$. By definition, we also have $\text{CSatDiags}_A(\Phi) \subseteq \text{Diags}_A(\Phi)$. \square

Lemma 3.14 (Independence of connected components). *Let Φ be a constellation. If $\mathfrak{D}[\Phi; A]$ has n connected component corresponding to the subconstellations $\Phi_1, \dots, \Phi_n \subseteq \Phi$, then $\text{Ex}_A(\Phi) = \bigcup_{i=1}^n \text{Ex}_A(\Phi_i)$.*

Proof. Each connected component $G_i \subseteq \mathfrak{D}[\Phi; A]$ constitutes a constellation Φ_i . When we execute Φ , we form diagrams following the connexions of $\mathfrak{D}[\Phi; A]$. Since a diagram has to be connected and that no edge link the G_i in $\mathfrak{D}[\Phi; A]$, we necessarily have $\text{CSatDiags}_A(\Phi) = \text{CSatDiags}_A(\Phi_1) \cup \dots \cup \text{CSatDiags}_A(\Phi_n)$, hence $\text{Ex}_A(\Phi) = \bigcup_{i=1}^n \text{Ex}_A(\Phi_i)$. \square

An important result is the possibility of executing only some colours on some constellations first then the others without any effect on the normal form, that is $\text{Ex}_{A \cup B}(\text{Ex}_B(\Phi) \uplus \Phi') = \text{Ex}_{A \cup B}(\Phi \uplus \Phi')$ for some set of colours A and B . However, this is not valid in general as presented in Figure 20. The problem is that stars from two disjoint constellations can alter a same variable and a partial execution will erase some potential connexions which were present. This problem is reminiscent of the idea of *mutual exclusion* in concurrent programming [33]: the constellations Φ and Φ' can modify a same variable x but when executing Φ and accessing x , we may lose an access to x which is still required by Φ' .

$$\Phi = [X, +c(X)] + [-c(1 \cdot X)] \qquad [-c(\mathbf{r} \cdot X)] = \Phi'$$

FIGURE 20. Counter-example for partial pre-execution. We have $\text{Ex}_{\{c\}}(\Phi) = [1 \cdot X]$ and $\text{Ex}_{\{c\}}(\text{Ex}_{\{c\}}(\Phi) \uplus \Phi') = [-c(\mathbf{r} \cdot X)] + [1 \cdot X]$, but $\text{Ex}_{\{c\}}(\Phi \uplus \Phi') = [1 \cdot X] + [\mathbf{r} \cdot X]$ which is different. Notice that both $[-c(1 \cdot X)]$ and $[-c(\mathbf{r} \cdot X)]$ needs $[X, +c(X)]$ but when executing Φ , Φ' cannot be connected to it anymore.

This property of partial pre-execution is necessary in order to obtain a result of confluence and associativity of execution, hence a model of linear logic. We need to design a precondition for which these properties are valid and from which it is possible to express logic.

There are several possible choices. A simple choice is to reason on the accessibility of variables in a dependency graph. We do not want a variable to be accessible from two different constellations such that one is pre-executed before the other. For instance, in Figure 20, the variable X of $[X, +c(X)]$ is accessible both from Φ and Φ' .

Definition 3.15 (Shared variables). Variables are written X_j^i where i is an index of star and j an index of ray within that star. We write $\text{acc}_{\Phi_\omega}^A(X_j^i, \Phi)$ for a constellation $\Phi \subseteq \Phi_\omega$ and a set of colours $A \subseteq C$ when there is an edge path e_1, \dots, e_n in $\mathfrak{D}[\Phi_\omega; A]$ from some $\phi \in \Phi$ to $\Phi_\omega[i]$ such that $\ell(e_n) = r \bowtie r'$ with $r' \in \Phi_\omega[i]$ and $X \in \text{vars}(r')$.

We define the *set of variables shared by two constellations* Φ_1 and Φ_2 *w.r.t.* a set of colours $A \subseteq C$ as the set $\Phi_1 \cap_A \Phi_2$ such that we have $X_j^i \in \Phi_1 \cap_A \Phi_2$ when $\text{acc}_{\Phi_1 \uplus \Phi_2}^A(X_j^i, \Phi_1)$ and $\text{acc}_{\Phi_1 \uplus \Phi_2}^A(X_j^i, \Phi_2)$. We generalise the notation to the set of variables shared by n constellations with the associative notation $\Phi_1 \cap_A \dots \cap_A \Phi_n := \bigcap_{1 \leq i, j \leq n} \Phi_i \cap_A \Phi_j$.

Proposition 3.16 (Commutativity and associativity of shared variables). *For any constellations Φ_1, Φ_2 and Φ_3 and a set of colours $A \subseteq C$, we have $\Phi_1 \cap_A \Phi_2 = \Phi_2 \cap_A \Phi_1$ and if σ is any permutation on $\{1, 2, 3\}$, we have $\Phi_1 \cap_A \Phi_2 \cap_A \Phi_3 = \Phi_{\sigma(1)} \cap_A \Phi_{\sigma(2)} \cap_A \Phi_{\sigma(3)}$.*

Proof. We obviously have $X \in \Phi_1 \cap_A \Phi_2$ if and only if $X \in \Phi_2 \cap_A \Phi_1$ because in both cases, X is still accessible from both Φ_1 and Φ_2 . The same reasoning holds for the associativity. \square

Lemma 3.17 (Partial pre-execution). *Let Φ and Φ' be constellations and $A, B \subseteq C$ be sets of colours such that $\Phi \cap_{A \cup B} \Phi' = \emptyset$. We have $\text{Ex}_{A \cup B}(\text{Ex}_B(\Phi) \uplus \Phi') = \text{Ex}_{A \cup B}(\Phi \uplus \Phi')$.*

Proof. Assume we have a diagram $\delta_i^{A \cup B} \in \text{CSatDiags}_{A \cup B}(\text{Ex}_B(\Phi) \uplus \Phi')$. It is constructed by connecting the stars ϕ'_j of Φ' with stars ϕ_k^B of $\text{Ex}_B(\Phi)$. These stars ϕ_k^B of $\text{Ex}_B(\Phi)$ come from diagrams $\delta_k^B \in \text{CSatDiags}_B(\Phi)$. We can perform a “blow-up“ (*cf.* Figure 19) on $\text{Ex}_B(\Phi)$ by replacing the stars ϕ_k^B by their corresponding diagram δ_k^B . In some sense, we reversed the execution from $\text{Ex}_B(\Phi)$ to $\text{CSatDiags}_B(\Phi)$. This is only possible because we have $\Phi \cap_{A \cup B} \Phi' = \emptyset$, meaning that the stars of Φ and of Φ' cannot interfere by acting on a same variable in a same star. Hence, the execution of Φ makes diagrams in which variables are independent of the ones of Φ' . Otherwise, some connexions could disappear (as in Figure 20) and we would not preserve all connexions allowing this inversion of execution. We obtain diagrams $\varphi(\delta_k^B)$ corresponding to diagrams δ_k^B extended with stars of Φ' in exactly the same

way as how ϕ_k^B can be connected with Φ' . We have $\varphi(\delta_k^B) \in \text{CSatDiags}_{A \cup B}(\Phi \uplus \Phi')$ since it connects stars of both Φ and Φ' .

It remains to show that φ is invertible so that we have an isomorphism between $\text{CSatDiags}_{A \cup B}(\text{Ex}_B(\Phi) \uplus \Phi')$ and $\text{CSatDiags}_{A \cup B}(\Phi \uplus \Phi')$. Assume we have

$$\delta^{A \cup B} \in \text{CSatDiags}_{A \cup B}(\Phi \uplus \Phi').$$

We would like to define $\varphi^{-1}(\delta^{A \cup B})$. By the confluence of fusion (which is a consequence of the correspondence between fusion and actualisation, *cf.* Theorem 2.25), we can contract first the stars coming from Φ using colours in B and we obtain a diagram $\varphi^{-1}(\delta^{A \cup B})$. We have $\varphi^{-1}(\delta^{A \cup B}) \in \text{CSatDiags}_{A \cup B}(\text{Ex}_B(\Phi) \uplus \Phi')$.

It is obvious that $\varphi(\varphi^{-1}(\delta)) = \delta$ and $\varphi^{-1}(\varphi(\delta)) = \delta$ because φ is defined from the diagrams from which the stars of a normal form come from (star expansion), which is exactly the reverse operation of contracting stars by fusion. \square

Theorem 3.18 (Confluence). *For any constellation Φ , and $A, B \subseteq C$ two disjoint sets of colours such that $\Phi \cap_{A \cup B} \Phi' = \emptyset$, we have $\text{Ex}_B(\text{Ex}_A(\Phi)) = \text{Ex}_{A \cup B}(\Phi) = \text{Ex}_A(\text{Ex}_B(\Phi))$.*

Proof. By Lemma 3.17 with $\Phi' := \emptyset$ (in this case we trivially have $\Phi \cap_{A \cup B} \emptyset = \emptyset$ which is the required precondition) we have $\text{Ex}_{A \cup B}(\text{Ex}_B(\Phi)) = \text{Ex}_{A \cup B}(\Phi)$. Since $\text{Ex}_B(\Phi)$ already uses all colours in B , we have $\text{Ex}_{A \cup B}(\text{Ex}_B(\Phi)) = \text{Ex}_A(\text{Ex}_B(\Phi))$, hence $\text{Ex}_A(\text{Ex}_B(\Phi)) = \text{Ex}_{A \cup B}(\Phi)$. Since $A \cup B = B \cup A$, we also have $\text{Ex}_{A \cup B}(\Phi) = \text{Ex}_{B \cup A}(\Phi)$. By using again Lemma 3.17, we finally obtain $\text{Ex}_{B \cup A}(\Phi) = \text{Ex}_B(\text{Ex}_A(\Phi))$. \square

Remark 3.19. In Girard's first paper on Transcendental Syntax [55, Section 2.4], the constellation $\Phi = [+a(X), -a(X), +b(X)]$ is mentioned as a counter-example for the confluence of Ex (which only considers tree-shaped diagrams). Here, we have $\text{Ex}_{\{a\}}(\text{Ex}_{\{b\}}(\Phi)) = \text{Ex}_{\{b\}}(\text{Ex}_{\{a\}}(\Phi)) = \emptyset$ (because no saturated diagram on a nor on b can be constructed). Our understanding of Girard's failure comes from his limitation to strongly normalising constellations, so that $\text{Ex}_{\{a\}}(\Phi)$ was not defined because of the cyclic dependence between $+a(X)$ and $-a(X)$.

Also remark that Ex is analogous to the computation of all answers we can infer from a logic program, meaning that all possible paths of computation are considered, hence naturally leading to confluence.

Before ending our computational journey, let us stress (again) the fact that there are several differences between the stellar resolution and approaches in logic programming (although identical objects are used). Our approach is indeed a liberalised variant of first-order resolution but we are not aware of any similar uses of resolution. We suggest some comparisons with other approaches in the literature:

Original first-order resolution: It is almost identical. We add unpolarised rays which cannot be connected (it can still be simulated in resolution by using special unused predicates). In resolution, we are usually interested in the reachability of the empty clause (\square 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. Usual resolution is limited to tree derivations (corresponding to tree-like diagrams) whereas stellar resolution allows cyclic diagrams in order to interpret tiling-based computation. There are graph-based models [105, 72, 35] which are very similar to stellar resolution but they are still different for the reasons mentioned above.

Horn clauses and logic programming: By logic programming, we mean that we are interested in answering a query represented by a first-order atom (such as in PROLOG for instance). In order to answer the query, logic programming use a backward reasoning by going up from the unique conclusion to the premises. The 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 output \rightarrow inputs is imposed in the inference.

Stable model semantics: There are several languages based on stable model semantics such as disjunctive logic programming [83, 79] itself based on a subset of PROLOG called DATALOG. The notion of stable model is also the basis of answer set programming (ASP) [38, 36]. 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.

4. EMERGENCE OF PROOFS

In order to reconstruct logic, it is natural to start from linear logic [41] which decomposes both classical and intuitionistic logic. We choose to work with Girard’s representation of proofs called “proof-nets”¹³. We begin by defining the fragment of linear logic we work with. Useful definitions about hypergraphs are recalled in Appendix B.

4.1. Multiplicative proofs. Multiplicative linear logic (MLL) is a fragment of linear logic [41] restricted to the tensor \otimes and par \wp connectives which are respectively a sort of conjunction and disjunction. The set \mathcal{F}_{MLL} of MLL formulas is defined by the grammar of Figure 21a. Linear negation $(\cdot)^\perp$ is extended to formulas by involution and De Morgan laws: $X_i^{\perp\perp} = X_i$, $(A \otimes B)^\perp = A^\perp \wp B^\perp$, and $(A \wp B)^\perp = A^\perp \otimes B^\perp$.

MLL proofs can be written in the traditional sequent calculus fashion by constructing trees using the set of rules shown in Figure 21b. These rules use sequents $\vdash \Gamma$ stating the provability of a set of formulas $\Gamma \subseteq \mathcal{F}_{\text{MLL}}$. Instead of the MLL sequent calculus, we choose to work with Girard’s proof-nets, a “parallel” syntax for proofs akin to Gentzen’s natural deduction which captures the essence of proofs by forgetting the order of rules. In order to define proof-nets, we first define *proof-structures* which are purely computational and structural objects with no logical meaning. They represent skeletons for proofs. In the same spirit as Girard’s ludics [48], “only location matters” at this point. The idea is that when considering the structure of proofs, formulas are nothing more than decorative labels which can be forgotten. In this syntax, we consider directed hypergraphs constructed with the hyperedges of Figure 21c.

Definition 4.1 (Proof-structure, Figure 22). A *proof-structure* is defined by a tuple $\mathcal{S} = (V, E, \text{in}, \text{out}, \ell_E)$ where $(V, E, \text{in}, \text{out})$ is a directed hypergraph and $\ell_E : E \rightarrow \{\otimes, \wp, \text{ax}, \text{cut}\}$ is a labelling map on hyperedges. A proof-structure is subject to these additional constraints:

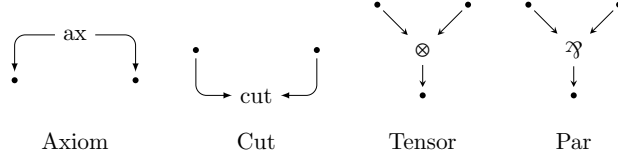
¹³Our definitions differ from the usual definitions of the literature but are more convenient for the results presented in this paper.

$$A, B = X_i \mid X_i^\perp \mid A \otimes B \mid A \wp B \quad i \in \mathbf{N} \quad (\mathcal{F}_{\text{MLL}})$$

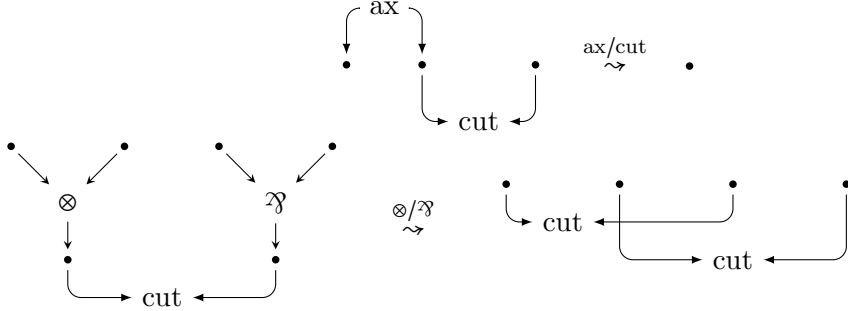
(A) MLL Formulas.

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

(B) MLL sequent calculus rules.



(C) Links/constructors of proof-structures as hyperedges.



(D) Cut-elimination reductions. The ax/cut case is a graph contraction and \otimes/\wp is a rewiring.

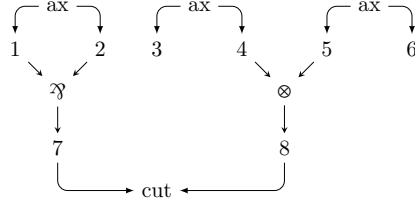
FIGURE 21. Syntax of Multiplicative Linear Logic (MLL).

- ▷ the hyperedges satisfy the arities and labelling constraints shown in Figure 21c;
- ▷ 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 \wp hyperedge with the conclusion of a \otimes hyperedge, or
 - two atoms.

Convention 4.2 (Left and right sources). For practical purposes, the sources of hyperedges are ordered, and we will talk about the “left” and “right” sources since there are never more than two; illustrations in Figure 21c implicitly represent the left (*resp.* right) source on the left (*resp.* right).

Notation 4.3 (Axioms and cuts). Let \mathcal{S} be a proof-structure. We write $\text{Ax}(\mathcal{S})$ (*resp.* $\text{Cuts}(\mathcal{S})$) the set of axioms (*resp.* cut) hyperedges in \mathcal{S} . Given $e \in \text{Ax}(\mathcal{S})$ ($e \in \text{Cuts}(\mathcal{S})$), we write \vec{e} and \bar{e} the left and right conclusion (*resp.* sources) of e respectively.

Notation 4.4 (Conclusions and atoms). The *conclusions* of \mathcal{S} are defined by the set $\text{Concl}(\mathcal{S}) = \{v \in V \mid \text{there is no } e \in E \text{ such that } v \in \text{in}(e)\}$. Similarly, the *atoms* of \mathcal{S} are defined by the set $\text{Atoms}(\mathcal{S}) = \{v \in V \mid \exists e \in \text{Ax}(\mathcal{S}) \text{ such that } v \in \text{out}(e)\}$. They are conclusions of axiom hyperedges.

FIGURE 22. Example of unlabelled proof-structure with vertices in \mathbf{N} .

$$\frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta} \text{ mix} \qquad \frac{\frac{}{\vdash X_1, X_1^\perp} \text{ ax} \quad \frac{}{\vdash X_2, X_2^\perp} \text{ ax}}{\vdash X_1^\perp, X_2^\perp, X_1, X_2} \text{ mix}$$

FIGURE 23. The MIX rule and a of sequent calculus proof of $X_1 \otimes X_2 \multimap X_1 \wp X_2$ using the MIX rule.

The hyperedges of Figure 21c are the elementary bricks for proof-structures. Notice that the \otimes and \wp hyperedge are structurally identical and that their label is irrelevant. As a first step, we will consider them identical. It is only later, in Section 4.3, that these two constructions will be distinguished by the logical meaning we associate to them.

The cut-elimination procedure on proof-structures (corresponding to program execution) is defined as a graph-rewriting system on proof-structures, defined by the two rewriting rules in Figure 21d.

There exists a remarkable extension of MLL with a rule called MIX (*cf.* Figure 23), initially studied by Fleury and Rétoré [37]. This rule corresponds to the axiom scheme $A \otimes B \multimap A \wp B$ 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.

We now would like to define the underlying proof-structure of an MLL+MIX sequent calculus proof. In order to do so, we define a labelling of the vertices of proof-structures by formulas in order to make proof-structures look like actual proofs. By doing so, we already give a little bit of meaning to the purely computational proof-structures but which is only superficial for the moment.

Definition 4.5 (Labelled proof-structure). A *labelled proof-structure* is a tuple

$$\mathcal{S} = (V, E, \text{in}, \text{out}, \ell_V, \ell_E)$$

where $(V, E, \text{in}, \text{out}, \ell_E)$ is a proof-structure and $\ell_V : V \rightarrow \mathcal{F}_{\text{MLL}}$ is a function labelling vertices of V by formulas.

We write $\vdash \mathcal{S} : \Gamma$ for a set of formula $\Gamma := \{\ell_V(v) \mid v \in \text{Concl}(\mathcal{S})\}$ in order to specify the formulas associated to the conclusions of \mathcal{S} .

In Figure 24, we define a translation $\llbracket \cdot \rrbracket$ 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 topological properties and which are considered “correct”. This is discussed

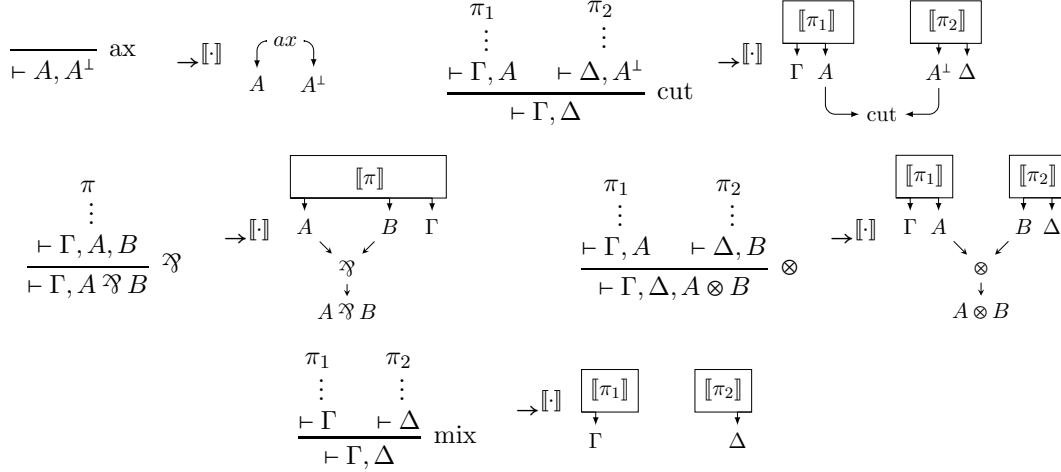


FIGURE 24. Translation of MLL+MIX sequent calculus proofs into labelled proof-structures.

in Section 4.3 but for the time being, we give a preliminary definition of proof-net, the proof-structures coming from sequent calculus proofs.

Also notice that 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 [41, Chapter 4].

Definition 4.6 (MLL and MLL+MIX proof-nets). An MLL (*resp.* MLL+MIX) *proof-net* is a proof-structure \mathcal{S} such that there exists an MLL (*resp.* MLL+MIX) sequent calculus proof π such that $\mathcal{S} = \llbracket \pi \rrbracket$.

In this paper, we show that both MLL and MLL+MIX can be interpreted in the stellar resolution.

4.2. Simulation of cut-elimination. Early investigations on the cut-elimination [44], simplified with Seiller’s interaction graphs [98], show that cut-elimination can be considered much simpler than the standard graph rewriting of Figure 21d. The \otimes/\wp cut-elimination rule pushes cuts to the top of the proof-structure (axioms) and the ax/cut rule identifies some atoms by contraction. It shows that we can see a proof-structure as a connexion between a permutation of atoms representing axioms and a partial permutation on atoms representing cuts (*cf.* Figure 25). The cut-elimination procedure is then seen as a computation of maximal alternating paths between the graph of these two permutations or equivalently as the complete edge contraction of a bipartite graph.

When considering the computational content of proofs, the connectives \otimes and \wp are then irrelevant since no reference to logic exists at this point and that the \otimes/\wp cut-elimination is a simple rewiring. For that reason, the simulation of cut-elimination in the stellar resolution only deals with the translation of axioms and cuts as binary stars with rays representing the *address* of atoms. The interpretation is the same for both MLL and MLL+MIX since they have the same cut-elimination.

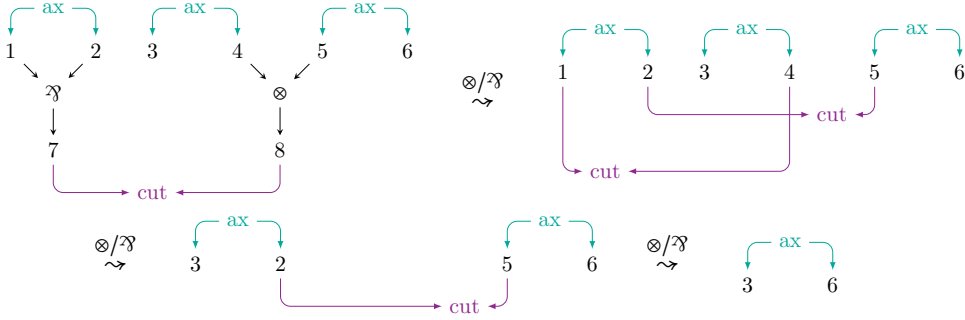


FIGURE 25. Cut-elimination for the proof-structure of Figure 22 represented as the juxtaposition of two partial permutations or graphs on atoms as suggested in the GoI.

In order to encode proof-structures, we fix a *basis of representation* \mathcal{B} with variables $V = \{X\}$, colours $C = \{+c, -c, +t, -t\}$, function symbols $F = \{c, t, \mathbf{l}, \mathbf{r}, \mathbf{g}, \cdot, p_A, q_A\}$ for $A \in \mathcal{F}_{\text{MLL}}$ such that c, t, p_A and q_A are unary, \cdot is binary and the symbols \mathbf{l}, \mathbf{r} and \mathbf{g} are constants. We define $\text{op}(+c) = -c$, $\text{op}(+t) = -t$, $[\pm c] = c$ and $[\pm t] = t$ for $\pm \in \{+, -\}$.

Similarly to unlabelled proof-structures, constellations are purely locative: only the locations appearing in a proof-structure \mathcal{S} are translated, without regard to labels. We would like to associate a unique address in $\text{Terms}(\mathbb{B})$ to the atoms $v \in \text{Atoms}(\mathcal{S})$ of a proof-structure \mathcal{S} . The address of v will be a term $p_v(t)$ where t is a path encoded as a sequence of \mathbf{l} (left) and \mathbf{r} (right) symbols representing the direction to follow in \mathcal{S} to get from the conclusion $v' \in \text{Concl}(\mathcal{S})$ to the atom v .

For convenience, we suggest an inductive definition of proof-structures based on their underlying hypergraph.

Remark 4.7 (Inductive definition of proof-structures). A proof-structure with only one hyperedge is necessarily an axiom with two conclusions, written $\text{Ax}_{u,v}$. 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 , \wp , or cut hyperedge on two of its conclusions u (left) and v (right). This is written $\text{Tens}^{u,v}(\mathcal{S}')$, $\text{Par}^{u,v}(\mathcal{S}')$ and $\text{Cut}^{u,v}(\mathcal{S}')$.

We use this inductive definition to define the *address* of atoms in a proof-structure.

Definition 4.8 (Vertex above another one). A vertex v is *above* another vertex u , written in a proof-structure if there exists a directed path (*cf.* Appendix B) from v to u going through only \otimes and \wp hyperedges.

Definition 4.9 (Address of an atom). We define the *path address* $\text{pAddr}_{\mathcal{S}}(v, X)$ to an atom v in a proof-structure \mathcal{S} *w.r.t.* the variable X inductively (*cf.* Remark 4.7):

- ▷ $\text{pAddr}_{\mathcal{S}}(v, X) = X$ when $\mathcal{S} = \text{Ax}_{v,*}$ or $\mathcal{S} = \text{Ax}_{*,v}$;
- ▷ $\text{pAddr}_{\mathcal{S}}(v, X) = \text{pAddr}_{\mathcal{S}_i}(v, X)$ if $\mathcal{S} = \mathcal{S}_1 \uplus \mathcal{S}_2$ and $v \in V^{\mathcal{S}_i}$;
- ▷ $\text{pAddr}_{\mathcal{S}}(v, X) = \mathbf{l} \cdot \text{pAddr}_{\mathcal{S}'}(v, X)$ if $\mathcal{S} = \text{Par}^{v,*}(\mathcal{S}')$ or $\mathcal{S} = \text{Tens}^{v,*}(\mathcal{S}')$;
- ▷ $\text{pAddr}_{\mathcal{S}}(v, X) = \mathbf{r} \cdot \text{pAddr}_{\mathcal{S}'}(v, X)$ if $\mathcal{S} = \text{Par}^{*,v}(\mathcal{S}')$ or $\mathcal{S} = \text{Tens}^{*,v}(\mathcal{S}')$ and $\text{pAddr}_{\mathcal{S}}(v, X) = \text{pAddr}_{\mathcal{S}'}(v, X)$ otherwise.

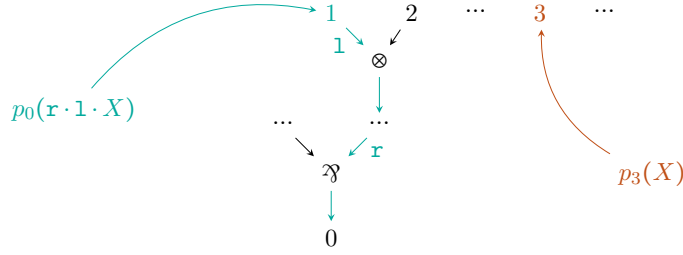


FIGURE 26. Addressing of the atoms 1 and 3 in a proof-structure relatively to the conclusion they come from.

The path address to v is uniquely defined *w.r.t.* to a conclusion $c \in \text{Concl}(\mathcal{S}')$ where \mathcal{S}' is \mathcal{S} without cuts, *i.e.* $E^{\mathcal{S}'} = E^{\mathcal{S}} \setminus \text{Cuts}(\mathcal{S})$ and the rest of \mathcal{S} is defined as in \mathcal{S}' .

The *address* of v is then defined as the term $\text{addr}_{\mathcal{S}}(v, X) := p_c(\text{pAddr}_{\mathcal{S}}(v, X))$.

Example 4.10. Figure 26 illustrates the idea of addressing of atoms. The address of the atom 1 in Figure 22 is $p_7(1 \cdot X)$ because it is reachable from the conclusion 7 by going to the left premise and the address of the atom 3 is $p_3(X)$ because it is directly reachable.

Definition 4.11 (Set of addresses). We define $\text{Addr}_x(\mathcal{S})$ as the *set of addresses* of the shape $\text{addr}_{\mathcal{S}}(-, X)$, *i.e.* the countable set of all terms of the form $p_c(f_1 \cdot \dots \cdot f_n \cdot X)$ where c ranges over conclusions of \mathcal{S} and $f_i \in \{1, \mathbf{r}\}$.

Notation 4.12 (Unary colour). We will often write $+c.t$ instead of $+c(t)$ for rays having a unary colour as prefix.

Definition 4.13 (Colour change). Let $\mathbb{B} = (V, C, F, \text{op}, [\cdot])$ and $\mathbb{B}' = (V', C', F', \text{op}', [\cdot]')$ be two signatures. A *colour change* from \mathbb{B} to \mathbb{B}' is a total injective function $\mu : C \rightarrow C'$.

A colour change of a constellation Φ is a constellation $\mu(\Phi)$ over \mathbb{B}' where the function $\mu : \text{colours}(\Phi) \rightarrow C'$ is a colour change such that $\text{colours}(\Phi)$ is the set of colours in Φ and $\mu(\Phi)$ is defined by replacing the colours c by $\mu(c)$ in Φ .

Definition 4.14 (Colour shift). A *colour shift* from a signature $\mathbb{B} = (V, C, F, \text{op}, [\cdot])$ to $\mathbb{B}' = (V', C', F', \text{op}', [\cdot]')$ is a colour change μ from \mathbb{B} to \mathbb{B}' such that $[\mu(c)] = \text{op}([\mu(\text{op}(c))])$.

Proposition 4.15. *Let Φ be a constellation, μ a colour shift and $A \subseteq C$ a set of colours. We have $\mathfrak{D}[\Phi; A] \simeq \mathfrak{D}[\mu(\Phi); A]$.*

Proof. We show that the matchability is preserved. Let r and r' two dual rays. By induction on r . If r is a variable, then even if we change the symbols of r' , x is still α -unifiable with r' . If r is $f(r_1, \dots, r_n)$, then we must have $r' := f(r'_1, \dots, r'_n)$. If f is a colour for which μ is defined the two f of r and r' become $\mu(f)$ which preserves the α -unifiability. We conclude with the induction hypothesis for the rays r_i and r'_i . Conversely, if r and r' are not matchable, they must be two terms with a mismatch of function symbol between f and g such that $f \neq g$. In this case, since μ is injective and total, we cannot have $\mu(f) = \mu(g)$. Hence, the non α -unifiability is preserved as well. \square

Definition 4.16 (Full colouration of constellation). The *full colouration* $c.\Phi$ of a constellation Φ is defined by a constellation $\mu(\Phi)$ where μ is a colour shift such that $\mu(x) = c$.

Definition 4.17 (Translation of the computational content of proof). The *vehicle* and the *cuts* of a proof-structure \mathcal{S} are respectively defined by the following constellations:

$$\Phi_{\mathcal{S}}^{\text{ax}} := \sum_{e \in \text{Ax}(\mathcal{S})} [\text{addr}_{\mathcal{S}}(\vec{e}, X), \text{addr}_{\mathcal{S}}(\vec{e}, X)], \quad \Phi_{\mathcal{S}}^{\text{cut}} := \sum_{e \in \text{Cut}(\mathcal{S})} [p_e^-(X), p_e^-(X)].$$

We define the *computational content* of \mathcal{S} as the constellation $\Phi_{\mathcal{S}}^{\text{comp}} := +c.\Phi_{\mathcal{S}}^{\text{ax}} \uplus -c.\Phi_{\mathcal{S}}^{\text{cut}}$.

We now show that the execution $\text{Ex}(\Phi_{\mathcal{S}}^{\text{comp}})$, which can be understood as an interaction between $+c.\Phi_{\mathcal{S}}^{\text{ax}}$ and $-c.\Phi_{\mathcal{S}}^{\text{cut}}$, has the same behaviour as the cut-elimination on \mathcal{S} . This shows that cut-elimination for proof-structures can be simulated in the stellar resolution.

Lemma 4.18 (Simulation of cut-elimination). *Let \mathcal{R} be a proof-structure such that $\mathcal{R} \rightsquigarrow \mathcal{S}$. We have $\text{Ex}(\Phi_{\mathcal{R}}^{\text{comp}}) = \text{Ex}(\Phi_{\mathcal{S}}^{\text{comp}})$.*

Proof. For convenience, we will sometimes use the notations of Definition 4.7. By case analysis on a reducible cut selected in \mathcal{R} .

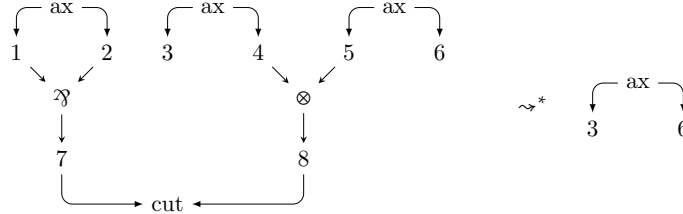
- ▶ Assume we have an ax/cut cut on two vertices v_1 and v_2 with v_1 conclusions of an axiom Ax_{v_0, v_1} and $v_0 \neq v_2$ such that v_2 is conclusion of some $e \in E^{\mathcal{R}_{\Gamma}}$ (i.e. $v_2 \in \text{out}(e)$) where \mathcal{R}_{Γ} is the rest of the proof-structure. Then we have an ax/cut cut-elimination which removes v_1 and v_2 then updates the targets $\text{out}(e)$ of e with $\text{out}(e) := \text{out}(e) \setminus \{v_2\} \cup \{v_0\}$ (v_2 and v_0 now refer to the same location by replacing v_2 by v_0 in the targets of e). This produces a new proof-structure \mathcal{S} where the only remaining cuts are the ones of \mathcal{R}_{Γ} . We have $\Phi_{\mathcal{R}}^{\text{comp}} = \Phi_{\mathcal{R}_{\Gamma}}^{\text{comp}} + [-c.p_{v_1}(X), -c.p_{v_2}(X)]$. By the definition of vehicle (cf. Definition 4.17) and addresses (cf. Definition 4.9), we necessarily have a star $[\varphi(+c.p_{v_0}(t)), +c.p_{v_1}(X)] \in \Phi_{\mathcal{R}_{\Gamma}}^{\text{ax}}$ for some t (with a colouring φ depending on whether the ray is related to a cut or not and $t = X$ if v_0 is not source of another hyperedge) and $[+c.p_{v_2}(X), r] \in \Phi_{\mathcal{R}_{\Gamma}}^{\text{ax}}$ for some ray r . By fusion, the cut star will merge these two stars and form $\phi := [\varphi(+c.p_{v_0}(t)), r]$. We finally obtain the constellation $\Phi_{\Gamma}^{\text{comp}} + \phi$. The translation of \mathcal{S} coincides with the previous constellation obtained by fusion because what was connected to v_2 (r) by an axiom is now connected to v_0 in ϕ , hence $\Phi_{\mathcal{S}}^{\text{comp}} = \Phi_{\Gamma}^{\text{comp}} + \phi$. It is indeed a relocation of atom. By partial pre-execution (cf. Lemma 3.17), we can focus on this step of fusion and preserve the correct diagrams without adding more diagrams because there is only one choice of connexion (this subgraph of $\mathcal{D}[\Phi_{\mathcal{R}}^{\text{comp}}]$ corresponds to a deterministic constellation). Therefore, $\text{Ex}(\Phi_{\mathcal{R}}^{\text{comp}}) = \text{Ex}(\Phi_{\mathcal{R}_{\Gamma}}^{\text{comp}} + [-c.p_{v_1}(X), -c.p_{v_2}(X)]) = \text{Ex}(\Phi_{\Gamma}^{\text{comp}} + \phi) = \text{Ex}(\Phi_{\mathcal{S}}^{\text{comp}})$.
- ▶ Assume we have an \otimes/\wp cut between two vertices: v_1 conclusion of a \wp hyperedge of inputs \vec{v}_1, \vec{v}_1 and v_2 conclusion of a \otimes hyperedge of inputs \vec{v}_2, \vec{v}_2 . We call \mathcal{R}_{Γ} the rest of the hypergraph. We have $\Phi_{\mathcal{R}}^{\text{comp}} = \Phi_{\mathcal{R}_{\Gamma}}^{\text{comp}} + [-c.p_{v_1}(X), -c.p_{v_2}(X)]$. By the definition of vehicle (cf. Definition 4.17) and addresses (cf. Definition 4.9), we necessarily have rays of the shape $+c.p_{v_1}(1 \cdot t_1), +c.p_{v_1}(\mathbf{r} \cdot t_2), +c.p_{v_2}(1 \cdot t_3), +c.p_{v_2}(\mathbf{r} \cdot t_4)$ for some t_1, t_2, t_3, t_4 . Since the lower part of \mathcal{R} is organised as a tree, the path address t_1 (resp. t_2) is designed to be equal to t_3 (resp. t_4) when they have the same position relatively to v_1 and v_2 or only α -unifiable when one is a subpath of the other. A cut star between the locations v_1 and v_2 uses the same variable for its two rays, hence it forces a connexion between two rays $+c.p_{v_1}(t)$ and $+c.p_{v_2}(u)$ where t and u are α -unifiable. There is only one possible such connexion by duplicating the cut and forming one diagram. Hence, the corresponding subgraph of $\mathcal{D}[\Phi_{\mathcal{R}}^{\text{comp}}]$ corresponds to a deterministic constellation. By partial pre-execution (cf. Lemma 3.17), we can focus on some steps of fusion while preserving correct diagrams. By fusion and by duplicating the cut star for each pair of rays $+c.p_{v_1}(t)$ and $+c.p_{v_2}(u)$

where t and u are α -unifiable, we can merge stars of the shape $[r, +c.p_{v_1}(1 \cdot t_i)]$ and $[r', +c.p_{v_2}(1 \cdot u_i)]$ and produce $[r, r']$ (same idea for \mathbf{r} instead of 1). This has exactly the same effect as relocating the atoms $\overleftarrow{v}_1, \overrightarrow{v}_1, \overleftarrow{v}_2$ and \overrightarrow{v}_2 in order to get rays of the shape $+c.p_{v_1}^{\leftarrow}(t_1), +c.p_{v_1}^{\rightarrow}(t_2), +c.p_{v_2}^{\leftarrow}(t_3), +c.p_{v_2}^{\rightarrow}(t_4)$ for the previous path addresses t_1, t_2, t_3 and t_4 with the cuts $[-c.p_{v_1}^{\leftarrow}(X), -c.p_{v_2}^{\leftarrow}(X)]$ and $[-c.p_{v_1}^{\rightarrow}(X), -c.p_{v_2}^{\rightarrow}(X)]$. This preserves the α -unifiability between rays and exactly coincides with $\Phi_{\mathcal{S}}^{\text{comp}}$ which removes the conclusions v_1 and v_2 , then relocates their sources which become conclusions. Therefore, the translation of \mathcal{R} and \mathcal{S} both have the same execution because of their structural equivalence which has no impact on the normal form. \square

Theorem 4.19 (Simulation of reduction for proof-nets). *For an MLL+MIX proof-net \mathcal{R} of normal form \mathcal{S} , we have $\text{Ex}(\Phi_{\mathcal{R}}^{\text{comp}}) = \Phi_{\mathcal{S}}^{\text{ax}}$.*

Proof. This result is a consequence of Lemma 4.18 by induction of the number of cut-elimination steps from \mathcal{R} to \mathcal{S} , as well as the fact that $\text{Ex}(\Phi_{\mathcal{S}}^{\text{comp}}) = \text{Ex}(\Phi_{\mathcal{R}}^{\text{ax}}) = \Phi_{\mathcal{R}}^{\text{ax}}$ since \mathcal{S} does not contain cuts. \square

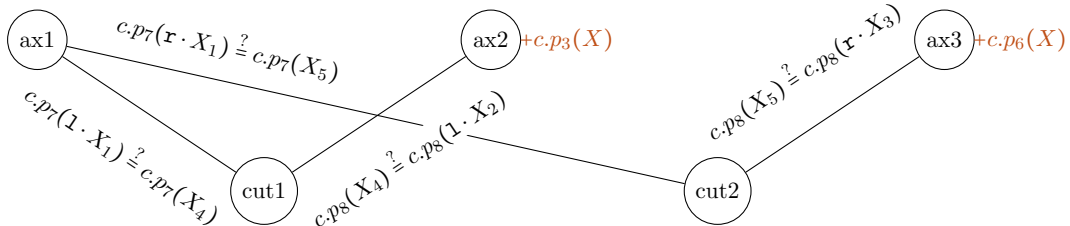
Example 4.20 (Correct cut-elimination). We have the following reduction $\mathcal{S} \rightsquigarrow^* \mathcal{S}'$ of proof-structure:



The proof-structure \mathcal{S} is translated into $\Phi_{\mathcal{S}}^{\text{comp}} =$

$$[+c.p_7(1 \cdot X), +c.p_7(\mathbf{r} \cdot X)] + [+c.p_3(X), +c.p_8(1 \cdot X)] + [+c.p_8(\mathbf{r} \cdot X), +c.p_6(X)] + [-c.p_7(X), -c.p_8(X)].$$

The ray $-c.p_7(X)$ can match either $+c.p_7(1 \cdot X)$ or $+c.p_7(\mathbf{r} \cdot X)$ and the same occurs for $-c.p_8(X)$. In order to satisfy these α -unifications, the cut star must be duplicated and each occurrence of cut must connect rays with the same address, *i.e.* the path addresses $1 \cdot X$ together and not $1 \cdot X$ with $\mathbf{r} \cdot X$. We obtain the following diagram:

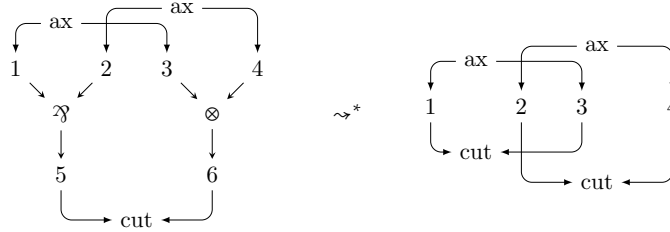


By case analysis, it is easy to check that it is the only possible diagram. Since the α -unification is exact (*cf.* Appendix A), it is simply a graph contraction doing no more than renamings and we get $\text{Ex}(\Phi_{\mathcal{S}}^{\text{comp}}) = [+c.p_3(X), +c.p_6(X)] = \text{Ex}(\Phi_{\mathcal{S}'}^{\text{comp}})$.

Example 4.21 (Incorrect cut-elimination). We have the following reduction $\mathcal{S} \rightsquigarrow^* \mathcal{S}'$ of proof-structure:

Structure	Axioms	Test 1	Test 2

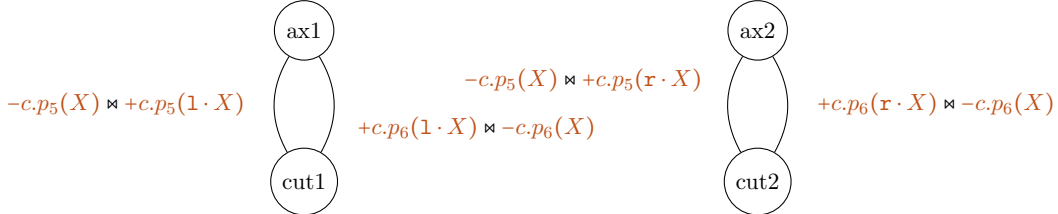
FIGURE 27. The axioms and tests of a proof-structure. The combination of axioms and a test corresponds to a correctness hypergraph representing a testing of the proof-structure.



The proof-structure \mathcal{S} is translated into $\Phi_{\mathcal{S}}^{\text{comp}} =$

$$[+c.p_5(1 \cdot X), +c.p_6(1 \cdot X)] + [+c.p_5(\mathbf{r} \cdot X), +c.p_6(\mathbf{r} \cdot X)] + [-c.p_5(X), -c.p_6(X)]$$

with the following dependency graph $\mathfrak{D}[\Phi_{\mathcal{S}}^{\text{comp}}]$:



The cycles in $\mathfrak{D}[\Phi_{\mathcal{S}}^{\text{comp}}]$ can be unfolded and yield infinitely many saturated correct diagrams, all actualising into $[\]$. We have $\text{Ex}(\Phi_{\mathcal{S}}^{\text{comp}}) = \sum_{i=1}^{\infty} [\] = \text{Ex}(\Phi_{\mathcal{S}'}^{\text{comp}})$.

4.3. Simulation of logical correctness. Since proof-structures are more general than proof-nets (*cf.* Definition 4.6), we have to check which proof-structures are “logically correct”. The idea of logical correctness traditionally corresponds to the fact of coming from a sequent calculus proof (*cf.* Figure 21b), which is taken as the natural understanding of what “being an actual proof” means.

A beautiful result of Girard, analysed by many subsequent works [26, 25, 75, 86, 31, 90, 13], is that the proof-structures that are proof-nets can be characterised by a topological/combinatorial property called a *correctness criterion*. While Girard’s original criterion, called the long-trip criterion [41, Section III.2], is about the set of walks in a proof-structure, we will here work with Danos and Regnier’s simplified criterion [26, Section 3.2] which is the most standard and which could not be treated by previous GoI models. Similarly to how a product has to pass several tests in order to be certified, this criterion defines tests to pass in order to be logically correct.

We define the *correctness hypergraphs* associated to a proof-structure \mathcal{S} as undirected copies of \mathcal{S} with one source of each \mathfrak{Y} -labelled hyperedge removed. They correspond to a

testing between the upper part made of axioms, the *tested* (corresponding to the vehicle in the stellar resolution¹⁴), and the lower part which is the *test*. This decomposition of a proof-structure into axioms and tests is illustrated in Figure 27. 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 (*cf.* Appendix B).

Since the idea of logical correctness is purely structural as well, our definitions still deals with unlabelled proof-structures.

Notation 4.22. Given a proof-structure $\mathcal{S} = (V, E, \text{in}, \text{out}, \ell_E)$, we write $\mathfrak{A}(\mathcal{S})$ the subset $P \subseteq E$ of \mathfrak{A} -labelled edges, *i.e.* $\mathfrak{A}(\mathcal{S}) = \{e \in E \mid \ell_E(e) = \mathfrak{A}\}$.

Definition 4.23 (Correctness hypergraph). Let $\mathcal{S} = (V, E, \text{in}, \text{out}, \ell_E)$ be a proof-structure. A *switching* is a map $\varphi : \mathfrak{A}(\mathcal{S}) \rightarrow \{\mathfrak{A}_L, \mathfrak{A}_R\}$. Its associated *correctness hypergraph* is the undirected hypergraph with labelled hyperedges $\mathcal{S}^\varphi = (V, E, \text{in}', \text{out}, \ell'_E)$ induced by the switching φ which is defined with

- $\text{in}'(e) = \{u\}$ where u is the left premise of e when $e \in \mathfrak{A}(\mathcal{S})$ and $\varphi(e) = \mathfrak{A}_L$;
- $\text{in}'(e) = \{u\}$ where u is the right premise of e when $e \in \mathfrak{A}(\mathcal{S})$ and $\varphi(e) = \mathfrak{A}_R$;
- $\text{in}'(e) = \text{in}(e) \cup \text{out}(e)$ in all other cases.

The labelling ℓ'_E is defined by $\ell'_E(e) = \varphi(e)$ when $e \in \mathfrak{A}(\mathcal{S})$ and $\ell'_E(e) = \ell_E(e)$ otherwise.

If testing proof-structures is seen as certifying products in a factory, the correctness criterion also gives them a label/certificate: the sequent they prove. This is represented as the possibility of labelling a proof-structure so that it corresponds to a proof-net. Hence, a proof-structure can actually correspond to several sequent calculus proofs depending on the labelling we choose.

Definition 4.24 (MLL-certification and MLL+MIX-certification). A proof-structure $\mathcal{S} = (V, E, \text{in}, \text{out}, \ell_E)$ is *MLL-certifiable* (*resp.* *MLL+MIX-certifiable*) with $\vdash A_1, \dots, A_n$ when there exists a vertex-labelling function ℓ_V such that $(V, E, \text{in}, \text{out}, \ell_V, \ell_E)$ is an MLL (*resp.* MLL+MIX) proof-net.

When there exists $\vdash A_1, \dots, A_n$ such that \mathcal{S} is MLL(+MIX)-certifiable with $\vdash A_1, \dots, A_n$ then we simply say that \mathcal{S} is MLL(+MIX)-certifiable.

Theorem 4.25 (MLL+MIX correctness). *A proof-structure \mathcal{S} is MLL+MIX-certifiable if and only if \mathcal{S}^φ is acyclic for all switching φ .*

Proof. Proven in [37, Theorem 4.7 and 4.8]. □

Theorem 4.26 (Danos-Regnier correctness). *A proof-structure \mathcal{S} is MLL-certifiable if and only if it is MLL+MIX-certifiable and \mathcal{S}^φ is connected for all switching φ .*

Proof. Proven in [26, Theorem 4]. □

The previous section handled cut-elimination by means of binary stars translating axiom and cut hyperedges. We now need to translate tests which contain \mathfrak{A} and \otimes hyperedges. Binary stars are no longer sufficient for a natural and satisfactory treatment of logical correctness since we now have to deal with a ternary hyperedge¹⁵: the tensor link.

¹⁴It also gives meaning to Girard's terminology of vehicle [55] which can be understood from its abstract definition, *e.g.* language as the vehicle of thought, or by its concrete definition, *e.g.* a car in an industry which is tested in order to be certified.

¹⁵As previous GoI models used binary objects (*e.g.* edge of a graph), ternary hyperedges could not be encoded.

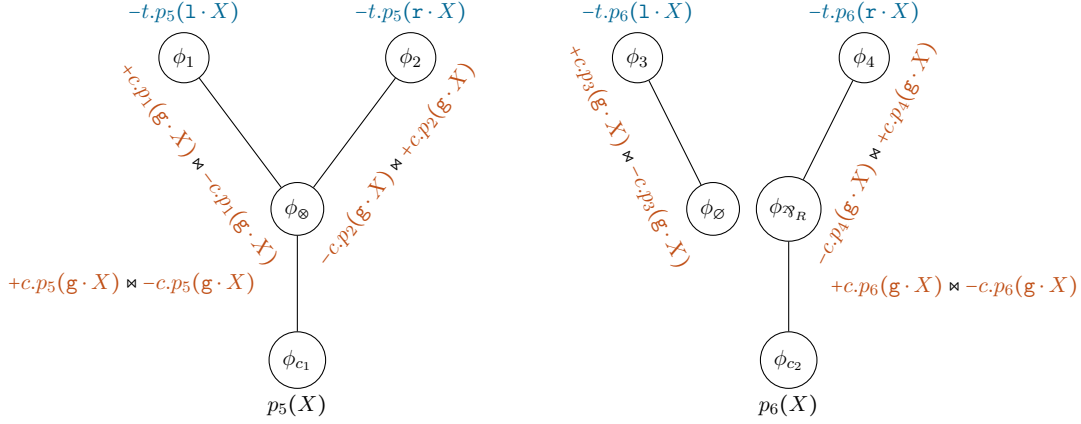


FIGURE 28. Dependency graph of the constellation corresponding to Test 2 in Figure 27.

For minor technical reasons, instead of directly translating hyperedges (which would be more natural), the vertices are translated¹⁶.

For our encoding, we use two colours: c (for computation) and t (for testing). A vehicle will be coloured with the colour c when we want its execution by connecting it with cuts and with t when being subject to an interaction against tests.

Definition 4.27 (MLL test). Let \mathcal{S} be a proof-structure and φ one of its switchings. The *test* associated to \mathcal{S}^φ is the constellation defined by $\Phi_{\mathcal{S}}^{\text{test}(\varphi)} := \sum_{v \in V\mathcal{S}^\varphi} v^\star$. We define the translation v^\star of a vertex v conclusion of an hyperedge e as follows:

- if $\ell_E(e) = \text{ax}$ then $v^\star = [-\text{addr}_{\mathcal{S}}(v, X), +c.p_v(\mathbf{g} \cdot X)]$;
- if $\ell_E(e) = \mathfrak{Y}_L$ and $\text{out}(e) = \{u, w\}$ then $v^\star = [-c.p_u(\mathbf{g} \cdot X), +c.p_w(\mathbf{g} \cdot X)] + [-c.p_w(\mathbf{g} \cdot X)]$;
- if $\ell_E(e) = \mathfrak{Y}_R$ and $\text{out}(e) = \{u, w\}$ then $v^\star = [-c.p_u(\mathbf{g} \cdot X)] + [-c.p_w(\mathbf{g} \cdot X), +c.p_v(\mathbf{g} \cdot X)]$;
- if $\ell_E(e) = \otimes$ and $\text{out}(e) = \{u, w\}$ then $v^\star = [-c.p_u(\mathbf{g} \cdot X), -c.p_w(\mathbf{g} \cdot X), +c.p_v(\mathbf{g} \cdot X)]$;
- if $v \in \text{Concl}(\mathcal{S})$ then $v^\star = [-c.p_v(\mathbf{g} \cdot X), p_v(X)]$;
- for each cut (hyperedge e such that $\ell_E(e) = \text{cut}$), we add a star $[-c.p_e(X), -c.p_e(X)]$.

The technical purpose of the constant \mathbf{g} is to make the terms of the tests distinct from the ones of the vehicle so that the cuts can be applied to both the vehicle and the tests by simply changing colours ($-c.p_v(X)$ matches both $+c.p_v(X)$ and $+c.p_v(\mathbf{g} \cdot X)$). This allows more flexibility in the definitions.

Tests for a proof-structure \mathcal{S} are actually designed so that $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}]$ is structurally equivalent to \mathcal{S}^φ , as illustrated in Figure 28. We make this idea explicit by the following lemma and show that it leads to a simulation of logical correctness in the stellar resolution.

Lemma 4.28 (Structural realisation). *Let \mathcal{S} be a proof-structure and φ one of its switchings. We have $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}] \simeq \mathcal{S}^\varphi$.*

¹⁶Although they could have been added without problem, conclusion hyperedge were omitted for convenience in Definition 21c. However, since we will need to translate conclusions as rays, a translation of hyperedge would lack this information.

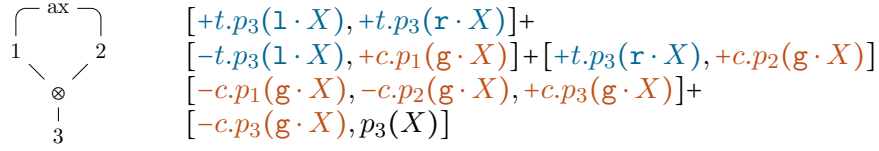


FIGURE 29. Incorrect correctness hypergraph for a proof-structure \mathcal{S} and its translation. Notice that the cycle is turned into a computational cycle (a loop in a program).

Proof. Remark that a test (*cf.* Definition 4.27) could alternatively be defined by a translation of hyperedges instead of vertices. Let v be a vertex of \mathcal{S}^φ . It is target of an hyperedge e with sources $\text{in}(e) = \{u_1, \dots, u_k\}$ with $k \leq 2$. We define $e^\star := v^\star$. By case analysis on e and by Definition 4.27, there is a correspondence between v, u_1, \dots, u_k and rays of e^\star . We write v_e^\star for the ray corresponding to v in e . The tests are designed so that two hyperedges $e_1, e_2 \in E^{\mathcal{S}^\varphi}$ share a vertex v if and only if $v_{e_1}^\star$ is α -unifiable with $v_{e_2}^\star$. Notice that they both correspond to the same formula seen as input or output depending on the e in v_e^\star . These two points of view correspond to matchable rays of opposite colours. In conclusion, $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}]$ preserves the structure of \mathcal{S}^φ as hypergraph and links vertices in the same way. \square

A technical corollary is that the translation of correctness hypergraph is deterministic and exact (*cf.* Definition 3.9). It ensures that all diagrams are correct but also that if its dependency graph is connected and acyclic, there is no branching leading to non-deterministic choices for a ray.

Corollary 4.29. *Let \mathcal{S} be a proof-structure and φ one of its switchings. Then $+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}$ is deterministic and exact.*

Proof. By Lemma 4.28, $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}] \simeq \mathcal{S}^\varphi$. The hypergraph \mathcal{S}^φ has vertices which are uniquely connected, *i.e.* for each $v \in V^{\mathcal{S}^\varphi}$, there is only one $e \in E^{\mathcal{S}^\varphi}$ such that $v \in \text{in}(e)$ or $v \in \text{out}(e)$. Therefore, rays are uniquely connected in $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}]$ and $+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}$ is deterministic. A simple analysis on Definition 4.27 shows that it is also exact. \square

The idea for MLL correctness is that we would like to have $\text{Ex}(+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}) = [p_{v_1}(X), \dots, p_{v_n}(X)]$ with $\text{Concl}(\mathcal{S}) = \{v_1, \dots, v_n\}$, which would imply that \mathcal{S} has an arborescent shape. In his original paper [55, Section 2.3], Girard only considers the case of cut-free proofs by forbidding cyclic diagrams and the empty star. However, it seems that these definitions lead to a technical mistakes in case of a cyclic correctness hypergraph. We describe the problem and explain why it does not occur in our case.

We write Ex^{RT} for the execution restricted to tree-shaped diagrams (execution used by Girard) which applies the operator \ddagger and \flat . If we consider the incorrect proof-structure \mathcal{S}' in Example 4.21, we have $\text{Ex}^{\text{RT}}(+t.\Phi_{\mathcal{S}'}^{\text{ax}} \uplus \Phi_{\mathcal{S}'}^{\text{test}(\varphi)}) = \emptyset$ making this incorrect proof-structure invisible in the normal form. Assume we have a correct proof-structure \mathcal{R} . By Lemma 3.14, the union of \mathcal{R} and \mathcal{S}' is translated into a correct constellation although it should not (because \mathcal{S}' is incorrect). This is due to the absence of conclusion caused by cuts. However,

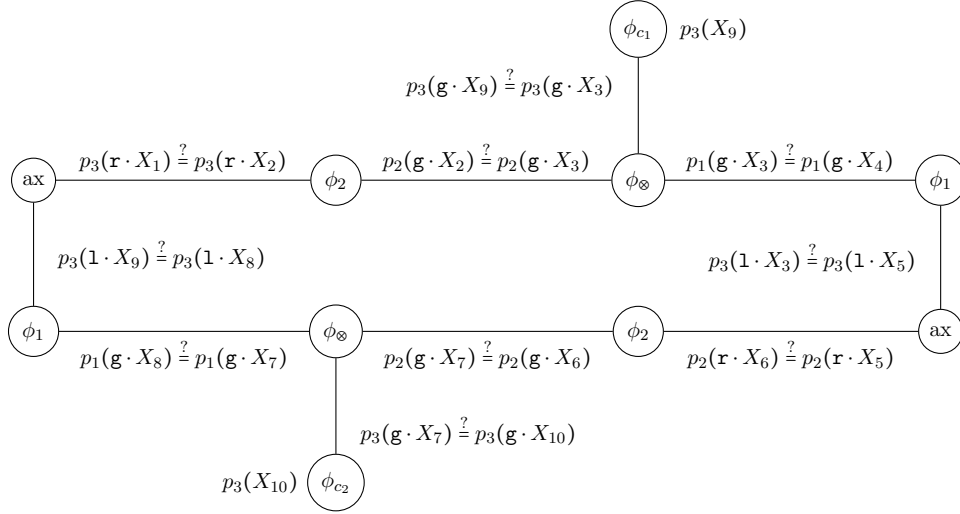


FIGURE 30. Example of a correct and saturated cyclic diagram for the constellation from Figure 29 actualising into $[p_3(X_9), p_3(X_{10})]$. The cycle can be extended infinitely many times by adding copies of three stars of the constellation.

even without cuts, the same phenomena can occur. For instance, if we consider the correctness hypergraph of Figure 29, infinitely many diagrams can be constructed because of the loop of dependencies but all the corresponding diagrams have free polarised rays. Such diagrams are erased by the operator $\not\downarrow$. Therefore, $\text{Ex}^{\text{RT}}(+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}) = \emptyset$.

A solution is to make incorrect proofs visible in the normal form. Our definition of execution makes this possible. In case of closed cycles, we can construct closed cyclic diagrams (which are accepted). Since proof-structures are always translated into exact constellations, such diagrams will always actualise into the empty star. As for cyclic diagrams with free rays, in the case of proof-structure, they will yield infinitely many stars by unfolding the loop infinitely many times. This makes incorrectness visible in the output of execution.

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 to proofs indicating the presence of cycles but we were able to simulate this idea by modifying the notion of execution instead.

We can now state the Danos-Regnier correctness criterion in the stellar resolution.

Proposition 4.30. *If a connected multiplicative correctness hypergraph \mathcal{S}^φ has no conclusion then it is cyclic.*

Proof. Since \mathcal{S}^φ 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. Therefore, by basic properties of cycles in graph theory, \mathcal{S}^φ must be cyclic. \square

Theorem 4.31 (Stellar correctness criterion). *A proof-structure \mathcal{S} such that $\text{Concl}(\mathcal{S}) = \{v_1, \dots, v_n\}$ is MLL-certifiable if and only if for all switchings φ , we have:*

$$\text{Ex}(+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}) = [p_{v_1}(X), \dots, p_{v_n}(X)].$$

Proof. Let φ be a switching of \mathcal{S} . By Lemma 4.28, we know that $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}]$ is structurally equivalent to \mathcal{S}^{φ} .

- (\Rightarrow) Assume \mathcal{S}^{φ} is connected and acyclic and so is $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}]$. By Lemma 3.12 and 3.13, $\mathfrak{D}[+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}]$ has a unique correct diagram. We can construct a diagram δ by following the links of \mathcal{S}^{φ} . We have that G_{δ} is a tree containing all conclusions v_1, \dots, v_n . Hence $\Downarrow \delta$ is the star $[p_{v_1}(X), \dots, p_{v_n}(X)]$.
- (\Leftarrow) Assume that $\text{Ex}(+t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}) = [p_{v_1}(X), \dots, p_{v_n}(X)]$. Assume by contradiction that \mathcal{S}^{φ} has at least two connected components. Assume that a component has no conclusion (because of cuts). Then, by Proposition 4.30, there is a cycle yielding infinitely many closed diagrams normalising into the empty star $[\]$. Hence, all connected components must have free rays corresponding to conclusion. By the independence of connected component (*cf.* Lemma 3.14), we can independently execute each connected component. Since they correspond to deterministic and exact subconstellation, the normalisation produces the constellation $\phi_1 + \dots + \phi_k$ for the k connected component, contradicting the hypothesis that we normalise into a single star. Therefore, \mathcal{S}^{φ} must be connected. Now, assume by contradiction that \mathcal{S}^{φ} 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 $[p_{v_1}(X), \dots, p_{v_n}(X)]$, contradicting the hypothesis. Therefore, \mathcal{S}^{φ} must also be acyclic. This proves that \mathcal{S}^{φ} must be a tree for any switching φ , *i.e.* \mathcal{S} is MLL-certifiable. □

The following corollary finally extends the logical correctness to MLL+MIX and suggest a more general variant which also captures MLL.

Corollary 4.32. *Let \mathcal{S} be a proof-structure and $\Phi := +t.\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{test}(\varphi)}$ be the constellation corresponding to the correctness hypergraph \mathcal{S}^{φ} for some switching φ . We have:*

- \mathcal{S}^{φ} is acyclic $\Leftrightarrow \mathfrak{D}[\Phi]$ is acyclic $\Leftrightarrow |\text{Ex}_A(\Phi)| < \infty$;
- \mathcal{S}^{φ} is connected and acyclic $\Leftrightarrow \mathfrak{D}[\Phi]$ is a deterministic tree $\Leftrightarrow |\text{Ex}(\Phi)| = 1$;
- \mathcal{S}^{φ} is connected and acyclic $\Leftrightarrow \Phi$ normalises into the star of its uncoloured rays.

Proof. The first equivalence of each point are direct consequences of Lemma 4.28. It only remains to show the last equivalences.

- If $\mathfrak{D}[\Phi]$ is acyclic, then by Lemma 3.11, $|\text{Ex}_A(\Phi)| < \infty$ because Φ is exact and deterministic (*cf.* Corollary 4.29). Now assume that $|\text{Ex}_A(\Phi)| < \infty$. The proof of Theorem 4.31 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. If $\mathfrak{D}[\Phi]$ is also connected, then by Lemma 3.12 and 3.13 and the fact that Φ is deterministic and exact (*cf.* 4.29), there is exactly a unique correct saturated diagram, hence 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.* proof of Theorem 4.31) 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}^φ 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.

□

The analysis of the computational and logical content of proofs in the Transcendental Syntax leads to a decomposition of proof-structures and give a new outlook on what being a “correct” proof means in proof theory.

Definition 4.33 (Translation of a proof-structure). The translation of a proof-structure \mathcal{S} is defined as the constellation $\mathcal{S}^\star = (\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{cut}}, \Phi_{\mathcal{S}}^{\text{format}})$ where $\Phi_{\mathcal{S}}^{\text{format}}$ is called *format*¹⁷ and is defined by $\Phi_{\mathcal{S}}^{\text{format}} := \{\Phi_{\mathcal{S}}^{\text{test}(\varphi)} \mid \varphi \text{ is a switching of } \mathcal{S}\}$.

As shown in Theorem 4.31, \mathcal{S}^\star corresponds to a proof-net if and only if it passes all the tests $\Phi \in \Phi_{\mathcal{S}}^{\text{format}}$. In particular, any proof-structure can be seen as a program (its set of axioms) already coming with some implicit constraining tests. This corresponds to a sort of hidden pre-typing. Proof-nets are programs coming with tests it can passes. Hence tests corresponds to a *certification* for programs. This demonstrates what Girard means by “making the hidden assumptions of logic explicit” (*cf.* Section 1).

Notice that these tests are entirely definable by MLL formulas (and thus, dependent of them) because only vertices of the lower part of \mathcal{S}^φ are used in the translation of Definition 4.27. We obtain a more general meaning of the idea of proof: a proof is a computational entity passing the tests corresponding to a certain notion of formula/specification yet to be defined.

5. EMERGENCE OF FORMULAS

Generalising the correctness criterion of proof-nets actually gives rise to a notion of type (or formula). We need to fix a symmetric binary relation between constellations formalising what we mean by “correctly passing a test”. For instance, Corollary 4.32 suggests three such relations we call \perp_{fin} , \perp^1 and \perp^R but others can be designed depending on what we want. The intention behind orthogonality relations is that they define linear negations for linear logic.

Definition 5.1 (Orthogonality). We define binary relations of *orthogonality* between two constellations Φ_1 and Φ_2 *w.r.t.* a set of colours $A \subseteq C$:

- $\Phi_1 \perp_A^{\text{fin}} \Phi_2$ when $|\text{Ex}_A(\Phi_1 \uplus \Phi_2)| < \infty$;
- $\Phi_1 \perp_A^1 \Phi_2$ when $|\text{Ex}_A(\Phi_1 \uplus \Phi_2)| = 1$;
- $\Phi_1 \perp_A^R \Phi_2$ when $\text{Ex}(\Phi_1 \uplus \Phi_2) = \{\text{Roots}(\Phi_1 \uplus \Phi_2)\}$ where $\text{Roots}(\Phi)$ is the star of uncoloured rays in Φ .

¹⁷*Gabarit* in Girard’s original papers. We choose the term “format” because it is less awkward in English and reminds of file formats in a computer.

The orthogonal of a set of constellations \mathbf{A} is defined by $\mathbf{A}^{\perp A} := \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp_A \Phi'\}$ for a relation of orthogonality \perp .

In order to allow typing for partial evaluations, the orthogonality relation \perp_A has to be parametrised by a set of colours A but we omit this parameter when considering all colours in C .

The orthogonal $\mathbf{A}^{\perp A}$ corresponds to the set of all constellations passing the tests of \mathbf{A} . But since test and tested are both constellations and that orthogonality relations are symmetric, they have interchangeable roles, hence \mathbf{A} is also the set of constellations passing the tests of \mathbf{A}^{\perp} .

The orthogonality \perp^{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.

Lemma 5.2 (Invariance of orthogonality under execution). *Let Φ and Φ' be constellations such that $\Phi \cap_{A \cup B} \Phi' = \emptyset$. We have $\Phi \perp \Phi'$ if and only if $\text{Ex}(\Phi) \perp \Phi'$ for $\perp \in \{\perp^1, \perp^{\text{fin}}, \perp^R\}$.*

Proof. These relation are satisfied when $P(\text{Ex}(\Phi \uplus \Phi'))$ is satisfied for some property P . By the lemma of partial pre-execution (cf. Lemma 3.17), we have $\text{Ex}(\text{Ex}(\Phi) \uplus \Phi') = \text{Ex}(\Phi \uplus \Phi')$. Hence we have $P(\text{Ex}(\Phi \uplus \Phi'))$ if and only if $P(\text{Ex}(\text{Ex}(\Phi) \uplus \Phi'))$, meaning that we have $\Phi \perp \Phi'$ if and only if $\text{Ex}(\Phi) \perp \Phi'$. \square

5.1. Types as labels certified by tests (l'Usine). In this section, we construct formulas by generalising the logical correctness of Section 4.3.

Definition 5.3 (Type label). A *type* is an object (or label) A associated to a finite set of constellations $\text{Tests}(A)$ called its *tests*. We say that a constellation Φ is of type A *w.r.t.* \perp if and only if $\Phi \in \text{Tests}(A)^{\perp}$.

A type corresponds to a *specification* for a computational entity (typically a program) certified by an associated set of *tests* as we do in software engineering or formal methods. For instance, in model checking [14], 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). 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 [63, 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} as we will see in Observation 6.15.

The purpose of having finite set of tests is to make type checking computable. However, this is only happens under some conditions such as the orthogonality relation between computable. 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.

Although similar, typing with finite tests is not quite the type checking with typing rules which appears in typed λ -calculus. Girard's Usine is meant to check *cut-free proofs* only,

whereas it is possible to verify the type of normalisable terms for a sequent $\vdash (\lambda x.M)N : A$ without actually doing the normalisation. This is because the transcendental syntax distinguishes between:

- characterising the shape of our logical objects (cut-free proofs), which corresponds to *Usine* and to the logical rules of sequent calculus;
- defining the use of our logical objects (interaction with cuts), which corresponds to *Usage* and to the cut rule of sequent calculus.

These notions are often mixed in proof theory: in order to even have the right to write an elimination rule such as modus ponens, we implicitly assume that we are given an object which has the shape of a proof of implication $A \Rightarrow B$ and that its interaction with any proof of A will produce a proof of B . In other words, we assume an *adequation* between *Usine* and *Usage* or that we have primitive objects (defined by finite tests) which will behave soundly *w.r.t.* some *use* (behaving like functions in the case of modus ponens).

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 atomic cut-elimination. By constructing a syntax hypergraph from a sequent, Definition 4.27 can be used.

Definition 5.4 (Test of a sequent). Let $\vdash \Gamma$ be a sequent of MLL where $\Gamma \subseteq \mathcal{F}_{\text{MLL}}$ 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 vertex 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;
- $ST(A \wp B)$ is an hyperedge labelled by \wp linking the conclusion of $ST(A)$ and $ST(B)$ as sources and having a vertex labelled by $A \wp B$ 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 4.23) 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 to the sequent $\vdash \Gamma$ and the switching φ is defined as the constellation $\mathbf{Test}(\vdash \Gamma)^\varphi$ such that $I_{\mathbf{Test}(\vdash \Gamma)^\varphi} := V^{ST(\vdash \Gamma)^\varphi}$ (it is indexed by vertices of the syntax tree) and $\mathbf{Test}(\vdash \Gamma)^\varphi[v] := v \star$ where $\ell(v)$ is not an atomic formula and $v \star$ is the translation of Definition 4.27. Notice that we reject the translation of atomic formulas because they depend upon a proof-structure \mathcal{S} . This dependency is actually not necessary.

The *set of tests* associated to the sequent $\vdash \Gamma$ is defined by $\mathbf{Tests}(\vdash \Gamma) := \{\mathbf{Test}(\vdash \Gamma)^\varphi \mid \varphi \text{ is a switching of } ST(\vdash \Gamma)\}$.

We now defined MLL sequents as type labels in the sense of Definition 5.3. 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 31. One solution is to extended the notion of colour shifts of Definition 4.14 to change rays in order to force the α -unification

Definition 5.5 (Conjugation). A *conjugation* $\mu : \text{IdRays}(\mathbb{S}) \rightarrow \text{IdRays}(\mathbb{S}')$ between two signatures \mathbb{S} and \mathbb{S}' is a function replacing the rays of a constellation such that it preserves its structure, *i.e.* $\mathfrak{D}[\mu(\Phi)] \simeq \mathfrak{D}[\Phi]$.



FIGURE 31. We expect this proof-structure to be able to pass any test of $\text{Tests}(\vdash X_1 \otimes X_2, X_1^\perp \wp X_2^\perp)$. 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.

Another solution is to use a computational realisation of conjugations by using stars $[-t, +u]$ where $+t$ is a ray of the vehicle and $-u$ is a ray of a test. This corresponds to a sort of generalised cut allowing a trivial connexion between two rays. We give a more general definition of that idea.

Definition 5.6 (Adapter). Let Φ be a constellation. An *adapter* for Φ is a star $[t, u]$ where t' and u' are rays in Φ which are respectively t and u with opposite polarity.

Conjugations induce adapters. Whenever we have a conjugation μ such that $\mu(r_1) = r_2$, we can construct an adapter $[r'_1, r'_2]$ where r'_i has a polarity opposite to r_i . We use adapters and conjugations indistinctly in this paper.

Notice that the translation of atomic formulas in Definition 5.4 actually corresponds to adapters between a vehicle and a test, hence tests are indeed independent of vehicles. This dependency is only artificial and appears when considering proof-structures as an entity which cannot be decomposed.

Proposition 5.7 (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 $\text{Ex}(\Phi_{\mathcal{S}}^{\text{test}(\varphi)}) = \text{Ex}(\text{Test}(\vdash \Gamma)^\varphi \uplus \Phi)$.*

Proof. The constellation $\Phi_{\mathcal{S}}^{\text{test}(\varphi)}$ corresponds to the syntax tree of a formula with exactly one premise cut for each \wp vertex. Hence, it naturally induces a sequent $\vdash \Delta$ and we defined $\Gamma := \Delta$. The constellation $\text{Test}(\vdash \Gamma)^\varphi$ structurally corresponds to $\Phi_{\mathcal{S}}^{\text{test}(\varphi)}$ without the upper rays of colour $-t$ which allows connexion with the right vehicle. Apart from that, they both use the same translation function $(\cdot)^\star$ on vertices for correctness hypergraphs. Assume we have a star for atom $[-t.p_v(t), +c.q_w(x)]$ related to some star $[-c.q_w(x), \dots]$ in $\text{Test}(\vdash \Gamma)^\varphi$. During the execution, they will merge into $[-t.p_v(t), \dots]$. However, it is possible to construct Φ so to reproduce this step with an adapter $[-t.p_v(t), +c.q_A(x)]$ (by definition, the star $[-c.q_A(x), \dots]$ which is isomorphic to $[-c.q_w(x), \dots]$ must be present in $\Phi_{\mathcal{S}}^{\text{test}(\varphi)}$). Moreover, because of the structural equivalence between the two constellations, they only differ by conjugation. It is then possible to extend Φ so that $\text{Test}(\vdash \Gamma)^\varphi$ is turned exactly into $\Phi_{\mathcal{S}}^{\text{test}(\varphi)}$. It follows that the two constellations must have the same normal form. \square

Definition 5.8 (Typing). We say that a constellation Φ is of type $\vdash \Gamma$, written $\vdash \Phi : \Gamma$ when $\Phi \in (\Phi_{\vdash \Gamma}^{\text{test}(\varphi)} \uplus \Phi_\mu)^\perp$ for a set of adapters Φ_μ and all switchings φ of $\vdash \Gamma$.

Proposition 5.9 (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 $\vdash +t.\Phi_{\mathcal{S}}^{\text{ax}} : \Gamma$ with $\perp \in \{\perp^1, \perp^R\}$. The same statement holds for MLL+MIX w.r.t. \perp^{fin} .

Proof. By Proposition 5.7, there exist some sequent $\vdash \Gamma$ such that $\Phi_{\mathcal{S}}^{\text{test}(\varphi)}$ is simulated by $\Phi \uplus \text{Test}(\vdash \Gamma)^\varphi$ for some constellation of adapters Φ . By invariance of orthogonality under execution (cf. Lemma 5.2), this connexion is equivalent to a connexion between $+t.\Phi_{\mathcal{S}}^{\text{ax}}$ and $\Phi_{\mathcal{S}}^{\text{test}(\varphi)}$. The orthogonality $+t.\Phi_{\mathcal{S}}^{\text{ax}} \in \text{Tests}(\vdash \Gamma)^\perp$ and the same statement for MLL+MIX (w.r.t. \perp^{fin}) both hold by a direct consequence of Corollary 4.32. \square

Now that we have finite tests able to certify computational entities, what remains is to be able to express the *real use* of these objects (defined by the set of their potential partners in interaction), which is usually infinite. Girard’s Usine is then only an effective approximation of this ideal use.

5.2. Interactive typing (l’Usage). By using an idea of *interactive typing* which was already present in ludics [48] and in the Geometry of Interaction [50, 98], it is possible to define “semantic-free formulas”. Such formulas are defined as set of constellations, not from a given semantics but from how the constellations interact with each other. We need two ingredients: a notion of *interaction* (the execution of constellations) and a symmetric and binary *orthogonality* relation which opposes constellations. This relation represents a *point of view* on interaction and formalises what it means to “interact correctly”.

This actually extends the previous idea of type but instead of arbitrary tests, a constellation is given a meaning by all its possible interaction with other constellations, relatively to a specific point of view. Since these potential opponents still define the meaning of a constellation, we keep the term of “test” (although effective testing is no more possible in general because a set of tests can be infinite).

The constellations are grouped into arbitrary sets called *pre-behaviours*, giving rise a notion of formula corresponding to a computational version of phase semantics [41, Section II.5].

Definition 5.10 (Pre-behaviour). A *pre-behaviour* \mathbf{A} is a set of constellations.

We now define the notion of *behaviour* which corresponds to the formulas/types appearing in linear logic. They represent idealised logical notions that we can only approximate if we wish for an effective type checking.

Definition 5.11 (Behaviour). A pre-behaviour \mathbf{A} is a *behaviour* when there exists a pre-behaviour \mathbf{B} such that $\mathbf{A} = \mathbf{B}^\perp$.

More intuitively, a behaviour is a group of computational objects which is entirely characterised by a (potentially infinite) set of tests: a pre-behaviour \mathbf{A} is a behaviour when there exists a set of tests (constellations) \mathbf{B} such that \mathbf{A} is exactly the set of constellations passing all the tests of \mathbf{B} . In other words, \mathbf{A} is a behaviour if and only if it is *testable*.

Lemma 5.12 (Invariance of typing under execution). *Let Φ be a constellation and \mathbf{A} a behaviour. We have $\Phi \in \mathbf{A}$ if and only if $\text{Ex}(\Phi) \in \mathbf{A}$.*

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 $\Phi \perp \Phi'$ for any $\Phi' \in \mathbf{A}^\perp$ if and only if $\text{Ex}(\Phi) \perp \Phi'$. This is the consequence of the invariance of orthogonality under execution (cf. Lemma 5.2). \square

There is an alternative (more standard) definition of behaviours which is called *bi-orthogonal closure*. It states a sort of balance between tests and tested. This is actually something very important we require in linear logic and which is not true in intuitionistic logic¹⁸: the linear negation is involutive.

Proposition 5.13 (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 [67, Proposition 15]. \square

In order to define the tensor of two behaviours (corresponding to an interactive version of the usual tensor type label), we have to exclude any interaction between them because we want the tensor to connect two independent proof-structures. Definition 3.15 of set of variables shared by two constellations makes this possible.

Definition 5.14 (Disjointness of behaviours). Let \mathbf{A} and \mathbf{B} be two behaviours and a set of colours $C' \subseteq C$. They are *disjoint* when for all $\Phi_A \in \mathbf{A}$ and $\Phi_B \in \mathbf{B}$, we have $\Phi_A \cap_{C'} \Phi_B = \emptyset$.

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 $\mathfrak{D}[\Phi_A \uplus \Phi_B]$: for instance, if we had a path from Φ_1 to a variable X in Φ_2 , this variable is still accessible from Φ_2 , hence X is shared by the two constellations.

Definition 5.15 (Pre-tensor). Let \mathbf{A} and \mathbf{B} be disjoint pre-behaviours. We define their pre-tensor by $\mathbf{A} \odot \mathbf{B} = \{\Phi_1 \uplus \Phi_2 \mid \Phi_1 \in \mathbf{A}, \Phi_2 \in \mathbf{B}\}$.

Definition 5.16 (Tensor). Let \mathbf{A} and \mathbf{B} be disjoint behaviours. We define their tensor by

$$\mathbf{A} \otimes \mathbf{B} = (\mathbf{A} \odot \mathbf{B})^{\perp\perp}.$$

The pre-tensor is the natural definition of the 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 have what we call an *internal completeness* property.

Proposition 5.17 (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 $\mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C}$. \square

The other connectives are then defined by interactive testing, *e.g.* the elements of $\mathbf{A} \wp \mathbf{B}$ are the elements passing the tests of $\mathbf{A}^\perp \otimes \mathbf{B}^\perp$. This is why we can speak about *interactive types* as we did in the introduction of this paper.

Definition 5.18 (Par and linear implication). Let \mathbf{A}, \mathbf{B} be disjoint behaviours. We define:

$$\mathbf{A} \wp \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp \quad \text{and} \quad \mathbf{A} \multimap \mathbf{B} = \mathbf{A}^\perp \wp \mathbf{B}.$$

¹⁸In intuitionistic logic, we do not have $\neg\neg A = A$ for any formula A .

$$\Phi_1 = [X, +c(X)] \begin{cases} \cdots [-c(1 \cdot X)] = \Phi_2 \\ \cdots [-c(\mathbf{r} \cdot X)] = \Phi_3 \end{cases}$$

FIGURE 32. Counter-example of non-associativity. We have $\text{Ex}_{\{c\}}(\Phi_1 \uplus \Phi_2) = [1 \cdot X]$ and $\text{Ex}_{\{c\}}(\text{Ex}_{\{c\}}(\Phi_1 \uplus \Phi_2) \uplus \Phi_3) = [-c(\mathbf{r} \cdot X)] + [1 \cdot X]$, but $\text{Ex}_{\{c\}}(\Phi_1 \uplus \text{Ex}_{\{c\}}(\Phi_2 \uplus \Phi_3)) = [-c(1 \cdot X)] + [\mathbf{r} \cdot X]$ which is different.

Remark 5.19 (Implicit exchange). The commutativity and associativity of \otimes are preserved for the \mathfrak{A} . For instance $\mathbf{A} \mathfrak{A} \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp = (\mathbf{B}^\perp \otimes \mathbf{A}^\perp)^\perp = \mathbf{B} \mathfrak{A} \mathbf{A}$. This corresponds to the fact that the exchange rule is implicit in usual linear logic.

In Figure 32, we show that associativity fails when execution is treated as a binary operator on constellations. However, this property is fundamental when speaking about (categorical [82, 94]) models of linear logic. We need a restriction on the interaction between constellations as in Seiller's works [98, Proposition 12][99, Theorem 24] where the same problem exists.

A technical precondition is defined for the associativity, and *trefoil property* [99, Theorem 40] is stated as a corollary. In particular, the trefoil property ensures that one can define a $*$ -autonomous category, which characterises denotational models of MLL [96].

Theorem 5.20 (Associativity of execution). *Choose a set of colours $A \subseteq C$. For constellations Φ_1, Φ_2 and Φ_3 such that $\Phi_1 \pitchfork_A \Phi_2 \pitchfork_A \Phi_3 = \emptyset$, we have:*

$$\text{Ex}_A(\Phi_1 \uplus \text{Ex}_A(\Phi_2 \uplus \Phi_3)) = \text{Ex}_A(\text{Ex}_A(\Phi_1 \uplus \Phi_2) \uplus \Phi_3).$$

Proof. Assume we have $\Phi_1 \pitchfork_A \Phi_2 \pitchfork_A \Phi_3 = \emptyset$. Hence, by definition, no variable is shared by the three constellations. Let $P(x_j^i)$ with x_j^i a variable using the notations of Definition 3.15, be the set of paths reaching x_j^i 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 3.17), the paths $P(x_j^i)$ traversing Φ_2 and Φ_3 can be reduced with no effect on other connexions (since no variables are shared). Hence, the stars of Φ_1 can connect to the stars of $\text{Ex}_A(\Phi_2 \uplus \Phi_3)$ in the same way as in $\Phi_2 \uplus \Phi_3$. It follows that $\text{Ex}_A(\Phi_1 \uplus \text{Ex}_A(\Phi_2 \uplus \Phi_3)) = \text{Ex}_A(\Phi_1 \uplus \Phi_2 \uplus \Phi_3)$. By the same reasoning, we also have $\text{Ex}_A(\text{Ex}_A(\Phi_1 \uplus \Phi_2) \uplus \Phi_3) = \text{Ex}_A(\Phi_1 \uplus \Phi_2 \uplus \Phi_3)$, hence execution is associative. \square

Theorem 5.21 (Trefoil Property for execution-based orthogonality). *Choose a set of colours $A \subseteq C$. For constellations Φ_1, Φ_2, Φ_3 and for $i, j, k \in \{1, 2, 3\}$ such that $\Phi_1 \pitchfork_A \Phi_2 \pitchfork_A \Phi_3 = \emptyset$, we have:*

$$\Phi_1 \perp_A \text{Ex}_A(\Phi_2 \uplus \Phi_3) \quad \text{if and only if} \quad \text{Ex}_A(\Phi_1 \uplus \Phi_2) \perp_A \Phi_3.$$

Proof. Assume that P is a property corresponding to the orthogonality relation \perp based on execution, *i.e.* we have $\Phi_1 \perp_A \Phi_2$ if and only if $P(\text{Ex}(\Phi_1 \uplus \Phi_2))$. The statement can be rewritten as follows: $P(\text{Ex}_A(\Phi_1 \uplus \text{Ex}_A(\Phi_2 \uplus \Phi_3)))$ if and only if $P(\text{Ex}_A(\text{Ex}_A(\Phi_1 \uplus \Phi_2) \uplus \Phi_3))$. This is a direct consequence of the associativity (*cf.* Theorem 5.20). \square

The trefoil property leads to the *adjunction*¹⁹ which has been stated in previous models of GoI [50, Theorem 3] or in ludics.

¹⁹Corresponding to the categorical adjunction in cartesian closed categories.



FIGURE 33. Rules for the units of MLL. Two hyperedges with no input and a single output are added in the construction of proof-structures.

Corollary 5.22 (Adjunction). *Choose a set of colours $A \subseteq C$. For all constellations Φ_f , Φ_a and Φ_b such that $\Phi_a \pitchfork_A \Phi_b = \emptyset$, we have:*

$$\Phi_f \perp_A \Phi_a \uplus \Phi_b \quad \text{if and only if} \quad \text{Ex}_A(\Phi_f \uplus \Phi_a) \perp_A \Phi_b.$$

Proof. By symmetry of orthogonality relations and invariance of orthogonality under execution (cf. Lemma 5.2), we have $\Phi_f \perp_A \Phi_a \uplus \Phi_b$ if and only if $\Phi_f \perp_A \text{Ex}_A(\Phi_a \uplus \Phi_b)$. In order to conclude with the trefoil property, it remains to show the precondition, *i.e.* that we have $\Phi_f \pitchfork_A \Phi_a \pitchfork_A \Phi_b = \emptyset$. We assumed $\Phi_a \pitchfork_A \Phi_b = \emptyset$, 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. \square

Thanks to the adjunction, it is possible to define a more intuitive linear implication seeing a constellation Φ_f as a function interacting with a constellation Φ_a as argument.

Proposition 5.23 (Alternative linear implication). *Let \mathbf{A}, \mathbf{B} be two disjoint behaviours. We have $\mathbf{A} \multimap \mathbf{B} = \{\Phi_f \mid \forall \Phi_a \in \mathbf{A}, \Phi_f \perp \Phi_a \text{ and } \text{Ex}(\Phi_f \uplus \Phi_a) \in \mathbf{B}\}$.*

Proof. By Definition 5.18, 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 $\Phi_a \in \mathbf{A}$, $\Phi_f \perp \Phi_a$ and $\text{Ex}(\Phi_f \uplus \Phi_a) \in \mathbf{B}$. Since \mathbf{B} is a behaviour, by Definition 5.11, there exists $\Phi_{b'} \in \mathbf{B}^\perp$ such that $\text{Ex}(\Phi_f \uplus \Phi_a) \perp \Phi_{b'}$. By the adjunction (cf. Corollary 5.22), $\Phi_f \perp (\Phi_a \uplus \Phi_{b'})$, hence $\Phi_f \in (\mathbf{A} \otimes \mathbf{B}^\perp)^\perp$. The proof only relies on equivalences hence a bi-inclusion is proved. \square

5.3. The case of multiplicative units. In this paper, we only mentioned MLL without units but linear logic is often presented with two formulas 1 and \perp corresponding to neutral elements for the \otimes and \wp connectives respectively. New rules and links for MLL units are presented in Figure 33.

Now, we look for behaviours corresponding to neutral elements for \otimes and \wp respectively. It is possible to define a pre-behaviour $\perp\!\!\!\perp$ called a *pole* [85, Definition 3.5] such that $\Phi \perp \Phi'$ if and only if $\text{Ex}(\Phi \uplus \Phi') \in \perp\!\!\!\perp$ for an execution-based orthogonality \perp and $\perp\!\!\!\perp$ must be closed under *anti-evaluation*, *i.e.* if $\Phi \in \perp\!\!\!\perp$ and $\text{Ex}(\Phi') = \Phi$, 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.

A natural choice of behaviour for the neutral element of \otimes *w.r.t.* \perp^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.

Proposition 5.24. *The pre-behaviour $\{\emptyset\}$ is a behaviour.*

Proof. A constellation of $\{\emptyset\}^\perp$ must self-normalise into the set of its roots since \emptyset has not effect when in interaction with another constellation. We have $\{\emptyset\}^\perp = \perp$. Now, a constellation $\Phi \in \perp^\perp$ is a constellation such that when it interacts with a constellation $\Phi' \in \perp^\perp$, we have $\text{Ex}(\Phi \uplus \Phi') \in \perp$. 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 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, $\perp^\perp = \{\emptyset\}^{\perp\perp} = \{\emptyset\}$. \square

Definition 5.25 (One). We define the behaviour $\mathbf{1} := \{\emptyset\} = \perp^\perp$.

Proposition 5.26. *We have $\mathbf{A} \otimes \mathbf{1} = \mathbf{A}$ for any behaviour \mathbf{A} .*

Proof. By definition, we have $\mathbf{A} \otimes \mathbf{1} = \{\Phi_A \uplus \emptyset \mid \Phi_A \in \mathbf{A}\}^{\perp\perp} = \{\Phi_A \mid \Phi_A \in \mathbf{A}\}^{\perp\perp} = \mathbf{A}^{\perp\perp} = \mathbf{A}$. \square

As for bottom, as usual in linear logic, we define it as $\mathbf{1}^\perp = \perp$.

Proposition 5.27. *The pre-behaviour $\mathbf{1}^\perp = \{\emptyset\}^\perp = \perp$ is a behaviour.*

Proof. Since it is known that $\mathbf{A}^\perp = \mathbf{A}^{\perp\perp}$ for any behaviour \mathbf{A} [67, Corollary 9], it follows that $\mathbf{1}^\perp$ (and thus $\{\emptyset\}^\perp$) is a behaviour. \square

Definition 5.28 (Bottom). We define the behaviour $\perp := \mathbf{1}^\perp$.

Proposition 5.29. *We have $\mathbf{A} \wp \perp = \mathbf{A}$ for any behaviour \mathbf{A} when considering \perp^R .*

Proof. We have $\mathbf{A} \wp \perp = (\mathbf{A}^\perp \otimes \perp^\perp)^\perp = (\mathbf{A}^\perp \otimes \{\emptyset\}^{\perp\perp})^\perp = (\mathbf{A}^\perp \otimes \{\emptyset\})^\perp = \mathbf{A}^{\perp\perp} = \mathbf{A}$ (since \mathbf{A} is a behaviour). \square

Proposition 5.30. *We have $\mathbf{A}^\perp = \mathbf{A} \multimap \perp$ for any behaviour \mathbf{A} when considering \perp^R .*

Proof. We have $\mathbf{A} \multimap \perp = \mathbf{A}^\perp \wp \perp$. Since \perp is a neutral element for \wp , it follows that $\mathbf{A}^\perp \wp \perp = \mathbf{A}^\perp$. \square

We defined interactive types for units which correspond to idealised neutral elements (Girard's *Usage*). Now, considering a constellation Φ in the wild, are we able to effectively tell whether it is in $\mathbf{1}$ (respectively \perp) or not? (Girard's *Usine*).

We consider \perp^R . In order to tell if $\Phi \in \perp$, we can use the fact that $\perp = \{\emptyset\}^\perp$. 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 $\Phi \in \perp$. As for $\mathbf{1}$, we just need to be able to tell if $\Phi = \emptyset$. This can be done with any constellation of $\mathbf{1}^\perp = \perp$.

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 either 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 usually to consider links (called “jumps” [47, Section A.2]) between \perp nodes and either axioms or 1 nodes to represent the dependency between \perp and its context. Girard's idea [56, Section 2.1.1] is to encode multiplicative units in second order linear logic because of this non-local dependency but we do not discuss it in this paper.

6. CONSERVATIVITY AND ADEQUACY

In this section, we propose two links:

- a proof of conservativity *w.r.t.* the original model of proof-nets for \perp^{fin} in order to capture MLL+MIX provability and for both \perp^R and \perp^1 in order to capture MLL provability. For that purpose, we state *soundness* and *completeness* theorems;
- a link between Usine and Usage (called *adequacy* by Girard) showing that the correctness criterion is sufficient to guarantee a sound use of proofs (interaction by cuts).

We interpret formula labels by behaviours where distinct behaviours are associated to occurrences of variables by a function called *basis of interpretation*. Following previous works of Seiller [98, 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 4.9) used to distinguish occurrences of atomic formulas. 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 conjugation.

Definition 6.1. A *basis of interpretation* is a function Ω producing a behaviour $\Omega(A, i, t)$ when given a formula $A \in \mathcal{F}_{\text{MLL}}$, a natural number i (index of occurrence) and a term $t \in \text{Addr}_x(\mathcal{S})$ (*cf.* Definition 4.9). A basis of interpretation has to satisfy the condition that $\Omega(A, i, t) + [+t.p_A(t), +t.p_B(u)] = \Omega(B, j, u)$ 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 .

Definition 6.2 (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 $p_A(x)$ (*cf.* Definition 4.9). We define the *interpretation* $\llbracket A, t \rrbracket_\Omega$ along Ω and a term t (encoding the address of A *w.r.t.* a conclusion C) inductively:

- $\llbracket C, X_i, t \rrbracket_\Omega = \Omega(C, i, t)$;
- $\llbracket C, X_i^\perp, t \rrbracket_\Omega = \Omega(C, i, t)^\perp$;
- $\llbracket C, A \otimes B, t \rrbracket_\Omega = \llbracket C, A, \mathbf{l} \cdot t \rrbracket_\Omega \otimes \llbracket C, B, \mathbf{r} \cdot t \rrbracket_\Omega$;
- $\llbracket C, A \wp B, t \rrbracket_\Omega = \llbracket C, A, \mathbf{l} \cdot t \rrbracket_\Omega \wp \llbracket C, B, \mathbf{r} \cdot t \rrbracket_\Omega$.

We write $\llbracket C \rrbracket$ for $\llbracket C, C, X \rrbracket$ and extend the interpretation to sequents with $\llbracket \vdash C_1, \dots, C_n \rrbracket_\Omega := \llbracket C_1 \rrbracket_\Omega \wp \dots \wp \llbracket C_n \rrbracket_\Omega$.

Remark 6.3. The interpretation of an axiom under an basis of interpretation Ω is defined by $\llbracket \vdash X_1, X_1^\perp \rrbracket_\Omega = \llbracket X_1 \rrbracket_\Omega \wp \llbracket X_1^\perp \rrbracket_\Omega = \Omega(X_1, \mathbf{l}, X) \wp \Omega(X_1^\perp, \mathbf{l}, X)^\perp$.

6.1. A complete model of MLL+MIX. We prove soundness and completeness for MLL+MIX. Theorem 4.31 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^{fin} exclusively.

Instead of the usual soundness property, we prove an extension called *full soundness* [98, 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 $\Phi : \Gamma$ implies $\llbracket \vdash \Gamma \rrbracket_\Omega$ for some basis of interpretation Ω , except that for $\Phi : \Gamma$ we only consider constellations coming from proof-nets.

Lemma 6.4. *Let A, B be MLL formulas, $\Gamma = C_1, \dots, C_n, \Delta = D_1, \dots, D_m$ be sets of MLL formulas and Ω be a basis of interpretation. We have $(\llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket A \rrbracket_\Omega) \otimes (\llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket B \rrbracket_\Omega) \subseteq \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket A \otimes B \rrbracket_\Omega$.*

Proof. The idea is to show $(\llbracket C_1^\perp \otimes \dots \otimes C_n^\perp \rrbracket_\Omega \multimap \llbracket A \rrbracket_\Omega) \otimes (\llbracket D_1^\perp \otimes \dots \otimes D_m^\perp \rrbracket_\Omega \multimap \llbracket B \rrbracket_\Omega) \subseteq (\llbracket C_1^\perp \otimes \dots \otimes C_n^\perp \rrbracket_\Omega \otimes \llbracket D_1^\perp \otimes \dots \otimes D_m^\perp \rrbracket_\Omega) \multimap \llbracket A \otimes B \rrbracket_\Omega$ which is equivalent to $(\llbracket C \rrbracket_\Omega \multimap \llbracket A \rrbracket_\Omega) \otimes (\llbracket D \rrbracket_\Omega \multimap \llbracket B \rrbracket_\Omega) \subseteq (\llbracket C \rrbracket_\Omega \otimes \llbracket D \rrbracket_\Omega) \multimap \llbracket A \otimes B \rrbracket_\Omega$ for $C := C_1^\perp \otimes \dots \otimes C_n^\perp$ and $D := D_1^\perp \otimes \dots \otimes D_m^\perp$. Assume we have two functions $\Phi_{C,A} \in \llbracket C \rrbracket_\Omega \multimap \llbracket A \rrbracket_\Omega$ and $\Phi_{D,B} \in \llbracket D \rrbracket_\Omega \multimap \llbracket B \rrbracket_\Omega$. We can construct their disjoint union $\Phi_{C,A} \uplus \Phi_{D,B} \in (\llbracket C \rrbracket_\Omega \multimap \llbracket A \rrbracket_\Omega) \otimes (\llbracket D \rrbracket_\Omega \multimap \llbracket B \rrbracket_\Omega)$. If we provide to $\Phi_{C,A} \uplus \Phi_{D,B}$ an argument $\Phi \in \llbracket C \rrbracket_\Omega \otimes \llbracket D \rrbracket_\Omega$, then since C and D are disjoint, each function $\Phi_{C,A}$ and $\Phi_{D,B}$ will take their argument separately and produce $\Phi' \in \llbracket A \otimes B \rrbracket_\Omega$. Therefore, $\Phi_{C,A} \uplus \Phi_{D,B} \in (\llbracket C \rrbracket_\Omega \otimes \llbracket D \rrbracket_\Omega) \multimap \llbracket A \otimes B \rrbracket_\Omega$. \square

Lemma 6.5. *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. \square

Lemma 6.6. *If \mathbf{A} is a behaviour and $\Phi \in \mathbf{A}$ then $|\text{Ex}(\Phi)| < \infty$.*

Proof. By definition of behaviour, we have $\mathbf{A} = \mathbf{A}^{\perp\perp}$. By Lemma 6.5 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' , 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. \square

Theorem 6.7 (Full soundness for MLL+MIX). *Let $\vdash \mathcal{S} : \Gamma$ be an MLL+MIX proof-net and Ω a basis of interpretation. We have $\text{Ex}(\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_{\mathcal{S}}^{\text{cut}}) \in \llbracket \vdash \Gamma \rrbracket_\Omega$.*

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 $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket X_i \rrbracket_\Omega \wp \llbracket X_i^\perp \rrbracket_\Omega = \llbracket X_i, X_i, X \rrbracket_\Omega \wp \llbracket X_i^\perp, X_i, X \rrbracket_\Omega^\perp = \Omega(X_i, i, X) \wp \Omega(X_i^\perp, i, X)^\perp = (\Omega(X_i, i, X)^\perp \otimes \Omega(X_i^\perp, i, X))^\perp$. 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 $|\text{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi_{\mathcal{S}}^{\text{ax}})| < \infty$, *i.e.* that the axiom strongly normalises with its tests. By Definition 6.1 since we have $\Phi_{\mathcal{S}}^{\text{ax}} = [+t.p_{X_i}(X), +t.p_{X_i^\perp}(X)]$, we have $\Phi_{\mathcal{S}}^{\text{ax}} \uplus \Phi_2 \in \Omega(X_i, i, X)$ which is orthogonal to Φ_1 . It follows that $|\text{Ex}(\Phi_1 \uplus \Phi_2 \uplus \Phi_{\mathcal{S}}^{\text{ax}})| < \infty$.
- 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 $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket A \otimes B \rrbracket_\Omega$. By induction hypothesis, we have $\Phi_{\mathcal{S}_1}^{\text{ax}} \in \llbracket \vdash \Gamma, A \rrbracket_\Omega = \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket A \rrbracket_\Omega$ and $\Phi_{\mathcal{S}_2}^{\text{ax}} \in \llbracket \vdash \Delta, B \rrbracket_\Omega = \llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket B \rrbracket_\Omega$. By a conjugation μ such that $\Phi_{\mathcal{S}_1}^{\text{ax}}$ and $\Phi_{\mathcal{S}_2}^{\text{ax}}$ are made distinct, we can relocate the atoms and obtain a constellation $\Phi_\mu \in (\llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket A \rrbracket_\Omega) \otimes (\llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket B \rrbracket_\Omega)$ such that $\Phi_\mu = \mu(\Phi_{\mathcal{S}_1}^{\text{ax}}) \uplus \Phi_{\mathcal{S}_2}^{\text{ax}}$. Now, by the definition of tensor for proof-structures, we have a preservation of axioms and $\Phi_{\mathcal{S}}^{\text{ax}}$ equivalent to Φ_μ up to conjugation (and this conjugation could be chosen for μ). By Lemma 6.4, we have $(\llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket A \rrbracket_\Omega) \otimes (\llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket B \rrbracket_\Omega) \subseteq \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket A \otimes B \rrbracket_\Omega$, hence $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket \vdash \Delta \rrbracket_\Omega \wp \llbracket A \otimes B \rrbracket_\Omega$.
- Assume we have $\vdash \mathcal{S} : \Gamma, A \wp B$ coming from $\vdash \mathcal{S}' : \Gamma, A, B$. We would like to show that $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket A \rrbracket_\Omega \wp \llbracket B \rrbracket_\Omega$. This directly follows from the induction hypothesis and the fact that we have $\llbracket \vdash \Gamma, A, B \rrbracket_\Omega = \llbracket \vdash \Gamma \rrbracket_\Omega \wp \llbracket A \rrbracket_\Omega \wp \llbracket B \rrbracket_\Omega$ 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 $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_{\Omega} \wp \llbracket \vdash \Delta \rrbracket_{\Omega}$ knowing that the induction hypothesis states that $\Phi_{\mathcal{S}_1}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_{\Omega}$ and $\Phi_{\mathcal{S}_2}^{\text{ax}} \in \llbracket \vdash \Delta \rrbracket_{\Omega}$. Since the MIX rule only places two proofs next to each other, we have $\Phi_{\mathcal{S}}^{\text{ax}} = \Phi_{\mathcal{S}_1}^{\text{ax}} \wp \Phi_{\mathcal{S}_2}^{\text{ax}}$ by definition. By definition of tensor, we have $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_{\Omega} \otimes \llbracket \vdash \Delta \rrbracket_{\Omega}$. It remains to show that $\mathbf{A} \otimes \mathbf{B} \subseteq \mathbf{A} \wp \mathbf{B}$ in general, which would imply $\Phi_{\mathcal{S}}^{\text{ax}} \in \llbracket \vdash \Gamma \rrbracket_{\Omega} \wp \llbracket \vdash \Delta \rrbracket_{\Omega}$. We have $\mathbf{A} \wp \mathbf{B} \subseteq (\mathbf{A}^{\perp} \otimes \mathbf{B}^{\perp})^{\perp}$, hence we have to show that $\mathbf{A} \otimes \mathbf{B} \subseteq (\mathbf{A}^{\perp} \otimes \mathbf{B}^{\perp})^{\perp}$. Let $\Phi_1 \wp \Phi_2 \in \mathbf{A} \otimes \mathbf{B}$ and $\Phi'_1 \wp \Phi'_2 \in \mathbf{A}^{\perp} \otimes \mathbf{B}^{\perp}$. We know that $\Phi_1 \perp \Phi'_1$ and $\Phi_2 \perp \Phi'_2$. We have $\Phi_1 \wp \Phi_2 \perp \Phi'_1 \wp \Phi'_2$, *i.e.* that $\Phi_1 \wp \Phi_2 \wp \Phi'_1 \wp \Phi'_2$ 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 6.6.

If the proof has cuts, then by Theorem 4.19, 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. \square

Lemma 6.8. *Let Ω be a basis of interpretation and $\vdash \Gamma$ an MLL sequent. Then, we have $\text{Tests}(\vdash \Gamma) \subseteq \llbracket \vdash \Gamma \rrbracket_{\Omega}^{\perp, \text{fin}}$.*

Proof. Assume we have $\text{Test}(\vdash \Gamma)^{\varphi} \in \text{Tests}(\vdash \Gamma)$ for a switching φ of $\vdash \Gamma$. The proof is done by induction on $\vdash \Gamma$.

- If $\Gamma = \{A_1, \dots, A_n\}$ 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 $\text{Ex}(\text{Test}(\vdash \Gamma)^{\varphi}) = \sum_{i=1}^n [-t.p_{A_i}(t_i), p_{A_i}(X)]$ where t_i is the expected encoding of the address of the atom A_i . We would like to show that $\text{Test}(\vdash \Gamma)^{\varphi} \in \llbracket \vdash A_1, \dots, A_n \rrbracket_{\Omega}^{\perp} = \llbracket A_1, A_1, t_1 \rrbracket_{\Omega}^{\perp} \otimes \dots \otimes \llbracket A_n, A_n, t_n \rrbracket_{\Omega}^{\perp}$. We show that $[-t.p_{A_i}(t_i), p_{A_i}(X)] \in \llbracket A_i, A_i, t_i \rrbracket_{\Omega}^{\perp}$. Let $\Phi_i \in \llbracket A_i, A_i, t_i \rrbracket_{\Omega}$. Because $\llbracket A_i, A_i, t_i \rrbracket_{\Omega}$ is a behaviour, we can use Lemma 6.6 and infer that $|\text{Ex}(\Phi_i)| < \infty$. Adding $[-t.p_{A_i}(t_i), p_{A_i}(X)]$ to a strongly normalising constellation cannot cause divergence, hence we must have $[-t.p_{A_i}(t_i), p_{A_i}(X)] \perp \Phi_i$ and $[-t.p_{A_i}(t_i), p_{A_i}(X)] \in \llbracket A_i, A_i, t_i \rrbracket_{\Omega}^{\perp}$. Now, since $\text{Test}(\vdash \Gamma)^{\varphi}$ is made of a disjoint union of constellations of $\llbracket A_i, A_i, t_i \rrbracket_{\Omega}^{\perp}$, it follows that $\text{Test}(\vdash \Gamma)^{\varphi} \in \llbracket \vdash A_1, \dots, A_n \rrbracket_{\Omega}^{\perp}$.
- If $\vdash \Gamma$ is $\vdash \Delta, A \wp B$, then a switching φ of $\vdash \Delta, A \wp B$ 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 \wp connective. By the induction hypothesis, we have $\text{Test}(\vdash \Delta, A, B)^{\bar{\varphi}} \in \llbracket \vdash \Delta, A, B \rrbracket_{\Omega} = \llbracket \vdash \Delta \rrbracket_{\Omega} \wp \llbracket A \rrbracket_{\Omega} \wp \llbracket B \rrbracket_{\Omega}$ and we would like to show that $\text{Test}(\vdash \Delta, A \wp B)^{\varphi} \in \llbracket \vdash \Delta, A \wp B \rrbracket_{\Omega} = \llbracket \vdash \Delta \rrbracket_{\Omega} \wp \llbracket A \wp B \rrbracket_{\Omega} = \llbracket \vdash \Delta \rrbracket_{\Omega} \wp \llbracket A \wp B, A, 1 \cdot X \rrbracket_{\Omega} \wp \llbracket A \wp B, B, r \cdot X \rrbracket_{\Omega}$. The constellation $\text{Test}(\vdash \Delta, A, B)^{\bar{\varphi}}$ uses terms $p_A(t)$ and $p_B(u)$ but when we add a \wp link between A and B , these terms are relocated relatively to the conclusion $A \wp B$ and we obtain $p_{A \wp B}(1 \cdot t)$ and $p_{A \wp B}(r \cdot u)$. Since they only differ by a conjugation, the two tests will react in the same way with respects to strong normalisation, *i.e.* $\text{Test}(\vdash \Delta, A, B)^{\bar{\varphi}} \in (\llbracket \vdash \Delta \rrbracket_{\Omega}^{\perp} \otimes \llbracket A \rrbracket_{\Omega}^{\perp} \otimes \llbracket B \rrbracket_{\Omega}^{\perp})^{\perp}$ implies $\text{Test}(\vdash \Delta, A \wp B)^{\varphi} \in (\llbracket \vdash \Delta \rrbracket_{\Omega}^{\perp} \otimes \llbracket A \wp B \rrbracket_{\Omega}^{\perp})^{\perp}$.
- 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 $\text{Test}(\vdash \Delta, A \otimes B)^{\varphi}$ can be defined from $\text{Test}(\vdash \Delta, A, B)^{\varphi}$ by removing the uncoloured rays $p_A(x)$ and $p_B(x)$, and adding new stars $[-c.p_A(\mathbf{g} \cdot X), -c.p_B(\mathbf{g} \cdot X), +c.p_{A \otimes B}(\mathbf{g} \cdot X)] + [-c.p_{A \otimes B}(\mathbf{g} \cdot X), p_{A \otimes B}(X)]$. One can show that $\llbracket \vdash \Delta, A \otimes B \rrbracket_{\Omega}$ is generated (in the sense of bi-orthogonal closure) by a pre-behaviour E , *i.e.* that $\llbracket \vdash \Delta, A \otimes B \rrbracket_{\Omega} = E^{\perp \perp}$ 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 5.15). 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 $\text{Tests}(\vdash \Delta, A, B) \subseteq \llbracket \vdash \Delta, A, B \rrbracket_{\Omega}^{\text{fin}}$, this shows the result since this implies that $\text{Test}(\vdash \Delta, A \otimes B)^{\varphi} \in E^{\text{fin}}$ and $\text{Test}(\vdash \Delta, A \otimes B)^{\varphi} \in E^{\perp\perp\perp} = \llbracket \vdash \Delta, A \otimes B \rrbracket_{\Omega}^{\perp}$ since it is known that $X^{\perp} = X^{\perp\perp\perp}$ in general for any pre-behaviour X [67, Corollary 9]. \square

Definition 6.9 (Proof-like constellation). The syntax tree $ST(\vdash \Gamma)$ of a sequent induces a set of rays by Definition 4.9 by computing the address of each atom in $ST(\vdash \Gamma)$. We note this set $\# \Gamma$. A constellation Φ is *proof-like w.r.t.* an MLL sequent $\vdash \Gamma$ if it is made of binary stars only and $\text{IdRays}(\Phi) = \# \Gamma$, i.e. it is a binary linking of atoms in Γ .

Example 6.10. A constellation which is proof-like w.r.t. $\vdash X_1^{\perp} \wp X_2^{\perp}, X_1 \otimes X_2$ is

$$[+c.p_{X_1^{\perp} \wp X_2^{\perp}}(\mathbf{l} \cdot X), +c.p_{X_1 \otimes X_2}(\mathbf{l} \cdot X)] + [+c.p_{X_1^{\perp} \wp X_2^{\perp}}(\mathbf{r} \cdot X), +c.p_{X_1 \otimes X_2}(\mathbf{r} \cdot X)].$$

However, even the wrong linking

$$[+c.p_{X_1^{\perp} \wp X_2^{\perp}}(\mathbf{l} \cdot X), +c.p_{X_1 \otimes X_2}(\mathbf{r} \cdot X)] + [+c.p_{X_1 \otimes X_2}(\mathbf{l} \cdot X), +c.p_{X_1 \otimes X_2}(\mathbf{r} \cdot X)]$$

is proof-like as well.

Theorem 6.11 (Completeness for MLL+MIX). *If a constellation $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}$ is proof-like w.r.t. $\vdash \Gamma$, then there exists an MLL+MIX proof-net $\vdash \mathcal{S} : \Gamma$ such that $\Phi = \Phi_{\mathcal{S}}^{\text{ax}}$.*

Proof. A proof-like constellation $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}$ 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 $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}$ we can use Lemma 6.8 and infer that for all switchings φ of $\vdash \Gamma$ (equivalently, of \mathcal{S}), $\text{Test}(\vdash \Gamma)^{\varphi} = \Phi_{\mathcal{S}}^{\text{test}(\varphi)} \perp \Phi$, excluding “wrong linking”. By Corollary 4.32, 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 . \square

6.2. A complete model of MLL. The soundness property actually holds for MLL with the same arguments as for MLL+MIX 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.

Theorem 6.12 (Full soundness for MLL). *Let $\vdash \mathcal{S} : \Gamma$ be an MLL proof-net and Ω a basis of interpretation. We have $\text{Ex}(\Phi_{\mathcal{S}}^{\text{ax}} \wp \Phi_{\mathcal{S}}^{\text{cut}}) \in \llbracket \vdash \Gamma \rrbracket_{\Omega}$.*

Proof. The idea of the proof is exactly the same as for Theorem 6.7. The only difference is in the axiom case. We need to show that $\text{Ex}(\Phi_1 \wp \Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}}) = \text{Roots}(\Phi_1 \wp \Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}})$ (respectively, $|\text{Ex}(\Phi_1 \wp \Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}})| = 1$). However, the properties of the basis of interpretation ensures that $\Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}}$ will be orthogonal to Φ_1 . Hence $\text{Ex}(\Phi_1 \wp \Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}}) = \text{Roots}(\Phi_1 \wp \Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}})$ (respectively, $|\text{Ex}(\Phi_1 \wp \Phi_2 \wp \Phi_{\mathcal{S}}^{\text{ax}})| = 1$). \square

The proof of Lemma 6.8 which is essential for the completeness property does not hold anymore because of a minor technical problem. This is because a general sequent $\vdash A_1, \dots, A_n$ 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

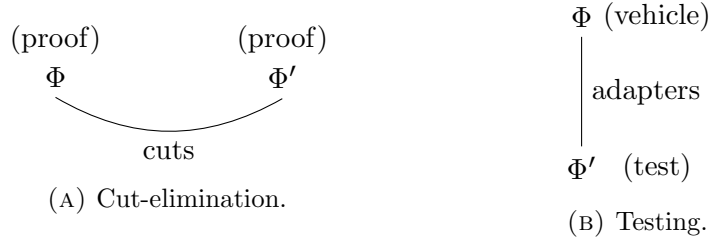


FIGURE 34. Cut-elimination seen as testing. The difference is only the point of view.

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 $\llbracket \vdash \Gamma \rrbracket_{\Omega}$ and a subset of $\mathbf{Tests}(\vdash \Gamma)^{\perp}$ corresponding to proof-structures, it is sufficient to identify $\llbracket \vdash \Gamma \rrbracket_{\Omega}$ and $\mathbf{Tests}(\vdash \Gamma)^{\perp}$ directly. We would then have to prove $\mathbf{Tests}(\vdash \Gamma) \subseteq \mathbf{Tests}(\vdash \Gamma)^{\perp\perp}$ which is always true in general [67, Proposition 7]. We do so by considering a notion of *strict interpretation*.

Definition 6.13 (Strict interpretations). We define the two strict interpretations for a given basis of interpretation Ω and an MLL sequent $\vdash \Gamma$:

$$\llbracket \vdash \Gamma \rrbracket_{\Omega}^1 = \mathbf{Tests}(\vdash \Gamma)^{\perp 1} \quad \text{and} \quad \llbracket \vdash \Gamma \rrbracket_{\Omega}^R = \mathbf{Tests}(\vdash \Gamma)^{\perp R}.$$

Theorem 6.14 (Completeness for MLL). *If a constellation $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}^R$ (respectively $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}^1$) 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 $\Phi = \Phi_{\mathcal{S}}^{\text{ax}}$.*

Proof. The proof begins like the proof of completeness for MLL+MIX (*cf.* Theorem 6.11) and reach the construction of a proof-structure with axioms translated into Φ . Now, $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}^R$ (respectively $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}^1$) implies that, in particular, Φ passes the Danos-Regnier correctness test for MLL (by Corollary 4.32). Therefore, the proof-structure we constructed must be correct. \square

Observation 6.15. Notice that if we have a constellation $\Phi \in \llbracket \vdash \Gamma \rrbracket_{\Omega}^X$ for some Ω , MLL sequent $\vdash \Gamma$ and $X \in \{1, R\}$, its Danos-Regnier tests Φ_1, \dots, Φ_n are constellations of $(\llbracket \vdash \Gamma \rrbracket_{\Omega}^X)^{\perp}$. This formalises the intuition in proof-nets that tests are sort of proofs of the dual.

6.3. Adequacy. Girard’s adequacy property [55, Section 4.4] is a way to relate type labels/formulas of Section 5.1 and behaviours of Section 5.2. In realisability interpretations [92, 84], this relation usually take the form of an *adequacy lemma* showing that type labels guarantee membership in some behaviour. The idea is that type labels have the same role as program specification and what we usually want is that passing some tests for a specification ensures that the program has the expected behaviour.

This adequacy actually corresponds to a cut-elimination theorem. This is because connecting two constellations with cuts can be seen as making two constellations interact with adapters, *i.e.* as testing a constellation against another constellation (*cf.* Figure 34).

It follows that the cut-elimination theorem (the fact we can eliminates all occurrences of cuts) states all possible interactions between our objects are sound. This is not always

the case in general: a cut on an axiom with proof-structures is ill-behaving. But if our objects have the *correct shaope* (passes the right tests) then all interaction must be sound.

Adequacy is therefore a direct consequence of full soundness (*cf.* Theorem 6.12) since:

- we are able to simulate cut-elimination for proof-structures (*cf.* Theorem 4.18);
- we can simulate proof-nets with constellations and the cut-elimination is known to hold for proof-nets.

7. PERSPECTIVES AND FUTURE WORKS

Alternative definitions of execution. In the simulation of Turing machines Section 3.2, we simulate runs by generating all the linear saturated and correct diagrams, which is an implicit reference to actual infinite. However, this is not a natural way to compute. It is actually possible to follow the usual computation of automata by traversing a dependency graph $\mathcal{D}[M^\star + w^\star]$ itself, seen as the state graph of a Turing machine reading an input word w . We then have to handle a unification problem when during the traversal. In case of error, the run is cancelled. In case of non-deterministic choice, we have parallel runs and only runs without unification errors survive. Although we do not explore this idea in this paper, any constellation seems to define a sort of generalised non-deterministic hypergraph automata with hyperedge transitions and graph-like runs where the fusion of stars triggers a propagation of information. This generalises various classes of automata (pushdown, alternating, Turing machines etc) and provide a machine-like execution of constellations. This would also generalise token machines of the geometry of interaction similarly to some existing works of the literature [24].

Categorical model of constellations. To ensure that we indeed have a model of MLL, it is possible to show that behaviours as categorical objects (providing we choose an orthogonality relation capturing MLL provability) and linear implications as arrows define a \star -autonomous category by following Seiller’s categorical model of interaction graphs [98, Section 3]. In particular, re-addressing of constellations has to be correctly treated in order to make composition of arrows possible. The associativity of execution (Theorem 5.20) is essential for the associativity of the composition and the property of adjunction (Corollary 5.22) ensures a monoidal closure. As for categorical interpretations of GoI with monoidal traced categories [58, 2], the execution of constellation should define a trace.

Extensions of other fragments of linear logic. We defined a model for both MLL and MLL+MIX but our (re)construction can be extended to other fragments of linear logic. In the first paper of Transcendental Syntax [55, Section 5], a reconstruction of intuitionistic exponentials for linear logic is sketched with a new correctness criterion.

The transcendental syntax also claims great improvements going beyond linear logic by suggesting in particular a new computational interpretation for second order logic and predicate calculus [54, Section 5] (which is seen as part of second order logic) but also Peano arithmetic [57, Section 5]. Second order logic uses more powerful constellations using internal colours within rays (*e.g.* $+c(+d(x))$) which adds a more complex combinatorics to constellations. This defines two classes of constellations: the ones of this paper, called *objective*, containing no internal colours and the ones with internal colours, called *subjective*. These two classes of constellations will allow for the definition of a non-empty behaviour $\mathbf{0}$

[54, Section 4.3] from which we can establish non-classical notions of truth [57, Section 3] but also state coherence, *i.e.* that $\mathbf{0}$ has no correct constellations.

Applications to implicit computational complexity. Several authors [11, 9, 15] used flows (which can be seen as binary stars) for an implicit complexity analysis using encodings of automata. Since the stellar resolution is designed to be an extension of this model, we can expect to capture other complexity classes. It would then be possible to extend Seiller’s idea of computational complexity [102] where the concepts of orthogonality and test are central.

Moreover, a reconstruction of predicate calculus may provide a new understanding to descriptive complexity [66] results, such as Immerman-Vardi theorem [109, 65] where predicate calculus is essential in the definition of complexity classes such as P and NP. The idea is that if a complexity class C is captured by a logic, then in our framework, formulas would correspond to set of programs bounded by a certain complexity and correctness criteria could also be used to check the complexity of a program (constellation). Formulas would then represent specification certifying complexity for a constellation seen as a program.

Formulas as specification for a computational behaviour. In model checking, the representation of a system (usually an automata or a transition system) is verified automatically in order to check if it satisfies a formula (typically in the temporal logic LTL [88]). Since the stellar resolution can provide natural encoding of state machines or labelled transition systems, and that it is possible to design logics and formulas with the transcendental syntax (by using both interactive typing and tests of type labels), we can imagine extensions of model checking to other models of computation or to other (existing or designed) logics capturing more fine-grained properties. The correctness criterion of linear logic can itself be seen as a way to certify a computational object as a proof of some formula A corresponding to a specification. Since the λ -calculus can be nicely encoded with proof-nets, higher order model checking for λ -calculus may be investigated as well.

Towards a “Logic of Interaction?”. Inspired by computation and linear logic, authors such as Curien [22] and Abramsky [2, Section 5] exposed a paradigm of interaction where the notion of interaction would be central in logic before anything else. Similarly to how complex behaviours arise from a system of interacting agents as in biology or more generally, the theory of complex systems (see chemical reaction networks for instance [70]), can a notion of logic emerge from any system of interactions?

Our stellar resolution is indeed an instance of a system of interacting agents (stars) from which emerge complex concepts (proofs and formulas). But even beyond logic, constellations are able to represent automata and other models of computation as interacting agents transmitting information in a graph-like structure and, by interactive typing, it is possible to design formulas describing their computational behaviour. Logic then appears as a way to describe computation or put constraint on it. This opens the materialistic idea of a “logic of things” analysing computational interactions.

REFERENCES

- [1] Samson Abramsky. Information, processes and games. *J. Benthem van & P. Adriaans (Eds.), Philosophy of Information*, pages 483–549, 2008.

- [2] Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.
- [3] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *The Journal of Symbolic Logic*, 59(2):543–574, 1994.
- [4] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for pcf. *Information and Computation*, 163(2):409–470, 2000.
- [5] Beniamino Accattoli. Proof nets and the linear substitution calculus. In *International Colloquium on Theoretical Aspects of Computing*, pages 37–61. Springer, 2018.
- [6] Roberto Amadio. *Operational methods in semantics*. PhD thesis, Univeristé Denis Diderot Paris 7, 2016.
- [7] Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ -calculus. *Theoretical Computer Science*, 142(2):277–297, 1995.
- [8] Clément Aubert and Marc Bagnol. Unification and logarithmic space. In *Rewriting and Typed Lambda Calculi*, pages 77–92. Springer, 2014.
- [9] 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.
- [10] Clément Aubert and Thomas Seiller. Characterizing co-nl by a group action. *Mathematical Structures in Computer Science*, 26(4):606–638, 2016.
- [11] Clément Aubert and Thomas Seiller. Logarithmic space and permutations. *Information and Computation*, 248:2–21, 2016.
- [12] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.
- [13] 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.
- [14] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.
- [15] Patrick Baillot and Marco Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2):1–31, 2001.
- [16] 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.
- [17] Andreas Blass. A game semantics for linear logic. *Annals of Pure and Applied logic*, 56(1-3):183–220, 1992.
- [18] Alain Bretto. Hypergraph theory. *An introduction*. *Mathematical Engineering*. Cham: Springer, 2013.
- [19] Junghuei Chen and Nadrian C Seeman. Synthesis from dna of a molecule with the connectivity of a cube. *Nature*, 350(6319):631–633, 1991.
- [20] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.
- [21] Alonzo Church. On the concept of a random sequence. *Bulletin of the American Mathematical Society*, 46(2):130–135, 1940.
- [22] P-L Curien. Symmetry and interactivity in programming. *Bulletin of Symbolic Logic*, pages 169–180, 2003.
- [23] Haskell B Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584, 1934.
- [24] 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.
- [25] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)*. PhD thesis, Paris 7, 1990.
- [26] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989.
- [27] Vincent Danos and Laurent Regnier. Proof-nets and the hilbert space. *London Mathematical Society Lecture Note Series*, pages 307–328, 1995.
- [28] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal λ -machines. *Theoretical Computer Science*, 227(1-2):79–97, 1999.

- [29] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425, 2001.
- [30] Martin Davis. Computability and unsolvability. 1982 ed, 1958.
- [31] Paulin Jacobé De Naurois and Virgile Mogbil. Correctness of linear logic proof structures is nl-complete. *Theoretical Computer Science*, 412(20):1941–1957, 2011.
- [32] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409, 2003.
- [33] Edsger W Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001.
- [34] Michael Dummett. *The logical basis of metaphysics*. Harvard university press, 1991.
- [35] Norbert Eisinger and Hans Jürgen Ohlbach. Deduction systems based on resolution. 1991.
- [36] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web International Summer School*, pages 40–110. Springer, 2009.
- [37] Arnaud Fleury and Christian Retoré. The mix rule. *Mathematical Structures in Computer Science*, 4(2):273–285, 1994.
- [38] Michael Gelfond. Answer sets. *Foundations of Artificial Intelligence*, 3:285–316, 2008.
- [39] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.
- [40] Gerhard Gentzen. Untersuchungen über das logische schließen. ii. *Mathematische Zeitschrift*, 39(1):405–431, 1935.
- [41] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.
- [42] Jean-Yves Girard. Geometry of interaction II: deadlock-free algorithms. In *International Conference on Computer Logic*, pages 76–93. Springer, 1988.
- [43] Jean-Yves Girard. Normal functors, power series and λ -calculus. *Annals of pure and applied logic*, 37(2):129–177, 1988.
- [44] 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.
- [45] Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.
- [46] Jean-Yves Girard. Geometry of interaction III: accommodating the additives. *London Mathematical Society Lecture Note Series*, pages 329–389, 1995.
- [47] Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. *Lecture Notes in Pure and Applied Mathematics*, pages 97–124, 1996.
- [48] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical structures in computer science*, 11(3):301, 2001.
- [49] Jean-Yves Girard. Geometry of interaction IV: the feedback equation. In *Logic Colloquium*, volume 3, pages 76–117, 2006.
- [50] Jean-Yves Girard. Geometry of interaction V: logic in the hyperfinite factor. *Theoretical Computer Science*, 412(20):1860–1883, 2011.
- [51] Jean-Yves Girard. Geometry of interaction VI: a blueprint for transcendental syntax. *preprint*, 2013.
- [52] Jean-Yves Girard. Three lightings of logic (invited talk). In *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- [53] Jean-Yves Girard. Transcendental syntax II: non-deterministic case. 2016.
- [54] Jean-Yves Girard. Transcendental syntax III: equality. 2016.
- [55] Jean-Yves Girard. Transcendental syntax I: deterministic case. *Mathematical Structures in Computer Science*, 27(5):827–849, 2017.
- [56] Jean-Yves Girard. La logique 2.0. 2018.
- [57] Jean-Yves Girard. Transcendental syntax IV: logic without systems. 2020.
- [58] Esfandiar Haghverdi and Philip Scott. A categorical model for the geometry of interaction. *Theoretical Computer Science*, 350(2-3):252–274, 2006.
- [59] Shawn Hedman. *A First Course in Logic: An introduction to model theory, proof theory, computability, and complexity*. OUP Oxford, 2004.
- [60] Jacques Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, Université de Paris, 1930.
- [61] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

- [62] 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.
- [63] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.
- [64] J Martin E Hyland and C-HL Ong. On full abstraction for pcf: I, ii, and iii. *Information and computation*, 163(2), 2000.
- [65] 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](https://doi.org/10.1016/S0019-9958(86)80029-8).
- [66] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.
- [67] 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.
- [68] 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.
- [69] Nataša Jonoska and Gregory L McColm. Flexible versus rigid tile assembly. In *International Conference on Unconventional Computation*, pages 139–151. Springer, 2006.
- [70] Jürgen Jost and Raffaella Mulas. Hypergraph laplace operators for chemical reaction networks. *Advances in mathematics*, 351:870–896, 2019.
- [71] Robert Kowalski. Predicate logic as programming language. In *IFIP congress*, volume 74, pages 569–544, 1974.
- [72] Robert Kowalski. A proof procedure using connection graphs. *Journal of the ACM (JACM)*, 22(4):572–595, 1975.
- [73] Robert Kowalski and Donald Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3-4):227–260, 1971.
- [74] JL Krivine, PL Curien, H Herbelin, and PA Melliès. Interactive models of computation and program behavior. 2009.
- [75] Yves Lafont. From proof nets to interaction nets. *London Mathematical Society Lecture Note Series*, pages 225–248, 1995.
- [76] J-L Lassez, Michael J Maher, and Kim Marriott. Unification revisited. In *Foundations of logic and functional programming*, pages 67–113. Springer, 1988.
- [77] 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.
- [78] Alexander Leitsch. *The resolution calculus*. Springer Science & Business Media, 2012.
- [79] Jorge Lobo, Arcot Rajasekar, and Jack Minker. Semantics of horn and disjunctive logic programs. *Theoretical Computer Science*, 86(1):93–106, 1991.
- [80] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.
- [81] Antoni Mazurkiewicz. Basic notions of trace theory. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 285–363. Springer, 1988.
- [82] Paul-André Melliès. Categorical semantics of linear logic. *Panoramas et synthèses*, 27:15–215, 2009.
- [83] Jack Minker. Overview of disjunctive logic programming. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):1–24, 1994.
- [84] Alexandre Miquel. De la formalisation des preuves à l’extraction de programmes. *HdR thesis, Université Paris*, 7, 2009.
- [85] Étienne Miquey. *Classical realizability and side-effects*. PhD thesis, Université Sorbonne Paris Cité-Université Paris Diderot (Paris 7 . . . , 2017.
- [86] Andrzej S Murawski and C-HL Ong. Dominator trees and fast verification of proof nets. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332)*, pages 181–191. IEEE, 2000.
- [87] Matthew J Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.
- [88] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. iee, 1977.
- [89] Laurent Regnier. *Lambda-calcul et réseaux*. PhD thesis, Paris 7, 1992.
- [90] Christian Retoré. Handsome proof-nets: perfect matchings and cographs. *Theoretical Computer Science*, 294(3):473–488, 2003.

- [91] Colin Riba. Strong normalization as safe interaction. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 13–22. IEEE, 2007.
- [92] Lionel Rieg. *On forcing and classical realizability*. PhD thesis, Ecole normale supérieure de lyon-ENS LYON, 2014.
- [93] John Alan Robinson et al. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
- [94] Andrea Schalk. What is a categorical model of linear logic. *Manuscript, available from <http://www.cs.man.ac.uk/schalk/work.html>*, 2004.
- [95] Kurt Schütte. Ein system des verknüpfenden schliessens. *Archiv für mathematische Logik und Grundlagenforschung*, 2(2):55–67, 1956.
- [96] Robert AG Seely. *Linear logic, *-autonomous categories and cofree coalgebras*. Ste. Anne de Bellevue, Quebec: CEGEP John Abbott College, 1987.
- [97] Nadrian C Seeman. Nucleic acid junctions and lattices. *Journal of theoretical biology*, 99(2):237–247, 1982.
- [98] Thomas Seiller. Interaction graphs: multiplicatives. *Annals of Pure and Applied Logic*, 163(12):1808–1837, 2012.
- [99] Thomas Seiller. Interaction graphs: additives. *Annals of Pure and Applied Logic*, 167(2):95–154, 2016.
- [100] 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.
- [101] Thomas Seiller. Interaction graphs: Graphings. *Annals of Pure and Applied Logic*, 168(2):278–320, 2017.
- [102] Thomas Seiller. Interaction graphs: Non-deterministic automata. *ACM Transactions on Computational Logic (TOCL)*, 19(3):1–24, 2018.
- [103] Thomas Seiller. Interaction graphs: Exponentials. *Logical Methods in Computer Science*, 15, 2019.
- [104] Thomas Seiller. Probabilistic complexity classes through semantics. *arXiv preprint [arXiv:2002.00009](https://arxiv.org/abs/2002.00009)*, 2020.
- [105] Sharon Sichel. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers*, (8):823–835, 1976.
- [106] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.
- [107] Sten-Åke Tärnlund. Horn clause computability. *BIT Numerical Mathematics*, 17(2):215–226, 1977.
- [108] Wolfgang Thomas. On logics, tilings, and automata. In *International Colloquium on Automata, Languages, and Programming*, pages 441–454. Springer, 1991.
- [109] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.
- [110] Hao Wang. Proving theorems by pattern recognition —II. *Bell system technical journal*, 40(1):1–41, 1961.
- [111] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, Citeseer, 1998.
- [112] 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.
- [113] Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable dna self-assembly. *Nature*, 567(7748):366–372, 2019. doi:10.1038/s41586-019-1014-9.

APPENDIX A. TERM UNIFICATION AND FIRST-ORDER RESOLUTION

We recall elementary definitions of term unification [60] and first-order resolution [93]. We refer the reader to the article of Lassez et al. [76] for more details which are often omitted in the literature or Baader et al. [12] for a broader view.

We will use uppercase letters such as X, Y, Z for variables and lowercase letters a, b, c, f, g and h for function symbols.

- ▷ A *signature* $\mathbb{S} = (V, F, \mathbf{ar})$ consists of a countable set V of variables, a countable set F of function symbols whose arities are given by $\mathbf{ar} : F \rightarrow \mathbf{N}$. We set a signature for this section.
- ▷ The set of *terms* $\mathbf{Terms}(\mathbb{S})$ is inductively defined by the following grammar:

$$t, u ::= X \mid f(t_1, \dots, t_n) \quad X \in V, f \in F, \mathbf{ar}(f) = n \quad (\text{Terms})$$

- ▷ A *substitution* is a function $\theta : V \rightarrow \mathbf{Terms}(\mathbb{S})$. Substitutions are extended from variables to terms by $\theta(f(u_1, \dots, u_k)) = f(\theta(u_1), \dots, \theta(u_k))$. The substitution θt is often written $\theta(t)$ or explicitly as a set of associations $\{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$ (often written $\{x := t\}$ when there is only one association).

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 [76, Corollary 6].

- ▷ A *renaming* is a substitution α such that $\alpha(X) \in V$ for all $X \in V$.
- ▷ An *equation* is an unordered pair $t \stackrel{?}{=} u$ of terms in $\mathbf{Terms}(\mathbb{S})$.
- ▷ A *unification problem* or simply *problem* is a set of equations $P = \{t_1 \stackrel{?}{=} u_1, \dots, t_n \stackrel{?}{=} u_n\}$.
- ▷ A *solution* 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 is a *unifier* for them.
- ▷ Two terms t and u are α -unifiable if there exists a renaming α such that $\{\alpha t \stackrel{?}{=} u\}$ has a solution which is called α -unifier. An α -unification between two terms is *exact* when it is a renaming.
- ▷ Two terms t, u are *equivalent up to renaming*, written $t \approx_\alpha u$, if there exists an exact α -unifier between them.

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.

Definition A.1 (Order on terms). We define the following partial order: given t, u two terms, $t \leq u$ if and only if there exists a substitution θ such that $t = \theta u$. We consider the order up to renaming, *i.e.* $t = u$ when $t \approx_\alpha u$.

Proposition A.2. *The relation \leq 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 $(\theta_a \circ \theta_b)t_3 = \theta_a(\theta_b t_3) = \theta_a t_2 = t_1$. \square

Our definition of α -unification comes from a simplification of Aubert and Bagnol's definition of *matching* [8, Definition 6] itself appearing in Girard's definitions [52, Section 1.1.2]. However, since *matching* already exists with a different definition in the literature we chose a different name. We show that our simplification is equivalent to Aubert and Bagnol's definition definition.

Proposition A.3. *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 $\mathbf{vars}(\alpha_1 t_1) \cap \mathbf{vars}(\alpha_2 t_2) = \emptyset$.*

Proof. (\Rightarrow) Assume that t_1 and t_2 are α -unifiable. Hence, there exists α such that $\theta \alpha t_1 = \theta t_2$ for some substitution θ . For $\alpha_1 := \alpha$ and $\alpha_2 := \emptyset$, the empty renaming which is indeed disjoint from α , we have $\theta \alpha_1 t_1 = \theta \alpha_2 t_2$. (\Leftarrow) Now 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 $\mathbf{vars}(\alpha_1 t_1) \cap \mathbf{vars}(\alpha_2 t_2) = \emptyset$. We have

$\theta\alpha_1t_1 = \theta\alpha_2t_2$ 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 $\psi\alpha t_1 = \psi t_2$ since we have $\theta\alpha_1t_1 = \theta\alpha_2t_2$. This shows that t_1 and t_2 are α -unifiable. \square

The problem of deciding if a solution to a given problem P exists is known to be decidable in the literature. Moreover, there exists a maximal solution $\mathbf{solution}(P)$ w.r.t. the preorder \leq , 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 [80].

Definition A.4 (Solved form). A unification problem $P = \{X_1 \stackrel{?}{=} t_1, \dots, X_n \stackrel{?}{=} t_n\}$ with $X_1, \dots, X_n \in V$ is in *solved form* if no variable X_1, \dots, X_n appears on the right of an equation, i.e. $\{X_1, \dots, X_n\} \cap \bigcup_{j=1}^n \mathbf{fv}(t_j) = \emptyset$. Its *underlying substitution* is defined by $\vec{P} := \{X_1 \mapsto t_1, \dots, X_n \mapsto t_n\}$.

Definition A.5 (Martelli-Montanari unification algorithm). We define a non-deterministic algorithm with inference rules read from top to bottom:

$$\begin{array}{c}
 \frac{P \cup \{t \stackrel{?}{=} t\}}{P} \text{ clear} \quad \frac{P \text{ (in solved form)}}{\text{success}} \quad \frac{P \text{ (not in solved form)}}{\perp} \text{ fail} \\
 \\
 \frac{P \cup \{f(t_1, \dots, t_n) \stackrel{?}{=} f(u_1, \dots, u_n)\}}{P \cup \{t_1 \stackrel{?}{=} u_1, \dots, t_n \stackrel{?}{=} u_n\}} \text{ open} \quad \frac{P \cup \{t \stackrel{?}{=} X\} \text{ with } t \notin \mathbf{vars}(t)}{P \cup \{X \stackrel{?}{=} t\}} \text{ orient} \\
 \\
 \frac{P \cup \{X \stackrel{?}{=} t\} \text{ with } X \in \mathbf{vars}(P) \text{ and } X \notin \mathbf{vars}(t)}{\{X \mapsto t\}P \cup \{X \stackrel{?}{=} t\}} \text{ replace}
 \end{array}$$

where $\mathbf{vars}(P)$ and $\mathbf{vars}(t)$ are the sets of variables occurring in P and t . A sequence constructed by these rules and ending with a success or fail rule when no other rule can be applied is called an *execution* of the unification algorithm. The last step of the sequence is written $\mathbf{solution}(P)$ for a problem P . If we can apply more rules, it is called a *partial execution*.

Theorem A.6 (Confluence of the unification algorithm). *Any solvable problem P has a unique solution modulo \approx_α .*

Proof. Proven in [76, Theorem 3.17]. \square

We complete the section with the resolution rule for first-order logic for which the unification is central.

Definition A.7 (Resolution rule). Let C, D be two set of first-order atoms and P a predicate of arity n . The *resolution rule* is defined by the following inference rule:

$$\frac{C \cup \{P(t_1, \dots, t_n)\} \quad D \cup \{\neg P(u_1, \dots, u_n)\}}{\theta(C \cup D)} \text{ Res}$$

where $\theta := \mathbf{solution}(\mathcal{P}(\bigcup_{k=1}^n \{t_i \stackrel{?}{=} u_i\}))$.

APPENDIX B. HYPERGRAPH THEORY

Hypergraphs are graphs with edges potentially linking several vertices and thus generalise graphs. We use a definition of directed hypergraphs with several targets (we usually consider a single target in the literature [18]).

Definition B.1 (Directed hypergraph). An *directed hypergraph* is defined by a tuple $H = (V, E, \text{in}, \text{out})$ where:

- V and E are respectively the set of vertices and hyperedges;
- $\text{in} : E \rightarrow P(V)$ defines the sources of a hyperedge;
- $\text{out} : E \rightarrow P(V)$ defines the targets of a hyperedge

where $P(V)$ is the powerset of V . The hypergraph is *undirected* when we forget out and in defines the vertices connected by an hyperedge.

Definition B.2 (Disjoint union of hypergraphs). The *disjoint union* of two hypergraphs $H = (V, E, \text{in}, \text{out})$ and $H' = (V', E', \text{in}', \text{out}')$ is defined by $H \uplus H' := (V \uplus V', E \uplus E', \text{in} \uplus \text{in}', \text{out}' \uplus \text{out}')$ where \uplus is the disjoint union of sets and $f \uplus f'$ for two functions f and f' is the function corresponds to the disjoint union of the function graph of f and f' .

Definition B.3 ((Multi)Graph homomorphism). A *(multi)graph homomorphism* is a function between graphs $f : G \rightarrow G'$ preserving the structure of graph, *i.e.* for all $(u, v) \in E^G$, we have $(f(u), f(v)) \in E^{G'}$.

Definition B.4 ((Multi)Graph isomorphism). A (multi)graph homomorphism f is an *isomorphism* if it is a bijection and that its inverse f^{-1} is also a (multi)graph homomorphism.

Definition B.5 (Path). A *(directed) path* is an ordered alternated sequence of vertices and hyperedges $v_1 e_1 \dots v_n, e_n, v_{n+1}$ for $v_i \in V$ and $e_i \in E$.

For an undirected graph, the path has an unordered sequence.

Definition B.6 (Accessibility relation). The notion of path defines an *accessibility relation* on vertices for a hypergraph H . For two vertices x and y , we have $x \equiv_{\text{acc}}^H y$ when there is a path from x to y and, if the edges are oriented/directed, from y to x as well.

Definition B.7 (Connected components and connectedness). The set of *connected components* of a hypergraph is the set of equivalence classes defined by the equivalence relation induced by the accessibility of vertices with paths. A hypergraph is connected when it has a unique connected component.

Definition B.8 (Cycles and acyclicity). A *cycle* is a path $v_1 e_1 \dots v_n, e_n, v_{n+1}$ such that $v_1 = v_{n+1}$. A hypergraph is *acyclic* when it has no cycle.

Notation B.9. For a (multi)graph or hypergraph G of vertices V and (hyper)edges E , we write V^G for V and E^G for E .