A gentle introduction to Girard's Transcendental Syntax for the linear logician

Boris Eng

A gentle introduction to Girard's Transcendental Syntax for the linear logician

Version 6

Boris Eng

The Transcendental Syntax is about designing logics with a computational foundation. It suggests a new framework for proof theory where logic (proofs, formulas, truth, ...) is no more primitive but computation is. All the logical entities and activities will be presented as formatting on a given model of computation which should be as general, simple and natural as possible. The selected ground for logic in the Transcendental Syntax is a model of computation I call "stellar resolution" which is basically a reformulation logic-free Robinson's first order clausal resolution with a dynamics related to tiling models. An initial goal of the Transcendental Syntax is to retrieve linear logic from this new framework In particular, this model naturally encodes cut-elimination for proof-structures. By using ideas from realisability theory, it is possible to design formulas/types generalising the connectives of linear logic. Finally, generalising ideas from the theory of proof-nets, we able to express an idea unit testing: we can encode a computational object (typically the computational content of a proof) and some tests certifying its logical correctness (that it is indeed a valid proof of some formula).

A friendly glossary of the Transcendental Syntax and of Girard's terminology is presented at the end of this paper.

Prerequisites: linear logic, sequent calculus, proof-nets, correctness criteria, term unification, first order resolution.

Contents

1. Introduction

1.1. History and scientific context

The Transcendental Syntax is the direct successor of Girard's *Geometry of Interaction* (GoI) [17, 16, 15, 18, 20, 21]. The initial idea of GoI was to study cut-elimination in a purely computational and mathematical way. For instance, in the case of MLL, Girard remarked [14] that it was sufficient to represent proofs as permutations on a finite subset $S \subseteq \mathbf{N}$ of natural numbers and cut-elimination as a procedure of interaction between permutations[1]. Then if we would like to extend to exponentials in order to speak about the erasure and duplication of formulas, we need more complex objects (infinitely many ones for the potentially infinite copies). This is why Girard introduced unusual interpretations with operator algebras [16]. The next articles are extensions of this idea to full linear logic and more. The current best historical introduction is Seiller's thesis [39] (written in French).

Unsatisfied with the infinite and complicated objects obtained, Girard introduced the Transcendental Syntax as a successor which would use simple and finite combinatorics, by extending a simplification of an idea of the third paper of GoI [18]. Philosophical motivations such as the wish for a finite and tractable reasoning also appeared along this new project. Moreover, inspired by the GoI, logic would not be primitive anymore: instead, we should start from a very general idea of computation and then a logic would be designed for this model. In some sense, it is opposite to the practice of the proof-programme correspondence: instead of taking a logic and turning it into a type system for a new programming language, we design a logic for a programming language which would speak about its computational behaviour. I personally like to see it as a sort of *reverse engineering* of logic where logic would be *designed* instead of *defined*.

Note that although the GoI has a quite rich history, it was a rather isolated series of works. In the literature, the term "Geometry of Interaction" usually refers to a simplification of Girard's first GoI [16] suggested by Danos and Regnier [12] which led to the idea of *token machine* [13, 4] used in rewriting-free evaluations of λ-terms and which also inspired categorical semantics of linear logic [2, 25].

1.2. Transcendental Syntax as an instance of realisability (optional)

In this section, I assume that you have (even vaguely) heard about classical realisability.

Although never explicitly mentioned by Girard, the Transcendental Syntax is an instance of the idea of *realisability*. In classical realisability theory [32, 37] for instance, a formula A interpreted by a set of λ-terms and its negation $A^{\perp}$ by a set of stacks. The evaluation is given by an interaction between a term and a stack by the Krivine Abstract Machine (KAM). By extending the model of computation, it is possible to capture more logical operations. For instance, in Krivine's realisability an operator `cc` is added to the usual λ-calculus in order to speak about classical logic and axioms of set theory.

[1] This idea has been simplified with graph theory by Seiller [38]

Although very similar, the Transcendental Syntax differs from realisability theory in few points:

- first, by a difference of goal. Realisability theory does not aim at characterising properties or behaviour of a model of computation. It is more focussed on logic (realising formulas) while we are focussed on computation (describing computation).
- In our case, the set of computational objects is not divided in two (terms and stacks). In particular, objects of A and $A^{\perp}$ are of the same kind.
- We are able to express linear logic (cut-elimination, proof-nets, logical correctness, connectives and formulas). There are other parallel approaches such as Beffara's concurrent realisability [9] but it seems that it cannot capture conveniently full linear logic and logical correctness.

1.3. Philosophical motivations

In this section, I assume that the reader is already (even vaguely) aware of Girard's intuition and conception of logic without necessarily understanding them.

A quite old very recurrent idea of Girard is the idea of *blind spot*: the idea that our understanding of logic is biased and only an approximation of something bigger we do not see. Indeed, the whole study of logic usually lies on primitive definitions and preconception of what a proof or a formula is. A reductionist approach is then taken in order to explore the mechanisms of these given concepts.

The Transcendental Syntax claims that writing a simple and naive proof in sequent calculus is not so innocent. The theory of proof-nets shows that when doing so, what is hidden is a computational object together with an implicit certificate asserting that the computational object is logically correct. Proofs are like individuals having a VIP card[2] but we forget that some people do not have it. What Girard claims is that these forgotten individuals are essential in the understanding of the mechanisms of logic. An ambition of the Transcendental Syntax is making implicit assumptions computationally explicit and to "put everything on the table".

In terms of linear logic, this idea is materialised by the correctness criterion of linear logic. In the theory of proof-nets, a computational object called proof-structures can be tested in order to tell whether or not it is logically correct (that it corresponds to a sequent calculus proof). In the Transcendental Syntax, this idea is generalised beyond proof-structures: any computational object is called "proof" when it is accompanied with some tests it satisfies. The idea of logical correctness then becomes relative/subjective. It means that what we mean by "passing the tests" is up to us, depending on what we would like to obtain (for instance, a specific computational behaviour).

Girard's philosophical motivation for the Transcendental Syntax is to establish a whole new architecture for logic which would be free of any logical preconception. Logic would be then completely emerging from computation. Everything starts from a chosen very

[2] or "sanitary pass" if you prefer.

general, simple and natural model of computation. This is what Girard calls the *analytics*, in reference to Kant. We require that the model includes reducible objects which can be evaluated (what Girard calls *performance*) to an irreducible object (Girard's *constat*). Both are separated by *indecidability*: the fact that an reducible object cannot always be reduced to its result (infinite loop may appear).

Once we have a computational ground, logic is a subjective way to provide a meaning to the computational objects, what Girard calls *synthetics*. There are two ways to make a computation meaningful. First, defining an arbitrary finite set of tests (as we would do in programming) and see if the object of interest passes these tests. Interestingly, these tests are also encoded as objects of the same kind as the tested and are part of the type theory of Transcendental Syntax instead of being external objects. This is Girard's *usine* (factory in English). It is like checking if an individual satisfies some stereotypes. Another way to provide meaning is to classify objects depending on how they interact with other ones. This is Girard's *usage* (use in English). Girard claims that the factory and the use are separated by *Gödel's incompleteness theorems* because tests (our arbitrary definitions of logic) cannot always perfectly guarantee the use of logical definitions. The tests are either too strict (and missing some use cases – or some true formulas becoming unprovable) or too permissive (thus contradictory).

1.4. Transcendental Syntax as a toolbox for computer science

The idea of capturing computational properties of behavour by formulas (synthetic and syntactic descriptions) is not new the computer science. Formulas are convenient ways to reason about general phenomena by only manipulating labels. The Transcendental Syntax generalise this practice of formulas in a larger and convenient framework.

- In *descriptive complexity* [26], complexity classes such as P and NP are captured by fragments of a logic and decision problems are characterised by formulas. In our case, we should be able to design atypical formulas existing outside a predefined logical system in order to describe computation in a more fine-grained way.

- In *model checking*, we are interested in reasoning about properties of a model of computation representing a real system. We will see that in the model of computation we consider, it is possible to naturally encode various classes of circuits, state machines and tile systems but also to design formulas speaking about them. From these formulas, it may be possible to extract a fine-grained logical system for tractable reasoning. In *program specification* there is a similar approach with (usually imperative) programs instead.

- In *unit testing*, a program is tested against a finite of tests in order to guarantee a specific computational behaviour and to certify a partial correctness. By generalising correctness criteria for proof-nets, we are able to speak about logical correctness as a specific case of unit testing. These tests can also be typed and unit testing applies to any model of computation we can express.

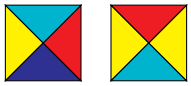
2. Stellar Resolution

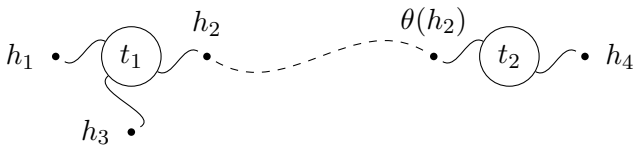
The stellar resolution is the computational ground selected by Girard. In fact, this is not a necessary choice. Other choices can be made but this one is especially convenient for linear logic. It comes from a generalisation of flows coming from the third article of GoI [18].

We use the name "stellar" for Girard's terminology of *stars and constellations* and "resolution" for its similarities with other resolution-based models [31, 43].

For pedagogical purposes, we describe our model of computation as a generalisation of tiling models and show that it behaves like a logic programming language.

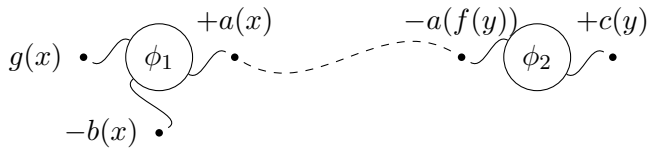
2.1. Tile systems

Wang tiles [46]  We consider bricks with four faces for the four directions in the plane $\mathbf{Z}^2$. We stick the faces together if they have the same colour. We are usually interested in making tilings by reusing the tiles as many times as possible. This is well-known model and a good introduction to tiling-based computation.



Flexible tiles [28, 27] Instead of imposing planarity to tilings, we can allow tiles to have flexible sides selected from set of *sticky-ends types* H. We connect the sticky-ends w.r.t. an involution θ defining complementarity. Surprisingly, this model is able to simulate rigid tiles such as Wang tiles [28].

2.2. Stars and constellations



The stellar resolution can be understood as a flexible tiling model with terms called *rays* as sticky-ends. A ray can either be polarised with a head symbol called *colour* (e.g $+a(x)$ or $-a(f(y))$ where a is a colour where t is any term) or unpolarised (e.g x or $g(x)$).

A *star* (tile) is a finite multiset of rays $\phi = [r_1, ..., r_n]$ and a *constellation* $\Phi = \phi_1 + ... + \phi_m$ (tile set / program) is a (potentially infinite) multiset of stars. We consider stars to be equivalent up to renaming and no two stars within a constellation share variables: these variables are local in the sense of usual programming. The empty star is written $[]$.

By using occurrences of stars from a given constellation, we connect them along their matchable rays of opposite polarity in order to construct tilings called *diagrams*.

We get the following correspondence of terminology:

Tiles	**Stellar Resolution**
Sticky-end	Ray $r = +a(t), -a(t), t$
Tile	Star $\phi = [r_1, ..., r_n]$
Tile set	Constellation $\Phi = \phi_1 + ... + \phi_m$
Tiling	Diagram

Example 1. *We encode two typical logic programs as constellations. We write s^n for n applications of the symbol s in order to encode natural numbers. A colour corresponds to a predicate and the polarity represents the distinction input/output or hypothesis/conclusion. An unpolarised ray cannot interact with other rays.*

```
add(0, y, y).
add(s(x), y, s(z)) :- add(x, y, z).
?add(s^n(0), s^m(0), r).
```

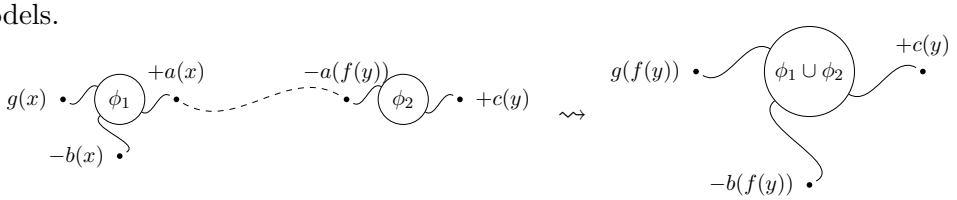
$$\Phi_{\mathbf{N}}^{n,m} = [+add(0, y, y)] + [-add(x, y, z), +add(s(x), y, s(z))] + [-add(s^n(0), s^m(0), r), r]$$

```
parent(d, j). parent(z, d). parent(s, z). parent(n, s).
ancestor(x, y) :- parent(x, y).
ancestor(x, z) :- parent(x, y), ancestor(y, z).
?ancestor(j, r).
```

$$\Phi_{family} = [+parent(d, j)] + [+parent(z, d)] + [+parent(s, z)] + [+parent(n, s)] +$$
$$[+ancestor(x, y), -parent(x, y)] + [+ancestor(x, z), -parent(x, y), -ancestor(y, z)] +$$
$$[-ancestor(j, r), r]$$

2.3. Evaluation of constellations

We are now interested in tiling evaluation which is not standard in the theory of tiling models.



We first define an evaluation of diagrams called *actualisation*. If stars are understood as kind of molecules, then evaluating diagrams corresponds to triggering the actual interaction between the stars along their connected rays, thus making a kind of chemical

reaction happen and propagate to the non-involved entities. Another way to see it is to solve constraints between the connected rays and propagating the solution to the free (unconnected) rays (which survive to the evaluation).

There are two equivalent ways of contracting diagrams into a star by observing that each edge defines a unification problem (equation between terms):

Fusion We can reduce the links step-by-step by solving the underlying equation, producing a solution θ. The two linked stars will merge by making the connected rays disappear. The substitution θ is finally applied on the rays of the resulting star. This is Robinson's resolution rule.

Actualisation The set of all edges defines a big unification problem. The solution θ of this problem is then applied on the star of free rays.

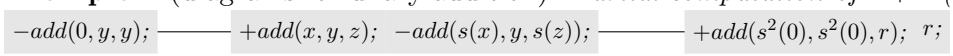
In order to get a successful evaluation we need our diagrams to satisfy few properties (only the first and last ones are mandatory). They usually need to be:

Connected because otherwise, we would always have infinitely many diagrams by repeating isolated stars (notice that it ensures a reduction into a single star);

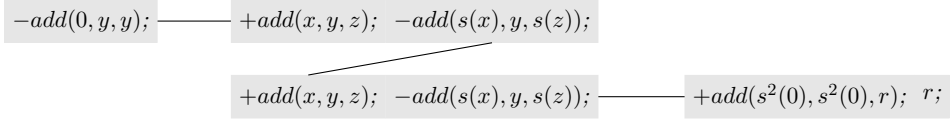
Saturated meaning that it is impossible to extend the diagram with more occurrences of stars. It represents a kind of maximal/complete computation. In practice, we often consider the more general exclusion of diagrams with free coloured rays since they represent incomplete computations in some sense (see subsection 3.1).

Correct meaning that the actualisation does not fail (no contradiction during conflict resolution).

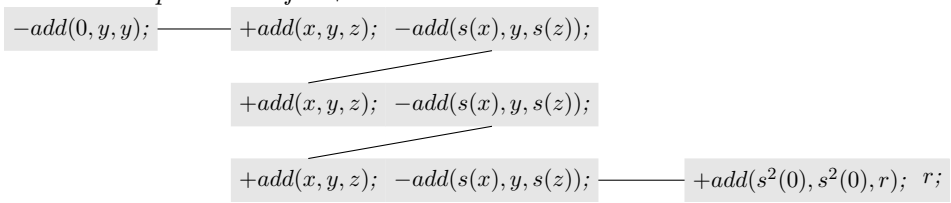
Example 2 (diagrams for unary addition)**.** *Partial computation of* $2+2$ *(0 recursion):*



Complete computation of $2+2$ *(1 recursion):*

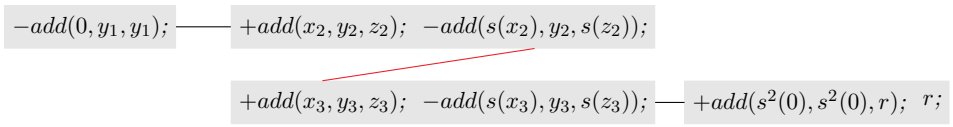


Over computation of $2+2$*:*

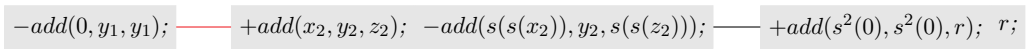


Note that in the case of $\Phi_{\mathbf{N}}^{n,m}$*, there is infinitely many saturated diagrams but only one is correct: the one corresponding to a successful computation of* $n+m$*. Also notice that the maximal size of diagrams naturally corresponds to the time complexity of constellations.*

Example 3 (Fusion)**.** *The full fusion of the diagram representing a complete computation of* $2+2$ *from example 2 is described below (we make the exclusion of variable explicit for clarity):*

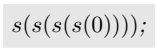


$$\downarrow \theta = \{x_3 \mapsto s(x_2), y_3 \mapsto y_2, z_3 \mapsto s(z_2)\}$$



$$\downarrow \theta = \{x_2 \mapsto 0, z_2 \mapsto y_2\}$$

$$\downarrow \theta = \{y_2 \mapsto s(s(0)), r \mapsto s(s(s(s(0))))\}$$



Example 4 (actualisation)**.** *If we take the diagram* δ *representing a complete computation in the example 2, it generates the following problem:*

$$\mathcal{P}(\delta) = \{add(0, y_1, y_1) \stackrel{?}{=} add(x_2, y_2, z_2), add(s(x_2), y_2, s(z_2)) \stackrel{?}{=} add(x_3, y_3, z_3),$$

$$add(s(x_3), y_3, s(z_3)) \stackrel{?}{=} add(s^2(0), s^2(0), r)\}$$

which is solved by a unification algorithm such as the Montanari-Martelli algorithm [34] in order to obtain a finale substitution:

$$\rightarrow^* \{x_2 \stackrel{?}{=} 0, y_2 \stackrel{?}{=} y_1, z_2 \stackrel{?}{=} y_1, x_3 \stackrel{?}{=} s(x_2), y_2 \stackrel{?}{=} y_3, z_3 \stackrel{?}{=} s(z_2),$$

$$s(x_3) \stackrel{?}{=} s^2(0), y_2 \stackrel{?}{=} s^2(0), s(z_3) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{y_2 \stackrel{?}{=} y_1, z_2 \stackrel{?}{=} y_1, x_3 \stackrel{?}{=} s(0), y_2 \stackrel{?}{=} y_3, z_3 \stackrel{?}{=} s(z_2), s(x_3) \stackrel{?}{=} s^2(0), y_2 \stackrel{?}{=} s^2(0), s(z_3) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{z_2 \stackrel{?}{=} y_1, x_3 \stackrel{?}{=} s(0), y_1 \stackrel{?}{=} y_3, z_3 \stackrel{?}{=} s(z_2), s(x_3) \stackrel{?}{=} s^2(0), y_2 \stackrel{?}{=} s^2(0), s(z_3) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{x_3 \stackrel{?}{=} s(0), y_1 \stackrel{?}{=} y_3, z_3 \stackrel{?}{=} s(y_1), s(x_3) \stackrel{?}{=} s^2(0), y_1 \stackrel{?}{=} s^2(0), s(z_3) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{y_1 \stackrel{?}{=} y_3, z_3 \stackrel{?}{=} s(y_1), s(s(0)) \stackrel{?}{=} s^2(0), y_1 \stackrel{?}{=} s^2(0), s(z_3) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{z_3 \stackrel{?}{=} s(y_3), s(s(0)) \stackrel{?}{=} s^2(0), y_3 \stackrel{?}{=} s^2(0), s(z_3) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{s(s(0)) \stackrel{?}{=} s^2(0), y_3 \stackrel{?}{=} s^2(0), s(s(y_3)) \stackrel{?}{=} r\}$$

$$\rightarrow^* \{y_3 \stackrel{?}{=} s^2(0), s(s(y_3)) \stackrel{?}{=} r\}$$
$$\rightarrow^* \{s(s(s^2(0))) \stackrel{?}{=} r\}$$
$$\rightarrow^* \{r \stackrel{?}{=} s(s(s^2(0)))\}$$

The solution of this problem is the substitution $\theta = \{r \mapsto s^4(0)\}$ *which is applied on the star of free rays* $[r]$*. The result* $[s^4(0)]$ *of this procedure is called the* actualisation *of* δ*.*

$$\Phi \quad \stackrel{\text{generates}}{\longrightarrow} \quad \cup_{k=0}^{\infty} D_k \quad \stackrel{\text{actualises}}{\longrightarrow} \phi_1 + ... + \phi_n$$

The *execution* or *normalisation* $\texttt{Ex}(\Phi)$ of a constellation Φ constructs the set of all possible correct saturated diagrams and actualises them all in order to produce a new constellation called the *normal form*.

In logic programming, we can interpret the normal form as a subset of the application of resolution operator [33]. It non-deterministically computes all the "maximal/complete" inferences possible.

If the set of correct saturated diagrams is finite (or the normal form is a finite constellation), the constellation is said to be *strongly normalising*.

Example 5. *For* $\Phi^{2,2}_{\mathbf{N}}$ *(example 1), one can check that we have* $\texttt{Ex}(\Phi^{2,2}_{\mathbf{N}}) = [s^4(0)]$ *because only the complete computation of example 2 succeed and all other saturated diagrams represents partial or over computations and fail.*

3. Encoding of some models of computation

We provide concrete examples of how our model computes. There are two interesting points about computing with stellar resolution.

- The stellar resolution computes by encoding a structure (hypergraph) and a transmission of information inside it. This generalises circuits and automata. It also make the correspondence between tiling models and automata direct (some links were already investigated [45]).
- Models of computation dependent of external definitions (an evaluation function as for circuits or an environment) are translated into two dual constellations, one corresponding to the model and the other corresponding to its environment. We require that they interact as expected. We will see it in the encoding of circuits.

3.1. Non-deterministic finite automata

The idea is to represent the transitions by binary bipolar stars.

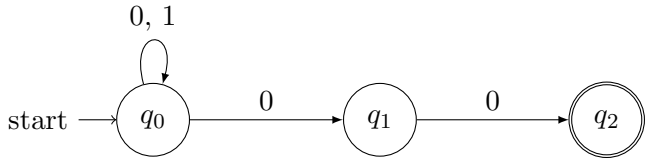
Let Σ be an alphabet and $w \in \Sigma^*$ a word. If $w = c_1...c_n$ then $w^\bigstar = [+i(c_1 \cdot (... \cdot (c_n \cdot \varepsilon)))]$. We use the binary function symbol $\cdot$ which is right-associative.

Let $A = (\Sigma, Q, Q_0, \delta, F)$ be a non-deterministic finite automata. We define its translation $A^\bigstar$:

- for each $q_0 \in Q_0$, we have $[-i(w), +a(w, q_0)]$.
- for each $q_f \in F$, we have $[-a(\varepsilon, q_f), \texttt{accept}]$.
- for each $q \in Q, c \in \Sigma$ and for each $q' \in \delta(q, c)$, we have the star

$$[-a(c \cdot w, q), +a(w, q')].$$

For instance, the following automaton A accepting binary words ending by 00:



is translated as:

$$A^\bigstar = [-i(w), +a(w, q_0)] + [-a(\epsilon, q_2), \texttt{accept}] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)]$$

The set of saturated correct diagrams corresponds to the set of non-deterministic runs. With the word $[+i(0 \cdot 0 \cdot 0 \cdot \varepsilon)]$ only the run $q_0q_0q_1q_2$ leads to the finale state. The corresponding diagram will actualise into $[\texttt{accept}]$. The other correct diagrams correspond to incomplete computations and are excluded. We get $\texttt{Ex}(A^\bigstar + [+i(0 \cdot 0 \cdot 0 \cdot \varepsilon)]) = [\texttt{accept}]$, meaning that the word is accepted.

This idea can easily be extended to pushdown automata. We briefly illustrate the idea: the star $[-a(1 \cdot w, 0 \cdot s), +a(w, s)]$ corresponds to checking if we read 1 and that 0 is on the top of the stack and if so, we remove it. Since we manipulate terms and that Turing machines can be seen as pushdown automata extended with two stacks, the extension to tree automata and Turing machines is direct.

Also remark that there is no explicit flow of computation and that paths are constructed non-deterministically in an asynchronous way.

3.2. Boolean circuits

The idea is to first encode an hypergraph representing the structure of a boolean circuit then to connect the resulting constellation to another one containing the implementation of logic gates (the semantics) as if it was a sort of `#include <proplogic.h>` in the C language.

$$VAR(Y, i) := [-val(x), Y(x), +c_i(x)]$$
$$SHARE(i, j, k) := [-c_i(x), +c_j(x), +c_k(x)]$$
$$AND(i, j, k) := [-c_i(x), -c_j(y), -and(x, y, r), +c_k(r)]$$

$$OR(i,j,k) := [-c_i(x), -c_j(y), -or(x,y,r), +c_k(r)]$$
$$NEG(i,j) := [-c_i(x), -neg(x,r), +c_j(r)]$$
$$CONST(k,i) := [+c_i(k)] \qquad QUERY(k,i) := [+c_i(k), R(k)]$$

where i, j, k are encodings of natural numbers representing identifiers. We also have a star $VAR(Y,i)$ for each variable Y we want as input in our boolean circuit.

We consider the following constellation representing a kind of "module" (as in any programming language) providing the definitions of propositional logic:

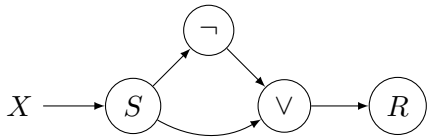
$$\Phi^{\mathcal{PL}} = [+val(0)] + [+val(1)] + [+neg(0,1)] + [+neg(1,0)] +$$
$$[+and(0,0,0)] + [+and(0,1,0)] + [+and(1,0,0)] + [+and(1,1,1)]$$
$$[+or(0,0,0)] + [+or(0,1,1)] + [+or(1,0,1)] + [+or(1,1,1)].$$

We can observe that by changing the module $\Phi^{\mathcal{PL}}$, we can adapt the encoding and obtain arithmetic circuits (possibly with several alternative implementations). Notice that syntax and semantics live in the same world (they are represented as objects of the same kind) and can interact.

We use the star $CONST(k)$ to force a value on the input and the star $QUERY(k)$ to ask for a particular output. For instance, $QUERY(1)$ asks for satisfiability.

To illustrate the encoding, we execute the constellation representing $X \vee \neg X$ where $\underline{n}$ is an encoding of natural number:

$$\Phi_{em} = VAR(X, \underline{0}) + SHARE(\underline{0}, \underline{1}, \underline{2}) + NEG(\underline{2}, \underline{3}) + OR(\underline{1}, \underline{3}, \underline{4}) + QUERY(1, \underline{4})$$



The constellation $\Phi_{em} \uplus \Phi^{\mathcal{PL}}$ will generate two diagrams: one corresponding to the input 0 and another one for 1. We obtain $\texttt{Ex}(\Phi_{em} + \Phi^{\mathcal{PL}}) = [X(0), R(1)] + [X(1), R(1)]$ stating that for the two valuations $x \mapsto 0$ and $x \mapsto 1$, the circuit outputs $R(1)$.

3.3. Comments on the model

This model of computation embodies the idea of computation-as-interaction [1]. It is a very primitive model of computation which is based on our intuition of space (a geometric/topological structure where information flows). The rays can be seen as wires we can divide (for instance $f(x)$ can be divided into subwires $f(\texttt{l} \cdot x)$ and $f(\texttt{r} \cdot x)$) and thus enjoy a topological/geometrical interpretation.

Relationship with logic programming At first, our model seems identical to Robinson's first order resolution using disjunctive clauses. The difference is that our model is purely computational (no reference to logic) and that we use it for a different

purpose (no interest in reaching the empty clause but rather the set of atoms we can infer). Although this dynamics is well-known it seems that it has never been used that way as a logic-free model of computation by its own. Moreover, our model will be extended with internal colours and possibly additional features in the future, thus justifying the use of a new name.

Relationship with tiling models The stellar resolution generalises *flexible tiles* [28, 27], a model of computation coming from DNA computing [47]. This model is itself able to simulate tile systems such as Wang tiles [46] or abstract tile assembly models [35] by encoding both the tile set and its environment by constellations. From our realisability construction, one can imagine methods of typing and implicit complexity analysis of these models.

Relationship with the GoI and complexity We can generalise flows [18, 7] and interaction graphs [39, 40] which have applications in implicit computational complexity [8, 5, 6, 42]. Some of these works encode some notions of automata which we can also reproduce and extend.

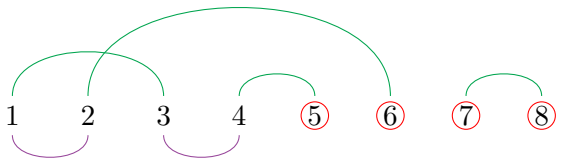
4. Geometry of Interaction for MLL

We explain how to encode MLL proof-structures and to simulate both MLL cut-elimination and logical correctness (by the Danos-Regnier criterion) as a first step towards a full reconstruction of linear logic.

4.1. Cut-elimination and permutations

When considering cut-elimination, axioms induce a permutation on the atoms which conclusion of axiom rules. As in ludics [19], we can call these atoms *loci* (plural of *locus*) and they represent the physical locations inside a proof. We can actually forget the formulas which are simply labels and only see abstract physical locations.

The $\parr/\otimes$ cuts can be seen as administrative/inessential cuts since all they do is basically a rewiring on the premises of the $\parr$ and $\otimes$ nodes connected together. Therefore, such a cut can be seen as a permutation/edge between two loci. We observe that the loci are the "support of the proof"; the locations where logical interaction takes place.



Seiller [39] studied the GoI through juxtaposition of graphs where cut-elimination becomes the computation of maximal alternating paths. In this setting, proof-nets simply speaks about locations and paths between them and truth/correctness is about cycles and connectivity.

The stellar resolution generalises this idea by encoding hypergraphs representing proof-structures. In the case of cut-elimination, we only need binary stars representing graph edges. The above graph becomes:

$$\Phi = [+c(1), -c(3)] + [+c(4), 5] + [+c(2), 6] + [7, 8]$$
$$[-c(1), -c(2)] + [-c(3), -c(4)]$$

Notice that we only use the colours $+c$ for the locations which interact with the cuts. We have $\mathtt{Ex}(\Phi) = [5, 6] + [7, 8]$ which corresponds to the expected normal form (set of maximal paths). Notice that the matching is exact, hence the actualisation is a trivial contraction of a unique diagram (since no non-deterministic choice is involved).

4.2. Correctness and partitions

If we wish to treat logical correctness in a satisfactory and natural way for MLL, we have to shift to partitions instead of permutations [14, 3]. Permutations can still be retrieved: a permutation $\{x_1 \mapsto y_1, ..., x_n \mapsto y_n\}$ on $X \subseteq \mathbf{N}$ representing a proof naturally induces a partition $\{\{x_1, y_1\}, ..., \{x_n, y_n\}\}$ in X.

A Danos-Regnier switching induces a partition depending on how it separates or groups atoms into different connected components:

The above switching corresponds to the partition $\{\{1, 2\}, \{3\}, \{4\}\}$. Partitions are related by orthogonality: two partitions are orthogonal if the graph constructed with sets as nodes and where two nodes are adjacent whenever they share a common value is a tree. Testing a partition for axioms against several partitions for the switchings is sufficient to speak about logical correctness. For more details, see Acclavio and Maieli's works [3].

Since stars are not limited to the binary case, we can naturally represent general partitions by constellations: $[-c(1), -c(2)] + [-c(3)] + [-c(4)]$ (notice the negative polarity in order to allow connexion with axioms). However, in this case, all diagrams are closed (no free rays). For technical reaosns, because otherwise the switchings would not be distinguished, we have to specify where the conclusions are located: $[-c(1), -c(2), 1 \otimes 2] + [-c(3)] + [-c(4), 3 \parr 4]$.

We now consider a constellation representing axioms all coloured with $+c$ in order to allow testing with switchings:

$$\Phi = [+c(1), +c(3)] + [+c(2), +c(4)]$$
$$\Phi^L_{\parr} = [-c(1), -c(2), 1 \otimes 2] + [-c(3)] + [-c(4), 3 \parr 4].$$

We have $\mathtt{Ex}(\Phi \uplus \Phi^L_{\parr}) = [1 \otimes 2, 3 \parr 4]$ which is the star of conclusions. In this case, we say that Φ passes the test $\Phi^L_{\parr}$.

The Danos-Regnier criterion is reformulated as follows: "a constellation representing a proof-structure is correct if and only if its execution against all the constellations representing its switchings produces the star of its conclusions.

5. Stellar interpretation of multiplicative linear logic

5.1. The computational content of multiplicatives

We now consider a more general setting by using terms containing variables as rays instead of constants. This is not especially useful for MLL but will be crucial for the exponentials or further extensions (for instance first and second order linear logic).

We set a basis of representation $\mathbb{B}$ with unary symbols p_A for all formulas of MLL, and constants $\mathtt{l}, \mathtt{r}$ to represent the address of a locus relatively to the tree structure of the lower part of a proof-structure. We use a right-associative binary symbol $\cdot$ to glue constants together. Any other isomorphic basis can be considered as well.

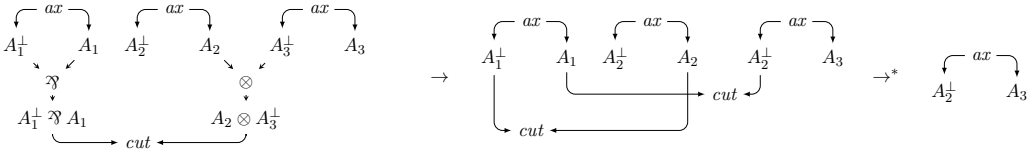
To simulate the dynamics of cut-elimination we translate the axioms and the cuts into stars:

1. A locus A becomes a ray $+c.p_A(t)$ if it is involved with a cut or $p_A(t)$ otherwise, where t represents the "address" of A relatively to the conclusions of the proof-structure (without considering cuts). We use and encoding of the path from a conclusion to A in the tree corresponding to the lower part of the proof-structure. The colour $+c$ stands for "positive computation".

2. An axiom becomes a binary star containing the translations of its loci as described above. For instance, the axiom $A, A^\perp$ becomes $[p_A(x), p_{A^\perp}(x)]$.

3. A cut between A and $A^\perp$ becomes a binary star $[-c.p_A(x), -c.p_B(x)]$ coloured with $-c$ for "negative computation". This colour of opposite polarity allows connexion with the axioms.

Example 6. *We encode the following cut-elimination $\mathcal{S} \to^* \mathcal{S}'$ of MLL proof-structures:*

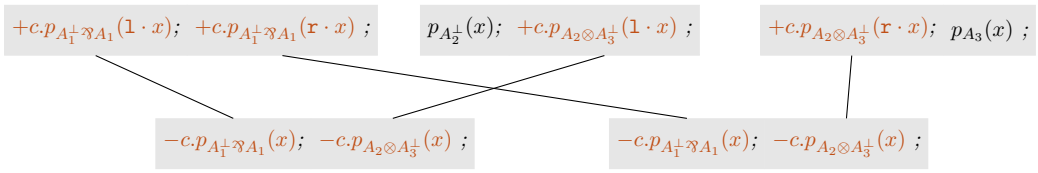


The address of $A_1^\perp$ is $p_{A_1^\perp ⅋ A_1}(\mathtt{l}\cdot x)$ because it is located on the left-hand side of $A_1^\perp ⅋ A_1$. The address of $A_3^\perp$ is $p_{A_2 \otimes A_3^\perp}(\mathtt{r}\cdot x)$ and the one for A_3 is $p_{A_3}(x)$. The proof-structure $\mathcal{S}$ is encoded as:

$$[+c.p_{A_1^\perp ⅋ A_1}(\mathtt{l}\cdot x), +c.p_{A_1^\perp ⅋ A_1}(\mathtt{r}\cdot x)] + [p_{A_2^\perp}(x), +c.p_{A_2 \otimes A_3^\perp}(\mathtt{l}\cdot x)] +$$

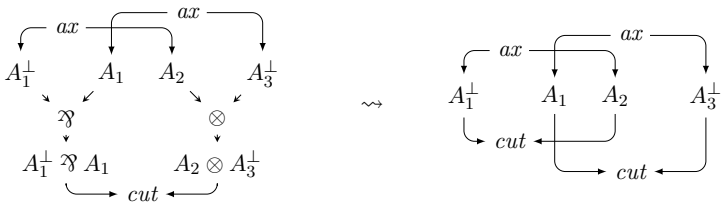
$$[+c.p_{A_2 \otimes A_3^\perp}(\mathtt{r}\cdot x), p_{A_3^\perp}(x)] + [-c.p_{A_2 \otimes A_3^\perp}(x), -c.p_{A_2 \otimes A_3^\perp}(x)]$$

The only correct saturated diagram is:



The matching is exact (since no non-deterministic choice involved) so the connected rays are removed and we end up with $[p_{A_2^\perp}(x), p_{A_3}(x)]$ *corresponding to* $\mathcal{S}'$*, as expected.*

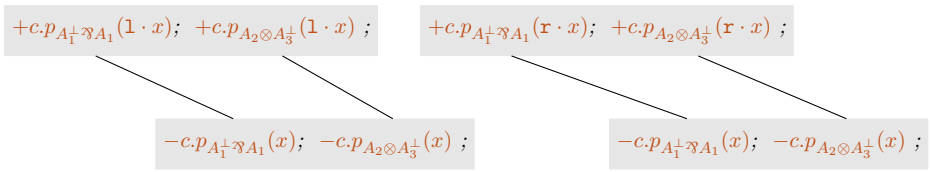
Example 7. *If we have the following reduction* $\mathcal{S} \to \mathcal{S}'$ *instead:*



The constellation corresponding to $\mathcal{S}$ *is*

$$[+c.p_{A_1^\perp \parr A_1}(\mathtt{l}\cdot x), +c.p_{A_2\otimes A_3^\perp}(\mathtt{l}\cdot x)] + [+c.p_{A_2\otimes A_2^\perp}(\mathtt{r}\cdot x), +c.p_{A_1^\perp \parr A_1}(\mathtt{r}\cdot x)] +$$
$$[-c.p_{A_1^\perp \parr A_1}(x), -c.p_{A_2\otimes A_3^\perp}(x)]$$

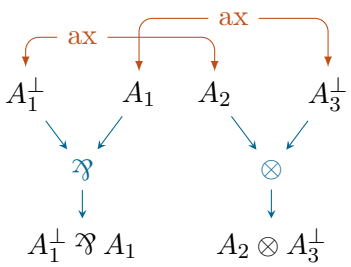
When trying to make a saturated diagram by following the shape of the proof-structure, we end up with:



which normalises into infinitely many occurrences of the empty star []*. Hence, the proof-structure is incorrect.*

5.2. The logical content of multiplicatives

We translate the correctness criterion of Danos-Regnier [11] by an encoding of hypergraphs (representing proof-structures) into constellations.



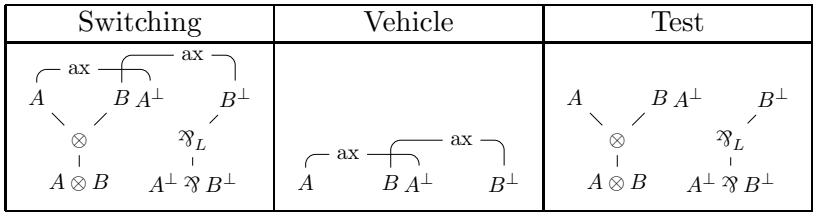


Figure 1: The vehicle of a proof-structure and a test corresponding to a Danos-Regnier switching graph.

Any proof-structure can be seen as the sum of two components:

- The upper part made of axioms appearing in a proof-structure is called the *vehicle*. It holds the computational content of the proof (since cuts ultimately only interact with axioms).
- The lower part is basically the syntax forest of the conclusion $\vdash A_1, ..., A_n$ of the proof-structure. It holds the logical content of the proof (formula/correctness). The Danos-Regnier correctness criterion is obtained by considering switchings on the lower part and checking the properties of acyclicity and connectedness for each. This can be understood as *testing* the vehicle against a *set of test* we call the *format*. This testing between vehicle and format produces a certification: if all tests pass, we have a *proof-net*. Note that testing is symmetric because tests are encoded as constellations as well: a format can also be seen as tested by a vehicle.

We call the translation of switching graphs *tests*. They are defined in a very natural way by translating the nodes of the lower part of a proof-structure $\mathcal{S}$ into constellations:

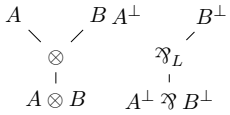
- $A^\star = [-t.\mathtt{addr}_{\mathcal{S}}(A), +c.p_A(\mathsf{g} \cdot x)]$ where A is a conclusion of axiom;
- $(A \parr_L B)^\star = [-c.p_A(\mathsf{g} \cdot x)] + [-c.p_B(\mathsf{g} \cdot x), +c.p_{A \parr B}(\mathsf{g} \cdot x)]$;
- $(A \parr_R B)^\star = [-c.p_A(\mathsf{g} \cdot x), +c.p_{A \parr B}(\mathsf{g} \cdot x)] + [-c.p_B(\mathsf{g} \cdot x)]$;
- $(A \otimes B)^\star = [-c.p_A(\mathsf{g} \cdot x), -c.p_B(\mathsf{g} \cdot x), +c.p_{A \otimes B}(\mathsf{g} \cdot x)]$;
- We add $[-c.p_A(\mathsf{g} \cdot x), p_A(x)]$ for each conclusion A.

where $-t.\mathtt{addr}_{\mathcal{S}}(A)$ corresponds to the address of A relatively to $\mathcal{S}$. The colour t stands for "typing".

Definition 1 (Logical correctness)**.** *The translation of a proof-structure of conclusion* $\vdash A_1, ..., A_n$ *is said to be correct when for each translation of switching graph, their union normalises into* $[p_{A_1}(x), ..., p_{A_n}(x)]$.

The essential point of the translation is that the corresponding dependency graph, obtained by describing how the rays can be connected to each other, will have the same structure as a switching graph.

Example 8. *Here is an example with the switching graph and the test of figure 1:*

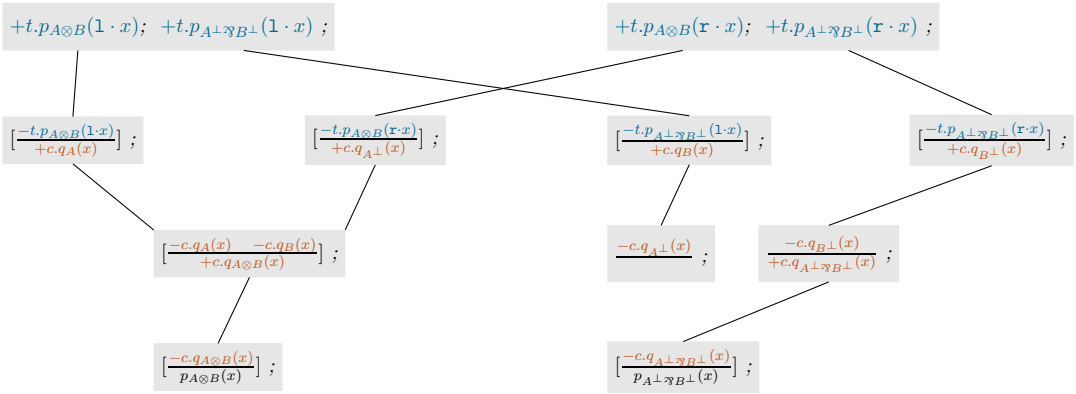


$$\begin{array}{l}[\frac{-t.p_{A\otimes B}(\mathtt{l}\cdot x)}{+c.q_A(x)}]+[\frac{-t.p_{A^\perp \parr B^\perp}(\mathtt{l}\cdot x)}{+c.q_B(x)}]+\\ [\frac{-t.p_{A\otimes B}(\mathtt{r}\cdot x)}{+c.q_{A^\perp}(x)}]+[\frac{-t.p_{A^\perp \parr B^\perp}(\mathtt{r}\cdot x)}{+c.q_{B^\perp}(x)}]+\\ [\frac{-c.q_A(x)\quad -c.q_B(x)}{+c.q_{A\otimes B}(x)}]+[\frac{-c.q_{A^\perp}(x)}{}]+[\frac{-c.q_{B^\perp}(x)}{+c.q_{A^\perp \parr B^\perp}(x)}]+\\ [\frac{-c.q_{A\otimes B}(x)}{p_{A\otimes B}(x)}]+[\frac{-c.q_{A^\perp \parr B^\perp}(x)}{p_{A^\perp \parr B^\perp}(x)}]\end{array}$$

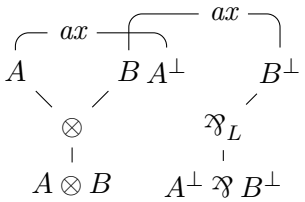
When connected to the vehicle:

$$[+t.p_{A\otimes B}(\mathtt{l}\cdot x), +t.p_{A^\perp \parr B^\perp}(\mathtt{l}\cdot x)]+[+t.p_{A\otimes B}(\mathtt{r}\cdot x), +t.p_{A^\perp \parr B^\perp}(\mathtt{r}\cdot x)]$$

we obtain the following dependency graph:

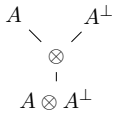


structurally corresponding to the following switching graph:



Since the matching of the constellation is exact and that the dependency graph is a tree, only the free rays will be kept in the normal form. We obtain $[p_{A\otimes B}(x), p_{A^\perp \parr B^\perp}(x)]$.

Example 9. *If we have the following test instead:*



$$\begin{array}{l}[\frac{-t.p_{A\otimes A^\perp}(\mathtt{l}\cdot x)}{+c.q_A(x)}]+[\frac{-t.p_{A\otimes A^\perp}(\mathtt{r}\cdot x)}{+c.q_{A^\perp}(x)}]+\\ [\frac{-c.q_A(x)\quad -c.q_{A^\perp}(x)}{+c.q_{A\otimes A^\perp}(x)}]+[\frac{-c.q_{A\otimes A^\perp}(x)}{p_{A\otimes A^\perp}(x)}]\end{array}$$

When connected to the vehicle $[+c.p_{A\otimes A^\perp}(\mathtt{l}\cdot x), +c.p_{A\otimes A^\perp}(\mathtt{r}\cdot x)]$*, a loop appears in the dependency graph and since the matching is exact, we can construct infinitely many correct diagrams. The constellation is not strongly normalising.*

5.3. What is a proof?

We started from very general untyped objects which are the constellations and reconstructed the elementary bricks of multiplicative linear logic. Our framework is general enough to write a lot of things which were impossible to write with proof-structures:

- an atomic pre-proof of conclusion $\vdash A$ as the constellation $\{[p_A(x)]\}$,
- an n-ary axiom as the constellation $\{[p_{A_1}(x), ..., p_{A_n}(x)]\}$.

We can finally define what the full translation of a proof-structure is. A proof-structure is translated into a triple (Φ_V, Φ_C, Φ_F) where:

- Φ_V is the uncoloured vehicle made of translations of axioms,
- Φ_C is the uncoloured translation of cuts,
- Φ_F is coloured translation of all ordeals (the format).

We see that proof-structures can be considered as being made of three components. We will then colour the components depending on if we want to perform a cut-elimination or test the correctness. This shows that proof-structures, although being considered untyped, actually come with an implicit pre-made typing. We made the implicit logical assumptions computationally explicit as stated in the introduction.

6. Two notions of type/formula

Depending of whether types or programs come first, we have two distinct notions of typing which are reunited in the Transcendental Syntax. Girard talks about *existentialism* when programs come first and *essentialism* when programs do.

We first have to define an orthogonality relation $\perp$ opposing constellations depending of what we consider a correct interaction. This is a subjective side a logic. Several choices are possible and lead to different results. For instance, we may consider:

- $\Phi \perp \Phi'$ when $|\mathtt{Ex}(\Phi \uplus \Phi')| < \infty$ (strong normalisation) which captures MLL+MIX correctness.
- $\Phi \perp \Phi'$ when $|\mathtt{Ex}(\Phi \uplus \Phi')| = 1$ which captures MLL correctness.

6.1. A priori typing / à la Church / Girard's factory

In usual proof theory and traditional type theory (Martin-Löf type theory for instance), types come first and can be seen as simple labels. We usually want type checking to be tractable by using rules and few good properties ensuring a sound computational behaviour. However, some programs may not be typable (for instance $\lambda x.xx$ in simply typed λ-calculus). Types represent arbitrary constraints on computation.

In the theory of proof-nets, we have proof-structures as general computational objects and we can check if they are logically correct. What we mean by "correct" is that it corresponds to a proof of sequent calculus. Here, the Danos-Regnier switchings correspond to a constraint certifying proof-nets. They are rules telling if a proof-structure is truly a proof of its conclusions.

The Transcendental Syntax takes inspiration from that and consider a notion of typing by testing. We can define an arbitrary set of tests (encoded as constellations) $\Phi_1, ..., \Phi_n$ and say that a constellation which is orthogonal to all these tests is of type A. Each formula is then associated to a set of tests.

For instance, we can consider the type Nat for natural numbers and say that $\Phi \perp \Phi'$ whenever $|\mathtt{Ex}(\Phi \uplus \Phi')| = 1$. We compare the traditional approach and the approach of the Transcendental Syntax where we only have a single test.

	Type Theory	**Transcendental Syntax**
Type checkers	$\dfrac{}{0 : \mathrm{Nat}} \quad \dfrac{n : \mathrm{Nat}}{s(n) : \mathrm{Nat}}$	$\Phi_{\mathbf{N}} := [+nat(0), ok] + [-nat(n), +nat(s(n))]$
Correct	$\dfrac{\dfrac{}{0 : \mathrm{Nat}}}{s(0) : \mathrm{Nat}}$	$\mathtt{Ex}(\Phi_{\mathbf{N}} + [-nat(s(0))]) = [ok]$
Incorrect	$0(0)$ not well-formed	$\lvert\mathtt{Ex}(\Phi_{\mathbf{N}} + [-nat(0(0))])\rvert \neq 1$

6.2. A posteriori typing / à la Curry / Girard's use

Another point of view is to consider realisability techniques [32, 30] for linear logic similarly to ludics [19]. Instead of types as labels, we consider types as collections of programs corresponding to a particular computational behaviour. Types are seen as descriptions of computational behaviours. It is a more general notion than usual typing. In particular, Church typing is a special case of Curry typing which require finite testing. Types are now *designed* instead of *pre-defined*.

By grouping some constellations together, it is possible to design various sort of types. In this section, we show how to design types corresponding to MLL formulas but atypical connectives can also be constructed.

Pre-conduct A pre-conduct is a set of constellations.

Orthogonal If we have a set of constellations $\mathbf{A}$, its orthogonal, written $\mathbf{A}^{\perp}$, is the set of all constellations which are strongly normalising when interacting with the constellations of $\mathbf{A}$. It corresponds to linear negation.

Conduct A *conduct* (or formula) $\mathbf{A}$ is the orthogonal of another pre-conduct $\mathbf{B}$ i.e $\mathbf{A} = \mathbf{B}^{\perp}$. It means that it interacts well (with respects to the orthogonality) with

another pre-conduct. It is equivalent to say that $\mathbf{A} = \mathbf{A}^{\perp\perp}$ meaning that it is closed by interaction. Such pre-conducts correspond to the ones which can be characterised by tests, in other words, be *testable*.

Atoms We define atoms with a *basis of interpretation* Φ associating for each type variable X_i a distinct conduct $\Phi(X_i)$. It represents a choice of formula for each variable. A more satisfactory way to handle variables is to consider second order quantification, in which case we need further correctness tests. Since our atoms are represented by rays (thus concrete entities), Girard even consider a constant $\mathcal{F}$ (fu) [23] which is self-dual.

Tensor The tensor $\mathbf{A} \otimes \mathbf{B}$ of two conducts is constructed by pairing all the constellations of $\mathbf{A}$ with the ones of $\mathbf{B}$ by using a multiset union of constellations $\Phi_1 \uplus \Phi_2$. The conduct $\mathbf{A}$ and $\mathbf{B}$ have to be disjoint in the sense that they cannot be connected together by two matching rays. Note that the cut is the same thing but the constellations can interact.

Par and linear implication As usual in linear logic, the par and linear implication are defined from the tensor and the orthogonal: $A \parr B = (A^\perp \otimes B^\perp)^\perp$ and $A \multimap B = A^\perp \parr B$. Notice that these connectives are not simply arbitrary definitions on labels but that they have a computational interpretation. The constellations of $A \parr B$ are the one passing the tests of $(A^\perp \otimes B^\perp)$. This testing is potentially infinite.

Alternative definition for linear implication An alternative but equivalent definition of linear implication is the set of all constellations Φ such that if we put them against any constellation of $\mathbf{A}$, the execution produces a constellation of $\mathbf{B}$. This is a standard definition in realisability theory.

It is possible to design any type we want providing it is a conduct. For instance, we could design a type $\mathbf{Cut}(\mathbf{A}, \mathbf{B})$ such that $\mathbf{A} \otimes \mathbf{B} \subseteq \mathbf{Cut}(\mathbf{A}, \mathbf{B})$.

Example 10 (Proof-nets and logical correctness)**.** *This is the more straightforward example. Given an interpretation of atoms, it is possible to associate to any formula label A of MLL, a conduct $\mathbf{A}$. Then if a constellation Φ representing a proof-structure of conclusion A then we have $\Phi \in \mathbf{A}$ and the translation of its Danos-Regnier switching graphs Φ_i can be typed by $\Phi_I \in \mathbf{A}^\perp$. Interestingly, this shows that only a small and finite subset of $\mathbf{A}^\perp$ is sufficient to assert that $\Phi \in \mathbf{A}$ instead of an infinite testing naturally suggested by realisability types. This is the main difference between the Transcendental Syntax and other approaches such as ludics and the GoI.*

Example 11 (Acceptation in finite automata)**.** *From the previous encoding of automata we can observe a duality between automata and words. It induces an orthogonality relation $A^\star \perp w^\star$ whenever $\mathtt{Ex}(A^\star + w^\star) \neq \emptyset$. An automaton becomes orthogonal to all the words it accepts and a word is orthogonal to all the automata which recognise it. Notice that if we have $L = \{w \mid w \text{ ends with } 00\}$, then $L^\perp$ without restriction is not exactly*

the set of automata recognising L by contains more constellations which are also able to recognise L in a different way. If we have an operation of pairwise concatenation $L_1 \bullet L_2 = \{w_1 w_2 \mid w_i \in L_i\}^{\perp\perp}$ (the double orthogonal is here to ensure that we have a conduct), then $(L_1 \bullet L_2)^{\perp}$ is indeed the set of constellations recognising $L_1 \bullet L_2$ but this also defines (by duality), an operator on automata $\mathbf{A_1} \circ \mathbf{A_1} = (\mathbf{A_1}^{\perp} \bullet \mathbf{A_2}^{\perp})^{\perp}$ which construct automata recognising the concatenation of two langauges. This is similar to Terui's works [44] which studied a computational variant of ludics with the difference that our framework is more general and convenient.

Example 12 (Queries and answers in logic programming)**.** *Although quite superficial, we can sketch an idea of typing for logic programming. Let*

$$\Phi^{+}_{\mathbf{N}} = [+add(0, y, y)] + [-add(x, y, z), +add(s(x), y, s(z))]$$

be a constellation computing the sum of two natural numbers. We consider the strong normalisation of the union of two constellations as orthogonality. Let

$$\mathbf{Q}_{+} = \{[-add(s^n(0), s^m(0), r), r] \mid n, m \in \mathbf{N}\}$$

and $\mathbf{A}_{+} = \{[s^n(0)] \mid n \in \mathbf{N}\}$. Take a constellation $[-add(s^n(0), s^m(0), r), r]$ and connect it with $\Phi^{+}_{\mathbf{N}}$. All diagrams corresponding to $\Phi^{+}_{\mathbf{N}}$ with n occurrences of

$$[+add(s(x), y, s(z)), -add(x, y, z)]$$

can be reduced to a star $[s^{n+m}(0)]$. It is easy to check that all other diagram fails. Therefore, for all $\Phi \in \mathbf{Q}_{+}$, $\mathtt{Ex}(\Phi \uplus \Phi^{+}_{\mathbf{N}}) \in \mathbf{A}_{+}$ and $\Phi^{+}_{\mathbf{N}} \uplus \Phi$. If $\mathbf{Q}_{+}$ and $\mathbf{A}_{+}$ are conducts (we need a more specific orthogonality) then $\Phi^{+}_{\mathbf{N}} \in \mathbf{Q}_{+} \multimap \mathbf{A}_{+}$. Although not explored here, it might be possible to retrieve existing type systems and extend them.

We can also imagine more interesting examples: typing for tilings models (thus typing for DNA computing), characterisation of properties of constellations (characterisation of complexity classes?).

As in classical realisability theory, we can consider an adequacy lemma linking the two notions of typing. It states that a finite set of tests for formula A is sufficient in order to ensure membership to the conduct $\mathbf{A}$ corresponding to A. In other words, that the label guarantees the behaviour.

Definition 2 (Adequacy property)**.** *Let Φ be a constellation and $\Phi_1, ..., \Phi_n$ be tests for a formula A. If for all $1 \leq i \leq n$, $\Phi \perp \Phi_i$ then $\Phi \in \mathbf{A}$ where $\mathbf{A}$ is the conduct corresponding to A (it is possible to translate a formula into a conduct by substituting the variables by other conducts).*

6.3. A logical constant from a topological invariant

In Transcendental Syntax's fourth paper [24], Girard gives hints for the definition of a self-dual conduct corresponding to a new logical constant.

The idea is that in the stellar resolution, atoms are translated into concrete objects which are not substituable, unlike in the original theory of proof-nets. It should be possible to group them in a conduct corresponding to the type of atoms. But the only thing they share in common is their topology (a single point). In order to retrieve Girard's logical constant フ, we will consider that two constellations sharing the same structure are seen as being in the same group in the point of view of typing.

This is actually an extension of the correctness criterion on partitions described before but adapted to the case of constellations. We only give informal definitions.

Definition 3 (Orthogonality)**.** *Two constellations* Φ_1 *and* Φ_2 *are* orthogonal*, written* $\Phi_1 \perp \Phi_2$*, if and only if*

- Φ'_1 *and* Φ'_2 *are* Φ_1 *and* Φ_2 *without unpolarised rays and where we force opposite colours on all the rays in order to make them connectable;*
- *the dependency graph of* $\Phi'_1 \uplus \Phi'_2$ *is a tree with all rays connected to exactly one other ray.*

We can then define a conduct corresponding to the type of atoms.

Proposition 1 (Logical constant フ)**.** *The pre-conduct* $フ = \{\{\phi\} \mid |\phi| = 1\}$ *is a self-dual conduct, that is* $フ = フ^{\perp} = フ^{\perp\perp}$*.*

Now, we can give ground to multiplicative propositions. All formula variables can be replaced by フ. For instance $\vdash X_1^{\perp} \parr X_2^{\perp}, X_1 \otimes X_2$ can be translated into the conduct $\vdash フ \parr フ, フ \otimes フ$.

There are few interesting points:

- for any formula A, we can type atomic pre-proofs: $[p_A(x)] \in フ$;
- the axioms are now typable with $[p_A(x), p_{A^{\perp}}(x)] \in フ \parr フ$;
- we can write stand-alone links: $[p_A(x), p_B(x)] \in フ \parr フ$ and $[p_A(x)] + [p_B(x)] \in フ \otimes フ$.

7. Extension with intuitionistic implication

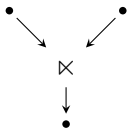
The exponentials have already been studied using flows (corresponding to binary stars) [18, 7] but the correctness was not considered.

The idea is that the stellar resolution already has a built-in mechanism of duplication (by multiple matching and reusing of stars) and weakening (rejection of some unwanted diagrams).

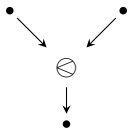
We restrict the definitions to MLL extended with the intuitionistic implication $A \Rightarrow B := !A \multimap B$ and its dual $(A \Rightarrow B)^{\perp}$ in order to follow Girard's papers [16].

- Non-linear formulas are written $\underline{A}$ (this corresponds to $?A$) and only appear at top-level (for conclusions and not subformulas).

- We define new connectives $A \ltimes B$ $(= ?A \parr B)$ and $A \oslash B$ $(= !A \otimes B)$ with the following links:



Exponential par



Exponential tensor

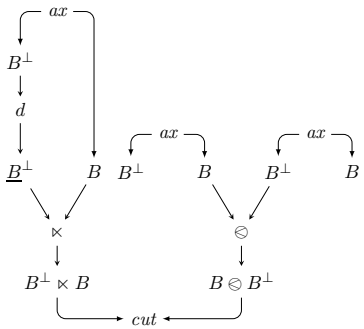
- Weakened formulas are not translated.
- Derelicted formulas $\underline{A}$ are translated as $p_A(t \cdot \mathtt{d})$.
- Contracted formulas $\underline{A}$ are translated into $p_A(t \cdot (\mathtt{l} \cdot y))$ and $p_A(t \cdot (\mathtt{r} \cdot y))$.
- Promoted formulas A are translated into $p_A(t \cdot y)$ for a fresh variable y and its auxiliary formulas B_i into $p_{B_i}(t_i \cdot y)$.

We can remark that subterms of the form $t \cdot u$ are used. The term t corresponds to the multiplicative part encoding the address of atoms as before and the term u encodes the nested boxes (e.g $(t \cdot y_1) \cdot y_2$) and the copy identifier for the contraction. In the case of dereliction, we prevent any duplication on the current layer by a constant $\mathtt{d}$. This gives a box-free approach to exponentials where box dependency is simulated by matchable terms.

7.1. The computational content of exponentials

We illustrate the cut-elimination by few examples by using the usual translation of simply typed λ-terms into proof-nets [10, 36].

Example 13 (Case dereliction/box)**.** *This corresponds to opening a box. We illustrate this case with the identity function applied to an argument:* $(\lambda x.x)y$.

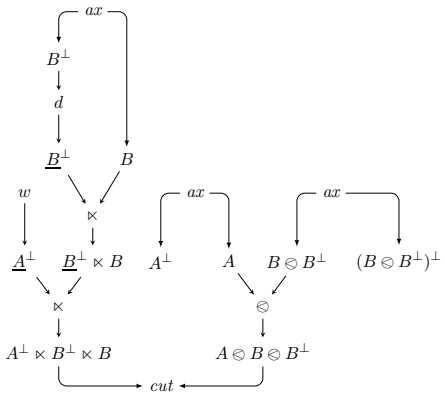


It is translated into:

$$[+c.p_{B^\perp \ltimes B}((\mathtt{l} \cdot x) \cdot \mathtt{d}), +c.p_{B^\perp \ltimes B}(\mathtt{r} \cdot x)]+$$
$$[p_{B^\perp}(x), +c.p_{B \oslash B^\perp}((\mathtt{l} \cdot x) \cdot y)]+$$
$$[+c.p_{B \oslash B^\perp}(\mathtt{r} \cdot x), p_B(x)]+$$
$$[-c.p_{B^\perp \ltimes B}(x), -c.p_{B \oslash B^\perp}(x)]$$

It is similar to the multiplicative case. The essential point is that $+c.p_{B^\perp \ltimes B}((\mathtt{l} \cdot x) \cdot \mathtt{d})$ *will interact with* $+c.p_{B \oslash B^\perp}((\mathtt{l} \cdot x) \cdot y)$ *replacing* y *by* $\mathtt{d}$*. It means that the rays corresponding to this box are no longer duplicable. In some sense, we opened the box.*

Example 14 (Case weakening/box)**.** *It corresponds to a box erasure. We consider the simple example of right projection* $(\lambda xy.y)z$*.*



It is translated into the constellation:

$$[+c.p_{A^\perp \ltimes B^\perp \ltimes B}((\mathtt{r} \cdot \mathtt{l} \cdot x) \cdot \mathtt{d}),$$
$$+c.p_{A^\perp \ltimes B^\perp \ltimes B}(\mathtt{r} \cdot \mathtt{r} \cdot x)]+$$
$$[p_{A^\perp}(x \cdot y), +c.p_{A \otimes B \otimes B^\perp}((\mathtt{l} \cdot x) \cdot y)]+$$
$$[+c.p_{A \otimes B \otimes B^\perp}(\mathtt{r} \cdot x), p_{A \otimes B \otimes B^\perp}(x)]+$$
$$[-c.p_{A^\perp \ltimes B^\perp \ltimes B}(x), -c.p_{A \otimes B \otimes B^\perp}(x)]$$

We can see that the cut will be duplicated into:

$$[-c.p_{A^\perp \ltimes B^\perp \ltimes B}((\mathtt{l} \cdot x) \cdot y), -c.p_{A \otimes B \otimes B^\perp}((\mathtt{l} \cdot x) \cdot y)]$$

and

$$[-c.p_{A^\perp \ltimes B^\perp \ltimes B}(\mathtt{r} \cdot x), -c.p_{A \otimes B \otimes B^\perp}(\mathtt{r} \cdot x)].$$

Since $-c.p_{A^\perp \ltimes B^\perp \ltimes B}((\mathtt{l} \cdot x) \cdot y)$ *has nothing to match (because of the weakening), all diagrams using terms matchable with* $-c.p_{A \otimes B \otimes B^\perp}((\mathtt{l} \cdot x) \cdot y)$ *will be excluded. This indeed corresponds to box erasure. Notice that if we had auxiliary gates, they would end up as independent stars in the normal form, as usual with proof-nets.*

Example 15 (Case contraction/box)**.** *It corresponds to a box duplication. Instead of illustrating it by the translation of a* λ*-term, we will describe how it works:*

$$[+c.p_A(t \cdot (\mathtt{l} \cdot y)), ...] + [+c.p_A(t \cdot (\mathtt{r} \cdot y)), ...] + [+c.p_B(t \cdot y), ...] + [-c.p_A(x), -c.p_B(x)]$$

The cut star forces the matching between rays for A *and for* B*. We see that* $+c.p_B(t \cdot y)$ *can interact with both* $+c.p_A(t \cdot (\mathtt{l} \cdot y))$ *and* $+c.p_A(t \cdot (\mathtt{r} \cdot y))$*, hence the star containing* $+c.p_B(t \cdot y)$ *and all the stars connected to it will be naturally duplicated.*

7.2. The logical content of exponentials

As usual with proof-nets, if we allow the MIX rule then the correctness criterion is straightforward. In this case, we only need to check acyclicity. The only difference with the multiplicative case is that we should check that the variables we use are handled correctly:

- in subterms $x \cdot y$, the variables x and y must be different;
- the left part of a $\ltimes$ must be of the shape $x \cdot y$.

If we reject the MIX rule, only the left part of $\ltimes$ which can be cancelled is problematic. We present the tests for MLL extended with $A \Rightarrow B$ by translating the hyperedges of the lower part of the proof-structure:

- $A^\star = [-t.\mathtt{addr}_{\mathcal{S}}(A), +c.q_A(x \cdot y)]$ where A comes from the left part of $\ltimes$ or $\invamp$;
- $(A \invamp_x B)^\star = [-c.q_A(x \cdot x), -c.q_B(x), +c.q_{A \invamp B}(x)]$;
- $(A \invamp_1 B)^\star = [-c.q_A(x \cdot \mathtt{1}), -c.q_B(x), +c.q_{A \invamp B}(x)]$;
- $(A \ltimes_L B)^\star = [-c.q_A(x \cdot y), +c.q_{A \ltimes B}(x \cdot y)]$ (cancelling);
- $(A \ltimes_R B)^\star = [-c.q_B(x), +c.q_{A \ltimes B}(x)] + [-c.q_A(x \cdot y)] + [-c.q_A(x' \cdot y')]$ (unless $x = x'$);
- $(\underline{A})^\star = [-c.q_A(x)]$ where $\underline{A}$ is a conclusion.

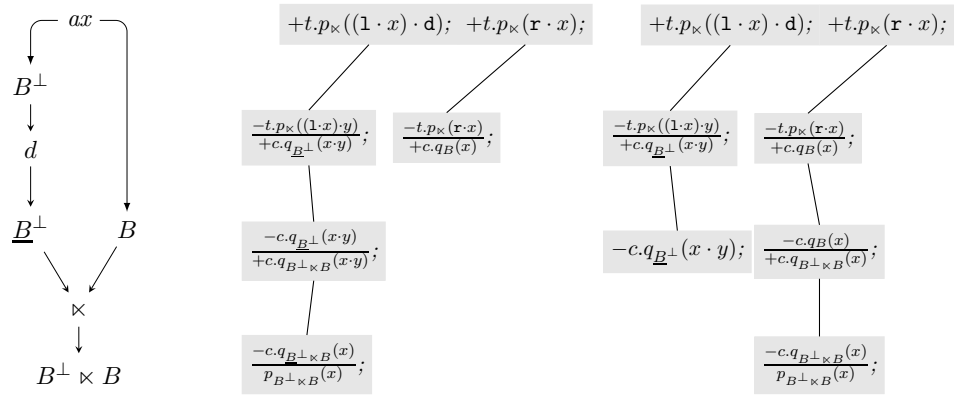
The two tests for $\invamp$ force the presence of different variables. The test $\ltimes_R$ only cancels the non-linear formulas. The star $[-c.q_A(x' \cdot y')]$ is only used when x' can be instanciated with something different from the variable x. This is a technical hack which forces the t in the $t \cdot u$ cancelled to be x.

Let us focus on the test $\ltimes_L$ which relies on the mechanisms of stellar resolution. Structural rules only appear on the left of $\ltimes$ and we need both acyclicity and connectedness. Remark that we ask for this test to be cancelling (the interaction normalises into the empty constellation $\emptyset$). The test connects the conclusion of $\ltimes$ to its left part. The possible cases for $A \ltimes B$ are the following ones:

- the stars connected to A form an independent connected component, form an open diagram and the normal form is non-empty. If the corresponding diagram is closed then the empty star [] appears in the normal form.
- a copy of A is connected to $A \ltimes B$ by the bottom (with a cut). In this case, we have a cycle and infinitely many correct diagrams. The constellation is not strongly normalising.
- the left part of $\ltimes$ is connected to the right part from the top. The whole constellation only normalises into $\emptyset$ when every formulas are connected. Cycles always involve exact matching, hence we must have acyclicity of the dependency graph.

We then ask that for each tests (except the ones containing $\ltimes_L$ which should be cancelled) we get the star of conclusions (as usual) but we also require that we only get the non-linear conclusions in particular. Hence, for a proof of $\vdash \Gamma, \underline{\Delta}$, we should obtain the star of conclusions coming from Γ. This is due to a recurrent problem (not necessarily a flaw) of GoI: it is impossible to act on the auxiliary conclusions of boxes and to always preserve the paths of proofs with non-linear conclusions. Since we do not consider full exponentials, non-linear conclusions $\underline{A}$ cannot appear as conclusion of a proof anyway.

Example 16 (Correctness of identity function)**.** *We check the correctness of the* λ*-term* $\lambda x.x$*.*



We obtain two diagrams corresponding to the two possible correction graphs. The left one using $\ltimes_L$ *is erased because* $+c.q_B(x)$ *has nothing to match. The right one using* $\ltimes_R$ *normalises into* $[p_{B^\perp\ltimes B}(x)]$*. Hence, the identity function is logically correct.*

8. New horizons for second order logic

A novelty of the Transcendental Syntax is the computational reconstruction of predicate calculus but also a redefinition of first and second order logic.

8.1. Girard's first and second order

Girard explained in a paper written in French [23] that second order logic is actually implicit in a lot of definitions of logic. For instance, the rule $(\vee e)$ for NJ implicitly requires a generic formula C:

$$\dfrac{\begin{array}{ccc} & [A] & [B] \\ \vdots & \vdots & \vdots \\ A\vee B & C & C \end{array}}{C}\vee e \qquad\qquad \dfrac{A \quad B}{A\wedge B}\wedge i$$

In theorems of propositional logic such as $A \Rightarrow B \Rightarrow A$, the formulas A and B are implicitly universally quantified and should be understood as $\forall A\ B.\ A \Rightarrow B \Rightarrow A$. The same thing occurs for P which is external in $\forall x.P(x)$. This is how Girard justifies the treatment of the additives and exponentials in second order for proof-nets because they use rules which are morally of second order.

If we look at the rule $\wedge i$, it only reunites hypotheses without any genericity. Since MLL proof-nets are hypergraphs linking formulas, it is similar. Even the atoms are concrete objects: simple vertices with non-essential labels. First order logic hence corresponds to the part of logic which uses nothing more than "what is already there" and second order logic as the part of logic referring to the set of all propositions. Therefore, predicate

calculus is actually part of second order logic while MLL extended with $\Rightarrow$ and $\forall$ is purely first order. We have the new following distinction between first and second order:

First order $\otimes, \parr, \multimap, \forall, \exists, \ltimes, \rtimes, \Rightarrow$;

Second order $?, !, \oplus, \&, \forall, \exists, \mathbf{1} = !\top, \bot = ?\mathbf{0}, \mathbf{0} = \forall X.X, \top = \exists X.X$.

where the additives (not detailled here) are defined as follows:

$$A\&B := \exists X.!(X \multimap A) \otimes !(X \multimap B) \otimes X$$

$$A \oplus B := \forall X.(A \multimap X) \Rightarrow ((B \multimap X) \Rightarrow X).$$

These new formulations for additives comes from consideration of the rules for $\wedge$ and $\vee$ in **NJ** which are morally of second order.

8.2. Girard's derealism

Girard's *derealism* corresponds to a new status for proofs, coming from a new treatment of second order quantification. His intuition comes from the second order definition of natural numbers:

$$\mathbf{nat} := \forall X.(X \multimap X) \Rightarrow (X \multimap X).$$

He claims [23] that $\mathbf{nat}^\perp$ corresponds to iteration/induction on natural numbers and that if testing for $\mathbf{nat}^\perp$ was finite, we could determine which iterations are licit, which is problematic. Further details are needed in order to make this idea explicit but it is not investigated here. Girard has the intuition that reasoning should be finite, hence we should preserve a finite testing. This leads to quantified entities being part of the proof itself. For instance, T should be part of a proof of $\exists X.A$ because it itself comes from $A[X := T]$ where T was present but disappeared.

A proof is now a tuple $(\Phi_V \uplus \Phi_M, \Phi_C, \Phi_F)$ where Φ_M is called the *mould* or *witness* of the proof and corresponds to a test associated to witnesses (the T associated to X in $\exists X.A$).

Since X in $\exists X.A$ may appear positively or negatively (X or $X^\perp$), the witness comes in two versions and we are interested in a balance between them which is not discussed here but the reader can find more details in the fourth article of Transcendental Syntax [24].

For technical reasons and to add more combinatorial complexity to proofs, we distinguish two classes of rays:

- *objective* rays which are the usual rays;
- *subjetive* rays allowing internal colours, for instance $+a(-b(x))$. As a consequence, we can to consider a subjective logical constant, similar to $\forall$. Girard calls it $\exists$ (wo). It can be tested with the test $[-t.p_\exists(t \cdot x), p_\exists(x)]$ where t is subjective (e.g $+a(x)$) ensuring the presence of an internal colour.

A star is objective if it only has objective rays and subjective if it only has subjective rays. Otherwise, it is *animist* (a mix of objective and subjective). An *épure* is a constellation without animist star, i.e. it can be put into the form $\Phi_O \uplus \Phi_S$ where Φ_O only contains objective stars and Φ_S subjective ones. This makes the idea of correct proof clearer: it has to be made of an épure with an objective part only containing binary stars (they represent axioms).

The interesting feature of this new combinatorics is that the status of stars (objective/subjective/animist) can change during the execution. Another consequence is that we can design a conduct $\mathbf{0}$ containing a constellation which can interact with other constellations but which is not correct (because of animist stars). This gives this constant a computational content, something which is usually not considered.

8.3. Additive neutrals

We illustrate the mechanisms of witnesses and subjective rays with additive neutrals by following the third paper of Transcendental Syntax [22]. We define the following rays:

$$C_\top(x) := p_\top(\mathtt{c} \cdot x) \qquad L_\top(x) := p_\top(-t(\mathtt{l} \cdot x)) \qquad R_\top(x) := p_\top(-t(\mathtt{r} \cdot x))$$

The neutral element $\top$ is defined by the orthogonal of the following tests where the second one is cancelling (as for $\ltimes_L$ in the exponentials):

$$\top_1 : \left[\frac{-t.C_\top(x), -t.L_\top(x)}{}\right] + \left[\frac{-t.R_\top(x)}{p_\top(x)}\right] \qquad \top_2 : \left[\frac{-t.C_\top(x), -t.L_\top(x)}{p_\top(x)}\right] \quad \text{(cancel)}$$

The constellation $[+t.C_\top(x)] + [+t.L_\top(x), +t.R_\top(x)]$ is an épure passing the tests. However, it is not a correct proof because the objective part is unary and not binary.

If we take $[+t.C_\top(\mathtt{l} \cdot x), +t.C_\top(\mathtt{r} \cdot x)] + [+t.L_\top(\mathtt{l} \cdot x)] + [+t.L_\top(\mathtt{r} \cdot x), +t.R_\top(x)]$ instead, then it is a correct and also passes the test. In fact, it is possible to check that for any $\Phi \in \top$, we have $\Phi \in\ ?(\forall \otimes \exists) \parr \exists = (\forall \otimes \exists) \ltimes \exists = (\forall \parr \exists) \Rightarrow \exists$ because of the possibility of duplicating $C_\top$ and $L_\top$ and still pass the tests.

We define the following rays:

$$C_{\mathbf{0}}(x) := p_{\mathbf{0}}(\mathtt{c} \cdot x) \qquad L_{\mathbf{0}}(x) := p_{\mathbf{0}}(-t(\mathtt{l} \cdot x)) \qquad R_{\mathbf{0}}(x) := p_{\mathbf{0}}(-t(\mathtt{r} \cdot x))$$

The neutral element $\mathbf{0}$ is defined from the following tests:

$$\mathbf{0}_1 : \left[\frac{-t.C_{\mathbf{0}}(x)}{}\right] + \left[\frac{-t.L_{\mathbf{0}}(x), -t.R_{\mathbf{0}}(x)}{p_{\mathbf{0}}(x)}\right] \qquad \mathbf{0}_2 : \left[\frac{-t.L_{\mathbf{0}}(x)}{}\right] + \left[\frac{-t.C_{\mathbf{0}}(x), -t.R_{\mathbf{0}}(x)}{p_{\mathbf{0}}(x)}\right]$$

$$\mathbf{0}_3 : \left[\frac{-t.R_{\mathbf{0}}(x)}{p_{\mathbf{0}}(x)}\right]$$

The constellation $[+t.R_{\mathbf{0}}(x)]+[+t.C_{\mathbf{0}}(x), +t.L_{\mathbf{0}}(x)]$ (containing an animist star) passes the tests but there is no épure. The reason is that the tests are designed so that the orthogonal of the tests must have a mix of objective and subjective rays. This new combinatorics for proofs allows to speak about logical coherence. Since $\mathbf{0} = \top^\perp$, we should have that for any $\Phi \in \mathbf{0}$, we have $\Phi \in (\forall \parr \exists) \oslash \exists$ (by following the expression of $\top$ in terms of $\forall$ and $\exists$).

8.4. Predicate calculus

Because of Girard's new distinction between first and second order, I choose to use the term "predicate calculus" instead of "first order logic" to avoid confusion.

As remarked before, the predicate calculus is part of second order logic but few conceptual choices remain. Girard remarks that by looking at Leibniz's equality defining $a = b$ as $\forall X.X(a) \Leftrightarrow X(b)$, the predicate X would play no role if written as a proof-net. We can connect a and b directly and X is seen as a sort of modality restricting connexions instead. This leads to the idea of representing terms/individuals as multiplicative formulas and equality as linear equivalence $A \equiv B$ defined as $(A \multimap B) \otimes (B \multimap A)$. In terms of derealism, the witnesses encode formats corresponding to multiplicative formulas. The encodings are free but for individuals, we need to ensure technical properties such as the injectivity of the encoding in order to have that $f(x) = f(y)$ implies $x = y$.

We do not give intuitive details about how it is treated in the Transcendental for the moment and refer to Girard's third article [22].

9. Beyond logic

What follows is a possibly exaggerated discussion about how logic could actually be more than "our" logic. Something I find quite remarkable is that the stellar resolution places the link between computation and logic in the broad world of emergence and complex systems. We compute by non-deterministic local interactions between independent entities with a notion of information propagation. A logic emerges from the behaviour of constellations. Since it is close to tiling models, we can also expect relationships with sequential dynamical systems and graph dynamical systems.

At the time of the document, I believe than no one truly understands the nature of logic. The Geometry of Interaction and the Transcendental Syntax showed that we can speak about linear logic with simple topology/geometry/dynamics by considering the locations of entities, their links, cycles, interaction, paths etc (a very general setting exists in Seiller's graphings [41]). These are rather tangible things reminiscent of interaction in the biological/physical world (for instance chemical reaction networks [29]).

I wonder if it is possible to consider a setting even more general than stellar resolution, for instance a dynamical system for which attractors would have a logical meaning. It may be possible that what we know about logic is only a particular case of something bigger, a *theory of interaction* as mentioned several times already [1, 17].

Indeed, we must take several additional things into account. First, logic has a subjective side because we look at computation from a specific point of view and a choice of what a sound/successful computation means, which depends on the use of objects and our own perception/goals. It probably makes purely mathematical techniques insufficient (unless we speak about the objective part of logic, i.e. computation). A more conceptual/philosophical study and connexions with other fields (typically physics and biology) may be useful and seems possible. Moreover, we can also wonder if logic exists beyond the computable.

Finally, I am curious about whether all of that has something to say at all about computational complexity (some hints seem to exists in Seiller's works [42]). Due to the barriers of problems such as the separations of classes, it may be necessary to look for new definitions of logic and computation.

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] Matteo Acclavio and Roberto Maieli. Generalized connectives for multiplicative linear logic. In *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[4] Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995.

[5] Clément Aubert and Marc Bagnol. Unification and logarithmic space. In *Rewriting and Typed Lambda Calculi*, pages 77–92. Springer, 2014.

[6] 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.

[7] Marc Bagnol. *On the resolution semiring*. PhD thesis, Aix-Marseille Université, 2014.

[8] Patrick Baillot and Marco Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2):1–31, 2001.

[9] Emmanuel Beffara. A concurrent model for linear logic. *Electronic Notes in Theoretical Computer Science*, 155:147–168, 2006.

[10] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)*. PhD thesis, Paris 7, 1990.

[11] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989.

[12] Vincent Danos and Laurent Regnier. Proof-nets and the hilbert space. *London Mathematical Society Lecture Note Series*, pages 307–328, 1995.

[13] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227(1-2):79–97, 1999.

[14] Jean-Yves Girard. Multiplicatives. 1987.

[15] Jean-Yves Girard. Geometry of interaction II: deadlock-free algorithms. In *International Conference on Computer Logic*, pages 76–93. Springer, 1988.

[16] 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.

[17] Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.

[18] Jean-Yves Girard. Geometry of interaction III: accommodating the additives. *London Mathematical Society Lecture Note Series*, pages 329–389, 1995.

[19] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical structures in computer science*, 11(3):301, 2001.

[20] Jean-Yves Girard. Geometry of interaction IV: the feedback equation. In *Logic Colloquium*, volume 3, pages 76–117, 2006.

[21] Jean-Yves Girard. Geometry of interaction V: logic in the hyperfinite factor. *Theoretical Computer Science*, 412(20):1860–1883, 2011.

[22] Jean-Yves Girard. Transcendental syntax III: equality. 2016.

[23] Jean-Yves Girard. La logique 2.0. 2018.

[24] Jean-Yves Girard. Transcendental syntax IV: logic without systems. 2020.

[25] Esfandiar Haghverdi and Philip Scott. A categorical model for the geometry of interaction. *Theoretical Computer Science*, 350(2-3):252–274, 2006.

[26] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.

[27] 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.

[28] Nataša Jonoska and Gregory L McColm. Flexible versus rigid tile assembly. In *International Conference on Unconventional Computation*, pages 139–151. Springer, 2006.

[29] Jürgen Jost and Raffaella Mulas. Hypergraph laplace operators for chemical reaction networks. *Advances in mathematics*, 351:870–896, 2019.

[30] Stephen Cole Kleene. On the interpretation of intuitionistic number theory. *The journal of symbolic logic*, 10(4):109–124, 1945.

[31] Robert Kowalski. A proof procedure using connection graphs. *Journal of the ACM (JACM)*, 22(4):572–595, 1975.

[32] JL Krivine, PL Curien, H Herbelin, and PA Melliès. Interactive models of computation and program behavior. 2009.

[33] Alexander Leitsch. *The resolution calculus*. Springer Science & Business Media, 2012.

[34] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.

[35] Matthew J Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

[36] Laurent Regnier. *Lambda-calcul et réseaux*. PhD thesis, Paris 7, 1992.

[37] Lionel Rieg. *On forcing and classical realizability*. PhD thesis, Ecole normale supérieure de lyon-ENS LYON, 2014.

[38] Thomas Seiller. Interaction graphs: multiplicatives. *Annals of Pure and Applied Logic*, 163(12):1808–1837, 2012.

[39] Thomas Seiller. *Logique dans le facteur hyperfini: géometrie de l'interaction et complexité*. PhD thesis, Aix-Marseille Université, 2012.

[40] 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.

[41] Thomas Seiller. Interaction graphs: Graphings. *Annals of Pure and Applied Logic*, 168(2):278–320, 2017.

[42] Thomas Seiller. Interaction graphs: Non-deterministic automata. *ACM Transactions on Computational Logic (TOCL)*, 19(3):1–24, 2018.

[43] Sharon Sickel. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers*, (8):823–835, 1976.

[44] Kazushige Terui. Computational ludics. *Theoretical Computer Science*, 412(20):2048–2071, 2011.

[45] Wolfgang Thomas. On logics, tilings, and automata. In *International Colloquium on Automata, Languages, and Programming*, pages 441–454. Springer, 1991.

[46] Hao Wang. Proving theorems by pattern recognition —II. *Bell system technical journal*, 40(1):1–41, 1961.

[47] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, Citeseer, 1998.

Appendix A Not a pure waste of paper (light version)

In this section, I would like to clarify Girard's terminology. To read after being a bit familiar with the content of this paper.

Analytic In reference to Kant's epistemology. A space of meaningless and computational entities. A ground for logic where objects are expressed before getting a logical meaning. A good analytic should be as natural and simple as possible.

Anarchy Refers to types constructed by realisability where types are arbitrary sets of computational objects and type checking is potentially infinite or untractable. Subtyping and intersection types naturally exist in this setting.

Animist In reference to Japanese traditional beliefs where lifeless entities can have both a subjective part (a spirit inhabiting it) and an objective part (its physical materialisation). In the Transcendental Syntax, a second order proof can have both a subjective (an existential witness coming with its own tests certifying its logical correctness) and objective side (its computational content).

Apodictic State of absolute certainty where only finitely many tests with a tractable testing are sufficient for type checking and that the adequacy lemma is tree (a formula label guarantee membership in some wanted behaviour). It happens for MLL but not in general.

Axiom Group of physical locations (usually two).

Behaviour A type or formula $\mathbf{A}$ constructed by realisability with satisfies the biorthogonal closure $\mathbf{A} = \mathbf{A}^{\perp\perp}$ ensuring that $\mathbf{A}$ is characterised by a (potentially infinite) set of tests and that it is morally testable.

Conduct Originally had a different meaning [21] but is the same thing as a behaviour in Seiller's works and this paper.

Constat Irreducible object (for instance, the normal form of a λ-term).

Cut Link connecting two physical locations.

Cut-elimination Resolution of addresses applied on a cut in order to identify some locations. This is the computational content of proof and it defines the dynamics happening in logic, allowing to go from an a reducible object to an irreducible one. This is an access to explicit knowledge by computation.

Derealism Approach of Transcendental Syntax (and especially second order logic) for which proofs do not exist by themselves but as composite entities containing a computational object and tests ensuring its logical correctness.

Dichology Formula or type constructed by realisability.

Dictatorship Refers to typing by finite testing (à la Church) where logic would be especially constrained. In particular, it is not possible to look outside of our definitions. Absolute certainty exists but only relatively to the definitions of logic or the tests provided.

Epidictic Refers to reasonable but not absolute certainty occurring in logic. Certainty is often inaccessible due to infinite testing but using a finite sample of tests provide an acceptable trust (for instance trust in the foundations of mathematics).

Epistate Computational object generalising proofs and tests asserting their logical correctness.

Epure Correct proof in second order logic. In reference to a French word representing a drawing with several views of the objects. These several views are represented by the different tests included with the existential witness provided in the second order proof.

Essentialism Philosophy in which the essence of objects comes before their existence. In the Transcendental Syntax, it refers to typing by testing where arbitrary tests defines the type of computational objects.

Existentialism Philosophy in which the essence of objects is defined by their existence. In the Transcendental Syntax, it refers to typing by realisability where types correspond to classification of computational objects according to how they interact with each other.

Factory Typing by testing. The sets of tests are usually required to be finite and testing should be tractable.

First order logic Fragment of logic where no genericity exist and reasoning is purely specific. Logical rules are usually physical moving of specific entities (reuniting two formulas together for the conjunction).

Format and Bureaucracy Refers a system of tests which produces a constraint on computational objects and format them by arbitrary choices. For instance, a human is "male" when it passes some chosen tests depending of a historical and sociological context. A format can be more or less strict but is usually finitely checked and tractable.

Formula Syntactic label materialising an assertion which usually formalises a question or a property. It can be understood as a contraint or a synthetic description.

Hidden files Often wrong or ill-behaving entities not considered in traditional proof theory and whose existence is computationally relevant in logic. For instance, the constant フ.

Interaction Phenomena occurring when opposing two compatible objects. In the Transcendental Syntax, this is represented by the execution of constellation.

Locativity Approach take in Ludics and the latest articles of GoI where the physical location, referred to by an address is given a great importance. For instance, the formulas in a sequent calculus proof are inessential labels which can be replaced by addresses (for instance natural numbers). A proof becomes a manipulation of addresses.

Monism Approach in which every entities is represented as objects of the same kind. For instance, proof-nets and their tests become constellations but so are circuits, automata, tile systems and Turing machines. It is the opposite of duality where we distinguish objects of two kinds (for instance program vs environment, syntax vs semantics etc).

Orthogonality Binary symmetric relationship formalising the compatibility of two computational objects. In the Transcendental Syntax, it also formalises the idea of "passing a test". We can design any orthogonality relationship we want and it leads to different computational analyses or models of logic.

Performance Procedure leading a reducible computational object to an irreducible one. It is essential in the theory of computation since reducible objects cannot be reduced to their results because of the computational indecidability (for instance, the halting problem).

Proof Object formalising an answer to a formula. It can takes several forms (tree, graph, number etc) but usually depends on the requirement of the formula and what is considered an answer to the formula. Logic is usually interested in the adequacy between proofs and formulas.

Realism Belief in which the meaning of reality of a concept exists externally to it. In particular, in the case of logic, when a system is mistakenly considered as a correct representation of reality.

Reality Refers to an unreachable entity which is approximated by analytic (the stellar resolution). In particular, the stellar resolution formalises our intuition of space and time (transition of information within a topological structure). The synthetic part of logic then organises our representation of reality depending on a point of view. It is also the belief that semantics is sufficient to capture the logical activity.

Second order logic A fragment of logic which uses generic reasoning. For instance, the rule of the modus ponens (from A and $A \Rightarrow B$ follows B) works for any formula A and B. Girard claims that a great part of traditional logic is actually second order (propositional logic and first order logic in particular).

Semantics System in which we associate a meaning to (often computational) objects. For instance, denotational semantics associate a mathematical object to proofs. Such object represents the meaning of the proof, which is purely syntactic.

Scientism Refers (usually negatively) to a belief in which science and more especially logic can explain everything. In particular, certainty is absolute and logic has the role of accessing knowledge and solving philosophical problems. It is then sufficient to compute the answer to access knowledge.

Stars and constellations Computational objects used in the Transcendental Syntax. The terminology comes from the fact that the stars are objects with can be connected to other ones along its branches. It could also be called atom or molecule which would probably be more appropriate but less poetic.

Synthetic In reference to Kant's epistemology. Space where a meaning is given to meaningless computational objects. In the Transcendental Syntax, it is represented by types by testing and types by interaction (using realisability techniques). It is what allow us to design formulas/types or questions/properties.

Test Way to assert that an object has a specific property or computational behaviour we expected. As in unit testing used for programming, it can represent specifications we wish for a computational object.

Type Same as formula by the Curry-Howard correspondence but more used in the context of programming and the theory of computation where types are used as labels preventing ill behaviours or unwanted situations to happen during computation.

Use Refers to how logical definitions can be used in practice. It is materialised by the cut which connects two proofs and the cut-elimination which makes their interaction effective. According to Girard's, Gödel's incompleteness theorems state that tests cannot always guarantee the use we wish for.

Vehicle The computational content of a proof. In the theory of proof-nets, it corresponds to the set of axioms of a proof-structure. It is the part which interacts with tests. The tests will then ensure that the vehicle has a specific computational behaviours when interacting with other objects (the vehicle of another proof). For instance, MLL+MIX correctness guarantees strong normalisation of cut-elimination.

Wire Analogy used by Girard in order to talk about logic in a more concrete and down-to-earth way. In particular, the theory of proof-nets sees proofs as wiring of formulas and the cut-elimination as a resolution of wiring. The Transcendental Syntax extends this analogy because, in the stellar resolution, wires are terms $c(x)$ which can be divided into infinitely many sub-wires, for instance $c(\mathtt{l} \cdot x)$ and $c(\mathtt{r} \cdot x)$, while keeping finite objects.