Stellar Resolution: Multiplicatives - for the linear logician, through examples

Boris Eng

Stellar Resolution: Multiplicatives

For the linear logician, through examples (version 4)

Boris Eng

The stellar resolution is an asynchronous model of computation used in Girard's Transcendental Syntax [19, 22, 20, 21, 24] which is tiling model with an evaluation of tiling based on Robinson's first-order clausal resolution [32]. By using realisability techniques for linear logic (similarly to phase semantics and ludics), we obtain an extensible model of multiplicative linear logic (MLL) able to represent proofs, cut-elimination, formulas/types, correctness and provability very naturally. Girard's philosophical justification of these works comes from Kantian inspirations: the Transcendental Syntax appears as a way to talk about *the conditions of possibility of logic*, that is the conditions from which logical constructions emerge out of the meaningless.

Contents

1 Frequently Asked Questions

- *What is the Transcendental Syntax about?* The Transcendental Syntax is a programme initiated by Girard which can be seen as the successor of his Geometry of Interaction (GoI) programme [15, 14, 13, 16, 17, 18] studying linear logic from the mathematics of the cut-elimination. In the same idea, the Transcendental Syntax aims at giving a fully computational foundation for logic where entities such as formulas, proofs, correctness, truth are reconstructed from a computational model as in classical realisability for instance, where the realisers are kind of logic programs seen as tile sets. One may see it as a kind of reverse engineering of linear logic. In particular, it presupposes the importance of linear logic in the study of logic in general.

- *Where does the model of stellar resolution come from?* We use "stellar" for Girard's terminology of *stars and constellations* and "resolution" for its similarities with other resolution-based models [28, 34]. The GoI began with a mathematical study of the cut-elimination procedure through the use of infinite-dimensional spaces and operator algebras in order to handle the non-linearity of full linear logic. In the third article of GoI [16], Girard introduced a simplification based on first-order unification: the model of flows [6] which is basically unary first-order resolution, well-known among computer scientists. The stellar resolution is simply an extension of this model which, unlike flows, is able to speak about correctness in a satisfactory way. One may choose another model of computation as a basis of the Transcendental Syntax but the stellar resolution is a natural and convenient one. In particular, the use of terms and unification internalises the infinity present in the GoI.

- *Isn't it identical to first-order resolution or 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). By looking at examples presented in this document, we can remark that it also computes differently. Moreover, our model will be extended in future works, thus justifying the use of a new name.

- *Is it related to any other works?*
  - The stellar resolution generalises *flexible tiles* [27, 26], a model of computation coming from DNA computing [37]. This model is itself able to simulate usual tiling-based models of computation such as Wang tiles [36] or abstract tile assembly models [31].

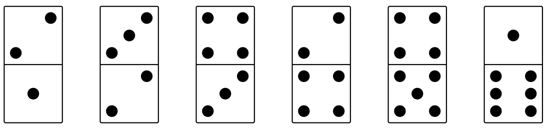
- The stellar resolution can also be seen as a generalisation of the model of flows used in the GoI and of Seiller's interaction graphs [33]. The reconstruction of types/formulas follow the constructions of the model of realisability for linear logic.

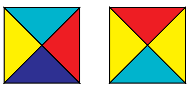
- *Why is it interesting?*
  - From our realisability construction, one can imagine methods of typing and implicit complexity analysis of tiling-based models used in DNA computing. It seems that our model is rather convenient to describe models of computation based on local interactions which are also dynamical systems.
  - We can generalise flows and interaction graphs which have an applications in implicit computational complexity [7, 4, 5]. Some of these works encode some notions of automata and it seems that the stellar resolution is linked to a very general notion of automata seen as non-deterministic tiling [35].
  - The Transcendental Syntax programme should be able to produce a more refined notion of type/formula but also, more interestingly, a computational and axiom-free reformulation of predicate logic with a better treatment of equality (instead of a predicate), a finite treatment of quantifiers, and a logical and computational status for first-order individuals (encoded as multiplicative propositions).

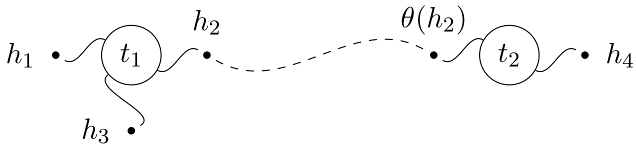
2 Stellar Resolution

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

2.1 Tiling models

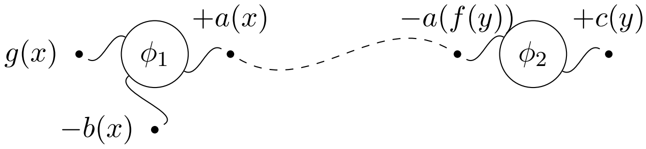
Dominos  Dominos are quite common and easy-to-understand objects. We have bricks with two faces containing a natural number. We can connect two faces if they have the same number. You can imagine several games with different rules from such objects (for instance, allow or forbid rotation).

Wang tiles [36]  We consider more general objects. 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.



Flexible tiles [27, 26] What if we generalise even more? Instead of imposing planarity to tilings, we allow tiles to have flexible sides selected within a 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.

2.2 Stars and constellations



The stellar resolution model can be understood as a flexible tiling model with polarised or unpolarised terms with a head symbol.

A *star* (tile) is a finite and non-empty multiset of rays $\phi = [r_1, ..., r_n]$ and a *constellation* $\Phi = \phi_1 + ... + \phi_m$ (tile set) 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.

By using occurences 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 change 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 and will necessarily become an output of the computation.*

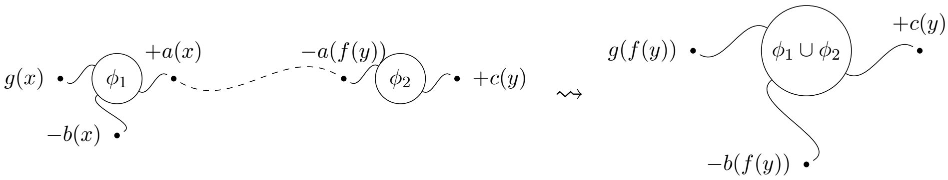
```
add(0, y, y).
add(s(x), y, s(z)) :- add(x, y, z).
?add(s^n(0), s^m(0), r).
```

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

```
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 first define an evaluation of diagrams (tilings) called *actualisation*. If stars are 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.

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 (unconnected rays).

In order to get a successful evaluation we need our diagrams to satisfy few properties. They need to be:

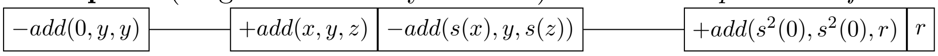
Connected because we would like the diagrams to contract into a single star;

Open meaning that we have unconnected rays because otherwise we would get the empty star which is not interesting (no visible output);

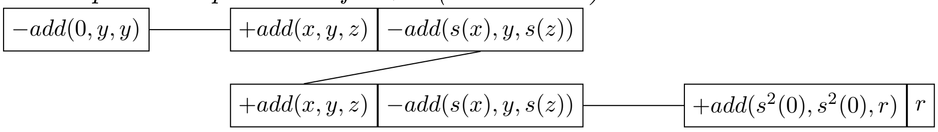
Saturated meaning that it is impossible to extend the diagram with more occurrences of stars. It represents a kind of maximal/complete computation;

Correct meaning that the actualisation does not fail.

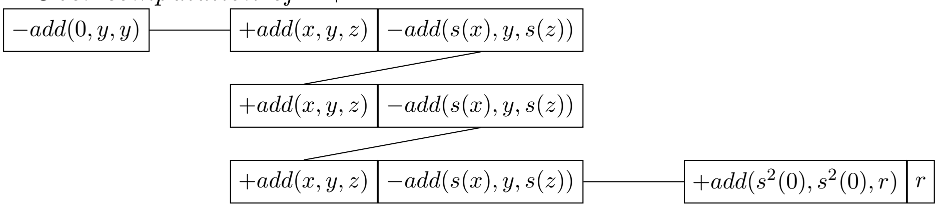
Example 2 (diagrams for unary addition)**.** *Partial computation of* $2+2$ *(0 recursion):*



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

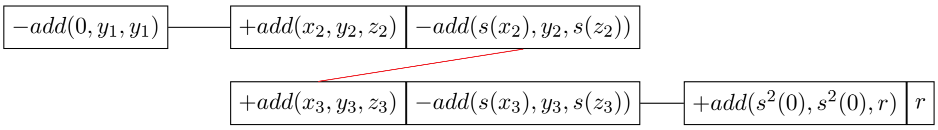


Over computation of $2+2$*:*

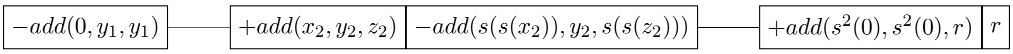


Note that in the case of $\Phi_{\mathbf{N}}^{n,m}$*, there is infinitely many saturated diagrams but only one is correct: the one corresponding to a successful computation of* $n+m$*.*

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):*

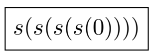


$$\downarrow \theta = \{x_3 \mapsto s(x_2), y_3 \mapsto y_2, z_3 \mapsto s(z_2)\}$$



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

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



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 [30] in order to obtain a finale substitution:

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

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

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

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

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

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

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

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

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

$$\to^* \{s(s(s^2(0))) \stackrel{?}{=} r\}$$

$$\to^* \{r \stackrel{?}{=} s(s(s^2(0)))\}$$

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. This can be thought as a chemical reaction having an effect on the non-involved entities, leaving a kind of residual.*

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

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 [29] corresponding to a certain class of clauses we can infer using the resolution rule.

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 $\Phi_{\mathbf{N}}^{2,2}$ (example 1), one can check that we have $\texttt{Ex}(\Phi_{\mathbf{N}}^{2,2}) = [s^4(0)]$ 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 more examples of how our model compute. It seems that it is especially convenient for models of computation which are also dynamical systems and compute by local interactions between independant objects.

3.1 Non-deterministic finite automata

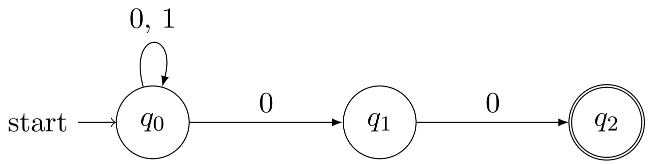
The idea is to represent the transitions by binary bipolar stars.

Let Σ be an alphabet and $w \in \Sigma^*$ a word. If $w = \varepsilon$ then $w^\bigstar = [+i(\varepsilon)]$ and 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 $[-a(\varepsilon, q_f), accept]$.
- for each $q \in Q, c \in \Sigma$ and for each $q' \in \delta(q, c)$ with $c \in \Sigma$, 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:

$$A^\bigstar = [-i(w), +a(w, q_0)] + [-a(\epsilon, q_2), accept]+$$

$$[-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)]+$$

$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)]$$

The set of saturated correct diagrams corresponds to the set of non-deterministic runs. With the word $[+i(0 \cdot 0 \cdot 0)]$ only the run $q_0q_0q_1q_2$ is lead to the finale state. The corresponding diagram will actualise into $[accept]$. The other diagrams will be incomplete and produce "junk stars" containing information about where the run stopped. We get $\texttt{Ex}(A^\bigstar + [+i(0 \cdot 0 \cdot 0)]) = [accept] + \Phi_J$ where Φ_J is an irrelevant constellation we keep hidden, meaning that the word is accepted.

3.2 Boolean circuits

The idea is to first encode an hypergraph representing the structure of a boolean circuit then to connect the resulting constellaion to another one containing the implementation of logic gates.

$$VAR(Y,i) := [-val(x), Y(x), +c_i(x)]$$
$$SHARE(i,j,k) := [-c_i(x), +c_j(x), +c_k(x)]$$
$$AND(i,j,k) := [-c_i(x), -c_j(y), -and(x,y,r), +c_k(r)]$$
$$OR(i,j,k) := [-c_i(x), -c_j(y), -or(x,y,r), +c_k(r)]$$
$$NEG(i,j) := [-c_i(x), -neg(x,r), +c_j(r)]$$
$$CONST(k,i) := [+c_i(k)] \quad QUERY(k,i) := [+c_i(k), R(k)]$$

where i, j, k are encodings of natural numbers representing identifiers and where we have a star $VAR(Y)$ for each variable Y we want 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:

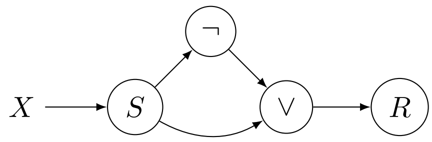
$$\Phi^{\mathcal{PL}} = [+val(0)] + [+val(1)] + [+neg(0,1)] + [+neg(1,0)] +$$
$$[+and(0,0,0)] + [+and(0,1,0)] + [+and(1,0,0)] + [+and(1,1,1)]$$
$$[+or(0,0,0)] + [+or(0,1,1)] + [+or(1,0,1)] + [+or(1,1,1)]$$

We can observe that by changing the module, we can adapt the model and obtain arithmetic circuits. Notice that syntax and semantics live in the same world (they are represent as objects of 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 $QUERY(1)$ ask for satisfiability.

For instance, 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 $\Phi_{em} + \Phi^{\mathcal{PL}}$ will normalise generating two diagrams: one corresponding to the input 0 and another one for 1. We obtain $\mathtt{Ex}(\Phi_{em} + \Phi^{\mathcal{PL}}) = [X(0), R(1)] + [X(1), R(1)]$ telling us that for the two valuations $x \mapsto 0$ and $x \mapsto 1$, the circuit output $R(1)$.

4 Geometry of Interaction for MLL

The geometry of interaction study generalisations of proof-nets by keeping only their essential parts and reconstruct linear logic from the computational content of the cut-elimination procedure.

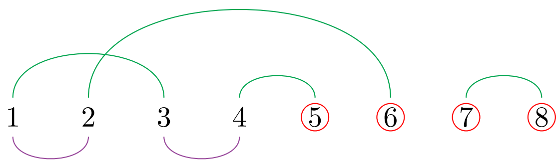
The term "geometry of interaction" also refers to categorical frameworks for linear logic [25, 1] but also combinatorial ones with the token machine [9, 3]. In this document, we are interested in Girard's original programme [14, 13, 16, 17, 18].

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 conclusion of axiom rules. In the spirit of Ludics, these atoms are called *loci* (plural of *locus*) and they represent kind of physical locations within a proof.

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 [33] studied the geometry of interaction through connexions of graphs where cut-elimination becomes computation of maximal 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 $\texttt{Ex}(\Phi) = [5, 6] + [7, 8]$ 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 want to handle correctness in a satisfactory and natural way, we have to shift to partitions instead of permutations [12, 2]. Permutations can still be retrived: a permutation $\{x_1 \mapsto y_1, ..., x_n \mapsto y_n\}$ on $X \subseteq \mathbf{N}$ representing a proof naturally induces a partition of binary sets $\{\{x_1, y_1\}, ..., \{x_n, y_n\}\}$ in X.

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

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 an axiom-partition "block" against several switching-partitions "blocks" is sufficient to talk about correctness.

Since stars are not limited to binarity, we can naturally represent general partitions: $[-c(1), -c(2)] + [-c(3)] + [-c(4)]$ (notice the negative polarity in order to allow connexion with axioms). However, in this case, all diagrams are closed (no free rays). We have to specify where the conclusions are located: $[-c(1), -c(2), A \otimes B] + [-c(3)] + [-c(4), A^\perp \parr B^\perp]$.

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

$$\Phi_A = [+c(1), +c(3)] + [+c(2), +c(4)]$$

$$\Phi_{\parr}^L = [-c(1), -c(2), A \otimes B] + [-c(3)] + [-c(4), A^\perp \parr B^\perp]$$

We have $\texttt{Ex}(\Phi_A \uplus \Phi_{\parr}^L) = [A \otimes B, A^\perp \parr B^\perp]$ which is the star of conclusions. In that case, we say that Φ_A passes the test $\Phi_{\parr}^L$.

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 Interpretation of the computational content of MLL

We set a basis of representation $\mathbb{B}$ with unary symbols p_A for all formulas of MLL, and constants $\texttt{l}, \texttt{r}$ 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.

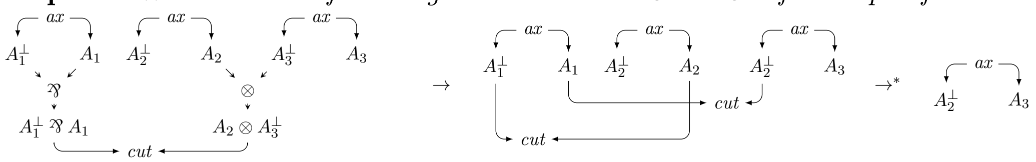
In order to take into account future works and have a nice definition of cut-elimination, we consider a more general interpretation than before. To simulate the dynamics of cut-elimination we translate the axioms and the cuts into stars:

1. A locus A becomes a ray $+c.p_A(t)$ if 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) encoded as a 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.

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:*

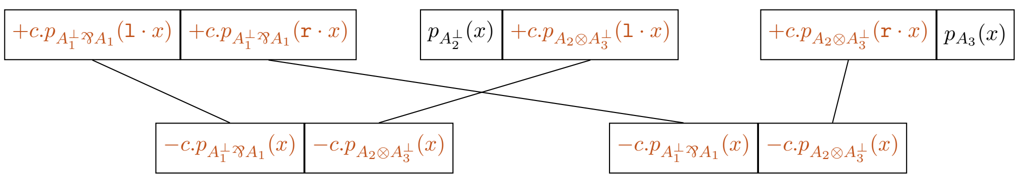


The address of $A_1^\perp$ is $p_{A_1^\perp \parr A_1}(\mathtt{l} \cdot x)$ because it is located on the left-hand side of $A_1^\perp \parr A_1$. The address of $A_3^\perp$ is $p_{A_2 \otimes A_3^\perp}(\mathtt{r} \cdot x)$ and the one for A_3 is $p_{A_3}(x)$. The proof-structure $\mathcal{S}$ is encoded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

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

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

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 $\mathcal{S}'$.

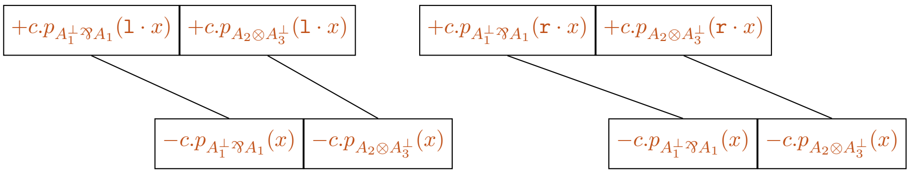
Example 7. *If we have the following reduction $\mathcal{S} \to \mathcal{S}'$ instead:*

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:

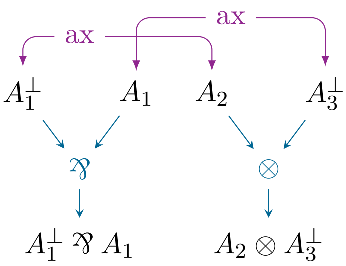


which is a closed diagram (thus incorrect), hence the normal form is the empty constellation.

6 Interpretation of the logical content of MLL

6.1 The correctness criterion of Danos-Regnier

We translate the correctness criterion of Danos-Regnier [8] by an encoding of hypergraphs (representing proof-structures) into constellations.



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 it).
- The lower part is basically the syntax trees of conclusions. It holds the logical content of the proof (formula/correcntess). The Danos-Regnier correctness criterion is obtained by swithings on the lower part and checking the properties of acyclicity and connectedness of each. This can be understood as *testing* the vehicle against a *set of test* we call the *format*. This test vehicle/format produces a certification: if all tests pass, we have a *proof-net*. This is Girard's "usine" (factory). Note that testing is symmetric: a format can also be seen as tested by a vehicle.

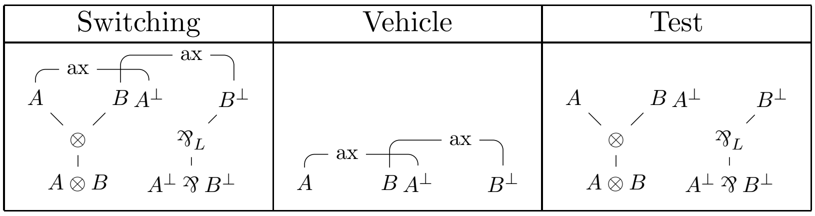


Figure 1: The vehicle of a proof-structure and a test corresponding to a Danos-Regnier switching graph.

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

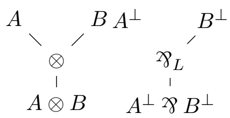
- $A^\bigstar = [-t.\mathtt{addr}_{\mathcal{S}}(A), +f.q_A(x)]$ where A is a conclusion of axiom,
- $(A \parr_L B)^\bigstar = [-f.q_A(x)] + [-f.q_B(x), +f.q_{A\parr B}(x)]$,
- $(A \parr_R B)^\bigstar = [-f.q_A(x), +f.q_{A\parr B}(x)] + [-f.q_B(x)]$,
- $(A \otimes B)^\bigstar = [-f.q_A(x), -f.q_B(x), +f.q_{A\otimes B}(x)]$,
- We add $[-f.q_A(x), p_A(x)]$ for each conclusion A.

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

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 core 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 shape as a switching graph.

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

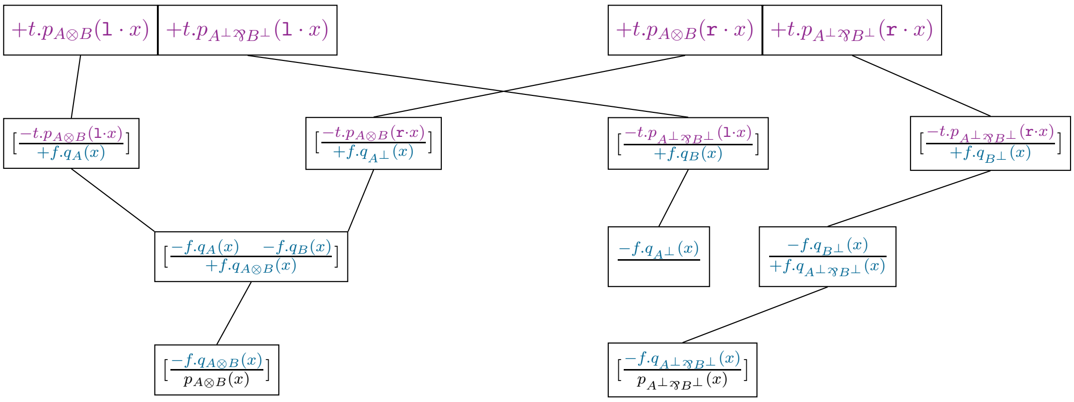


$$[\frac{-t.p_{A\otimes B}(\mathtt{l}\cdot x)}{+f.q_A(x)}] + [\frac{-t.p_{A^\perp\parr B^\perp}(\mathtt{l}\cdot x)}{+f.q_B(x)}] + [\frac{-t.p_{A\otimes B}(\mathtt{r}\cdot x)}{+f.q_{A^\perp}(x)}] + [\frac{-t.p_{A^\perp\parr B^\perp}(\mathtt{r}\cdot x)}{+f.q_{B^\perp}(x)}] +$$
$$[\frac{-f.q_A(x) \quad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

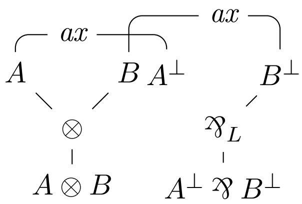
When connected to the vehicle

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

we obtain the following dependency graph:

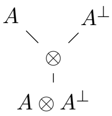


structurally corresponding to the following switching graph:



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 $[p_{A\otimes B}(x), p_{A^\perp ⅋ B^\perp}(x)]$.

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



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

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

6.2 What is a proof?

We started from very general untyped objects, the constellation 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 a kind of pre-made typing.

6.3 Interpretation of formulas

In order to reconstruct the types/formulas, we follow the construction of model of realisability.

- In the traditional type theory, types have a normative/constrictive role: they prevent some unwanted connexions to happen during the computation. This is Church-style typing to which Girard is often referring to as "essentialist".
- In our reconstruction of types, similarly to realisability, types have a descriptive role: they describe the common behaviour of a collection of computational objects (constellations in our case). We obtain a more refined idea of type but also a more complicated one to reason with. We can think of it as a kind of liberalised typing, free from constraints, but which is also very chaotic. This is Curry-style typing which can type pure/untyped terms. This is Girard's "existentialism".

The theory of types/formulas lies on a choosen and subjective definition of orthogonality defining what is a "good" interaction/computation. Several choices can be made. For instance:

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

Starting from a definition of orthogonality, we can reconstruct MLL types/formulas:

Pre-type. A pre-type is a set of constellations. They corresponds to classes of computational behaviours/objects.

Orthogonal. If we have a set of constellation $\mathbf{A}$, 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.

Type. A *type* (or formula) $\mathbf{A}$ is the orthogonal of another pre-type $\mathbf{B}$ i.e $\mathbf{A} = \mathbf{B}^\perp$. It means that it interacts well (with respects to the orthogonality) with another pre-type. It is equivalent to say that $\mathbf{A} = \mathbf{A}^{\perp\perp}$ meaning that it is closed by interaction.

Atoms. We define atoms with a *basis of interpretation* Φ associating of 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 type of constants "fu (katakana)" [23] which is auto-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 can't 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 $A \multimap B = A^\perp \parr B$.

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

Example 10. *Let $\mathbf{A} = \{[+a.x]\}$ be a pre-type of one constellation. We consider that $\Phi \perp \Phi'$ whenever $\Phi \uplus \Phi'$ is strongly normalising.*

- $\mathtt{Ex}([+a.x] + [+b.x]) = \emptyset$ *therefore* $[+a.x] \perp [+b.x]$ *and* $[+b.x] \in \mathbf{A}^\perp$.
- $\mathtt{Ex}([+a.x] + [a.x, x]) = [x]$ *therefore* $[+a.x] \perp [-a.x, x]$ *and* $[-a.x, x] \in \mathbf{A}^\perp$.
- $\mathtt{Ex}([+a.x] + [-a.x, +a.x])$ *isn't strongly normalising therefore* $[-a.x, +a.x] \notin \mathbf{A}^\perp$.

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: $A^\star \perp w^\star$ when $[accept] \in \mathtt{Ex}(A^\star + w^\star)$. An automaton becomes orthogonal to all the words it accepts and a word is orthogonal to all the automata which recognise it.*

Example 12 (MLL+MIX correctness)**.** *When taking the translation $\Phi \in \mathbf{A}$ of the vehicle of a proof-net of conclusion A, it is strongly normalising when interacting with the format $\mathcal{F}$ corresponding to the set of constellations containing the ordeals for A. The set of all constellations with which it strongly normalises is $\mathbf{A}^\perp$. Therefore, we have $\mathcal{F} \subseteq \mathbf{A}^\perp$. It is a materialisation of the fact that the ordeals for $\mathbf{A}$ represent partial proofs of $\mathbf{A}^\perp$. Conversely, all the constellations representing proofs certified by the MLL+MIX correctness criterion [11] are contained in $\mathcal{F}^\perp$.*

Example 13 (queries and answers in logic programming)**.** *We can sketch 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. We consider the strong normalisation of the union of two constellations as orthogonality. Let $\mathbf{QAdd} = \{[-add(s^n(0), s^m(0), r), r] \mid n, m \in \mathbf{N}\}$ *and* $\mathbf{AAdd} = \{[s^n(0)] \mid n \in \mathbf{N}\}$*. 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* $\Phi \in \mathbf{QAdd}$*,* $\mathtt{Ex}(\Phi \uplus \Phi^+_{\mathbf{N}}) \in \mathbf{AAdd}$ *and* $\Phi^+_{\mathbf{N}} \uplus \Phi$*. If* $\mathbf{QAdd}$ *and* $\mathbf{AAdd}$ *are proven to be types (we need a more specific orthogonality) then* $\Phi^+_{\mathbf{N}} \in \mathbf{QAdd} \multimap \mathbf{AAdd}$*. Although not explored here, it might be possible to retrieve existing type systems.*

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

7 Full linear logic and beyond

In this paper, we used only a small part of the power of stellar resolution. Multiplicative linear logic only needs constants instead of addresses but the stellar resolution offers a large variety of addresses and a mechanism of matching allowing us to imagine complex ways of connecting things. We may also use encoding of logic programs or imagine proof search with proof-nets.

We can feel that the stellar resolution have a built-in mechanism of erasure and duplication so the possibiliy of interpreting exponentials doesn't seem surprising. In order to represent the exponentials (the computational part was already studied independently by Duchesne [10] and Bagnol [6]), addresses have to consider an auxiliary term: $p_A(t)$ becomes $p_A(t \cdot u)$ where u represents the identifier of the copy of A. A proof using contraction will duplicate $+c.p_A(t)$ into $+c.p_A(t \cdot (\mathtt{l} \cdot y))$ and $+c.p_A(t \cdot (\mathtt{r} \cdot y))$. These copies can be erased by a cut with the void (it remains to exclude such incomplete diagrams from the execution). The dereliction stops the possibility of duplication by transforming $+c.p_A(t \cdot (u))$ into $+c.p_A(t \cdot (u \cdot \mathtt{d}))$. We need tests in order to check correctness in this case. This can be done in a limited fragment (intuitionistic MELL) by a checking of variables and a notion of coherence forbidding some substitutions to happen.

Other extensions with additive connectives, neutrals, second and first-order logic can be considered but need more technical developements and a slighly more complex framework by allowing internal colours within rays. This additional feature places the model beyond both tiling-based computation and logic programming.

The Transcendental Syntax finally shows us that logic may be about simple things such as locations, connexions, interaction, dynamics, testing, matching and errors: quite concrete and tangible things, reminiscent of the biological and physical world.

References

[1] Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

[2] 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.

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

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

[5] 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.

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

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

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

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

[10] Etienne Duchesne. *La localisation en logique: géométrie de l'interaction et sémantique dénotationnelle*. PhD thesis, Aix-Marseille 2, 2009.

[11] Arnaud Fleury and Christian Retoré. The mix rule. *Mathematical Structures in Computer Science*, 4(2):273–285, 1994.

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

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

[14] 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.

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

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

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

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

[19] Jean-Yves Girard. Geometry of interaction VI: a blueprint for transcendental syntax. *preprint*, 2013.

[20] Jean-Yves Girard. Transcendental syntax II: non-deterministic case. 2016.

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

[22] Jean-Yves Girard. Transcendental syntax I: deterministic case. *Mathematical Structures in Computer Science*, 27(5):827–849, 2017.

[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] 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.

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

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

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

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

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

[32] John Alan Robinson et al. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

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

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

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

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

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