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

Boris Eng

Stellar Resolution: Multiplicatives

For the linear logician, through examples

Boris Eng

The stellar resolution is an asynchronous model of computation used in Girard's Transcendental Syntax [15, 18, 16, 17, 20] which is based on Robinson's first-order clausal resolution [25]. By using methods of realisability for linear logic, we obtain a new model of multiplicative linear logic (MLL) based on sort of logic programs called *constellations* which are used to represent proofs, cut-elimination, formulas/types, correctness and provability very naturally. A philosophical justification of these works coming from the Kantian inspirations of Girard would be to study *the conditions of possibility of logic*, that is the conditions from which logical constructions emerge.

Contents

1 Stellar Resolution

1.1 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 [11, 10, 9, 12, 13, 14] 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 scratch with a computational model as it is done in classical realisability for instance. One may see it as a kind of reverse engineering of linear logic.

- *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 [21, 27]. 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 [12], Girard introduced a simplification based on first-order unification: the model of flows [4] 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.

- *Isn't it identical to first-order resolution or logic programming ?* At first, our model is 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). 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 is able to simulate models of computation which are also dynamical systems: abstract tile assembly models [24] which are used in DNA computing [29] but also the computational model of Wang tiles [28]. From our realisability construction, one can imagine methods of typing and implicit complexity analysis of various models. 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 [26]. The reconstruction of types/formulas follow the constructions of the model of realisability for linear logic (as in Ludics). Both Ludics and the Transcendental Syntax takes into account the idea of location of formulas within a proof so they are very close: the former is an abstraction of sequent proofs and the later of proof-nets.

- *Why is it interesting?* The stellar resolution generalises flows and interaction graphs which have an applications in implicit computational complexity [5, 2, 3]. 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 (not as mere predicate), a finite treatment of quantifiers, and a logical and computational status for first-order individuals (encoded as multiplicative propositions of a particular shape). Such an interpretation of predicate logic may also have applications in descriptive complexity. The case of MLL alone adds nothing more than already existing models of MLL but we have the hope that it can lead to more interesting things in future works.

1.2 Stars and constellations

We define *rays* by the following grammar:

$$r ::= +a(t_1, ..., t_n) \mid -a(t_1, ..., t_n) \mid t$$

where $+/-$ are polarities, a is a function symbol called a *colours* and $t_1, ..., t_n$ are first-order terms.

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

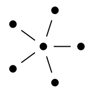
Example 1. *We encode a logic program. We write s^n for n applications of the symbol s (for instance $s^3(0) = s(s(s(0)))$). A colour is a predicate and the polarity represents the distinction input/output or hypothesis/conclusion. The absence of polarity means that the predicate is isolated and cannot be connected. We take two logic programs and their associated constellation to illustrate the model:*

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

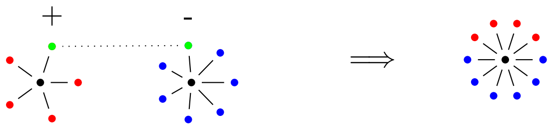
$$\Phi_{\mathbf{N}}^{n,m} = [+add(0, y, y)] + [+add(s(x), y, s(z)), -add(x, y, z)] + [-add(s^n(0), s^m(0), r), r]$$

```
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).
```

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

More graphically, a star may be depicted as an actual star:  for $[t_1, ..., t_5]$.

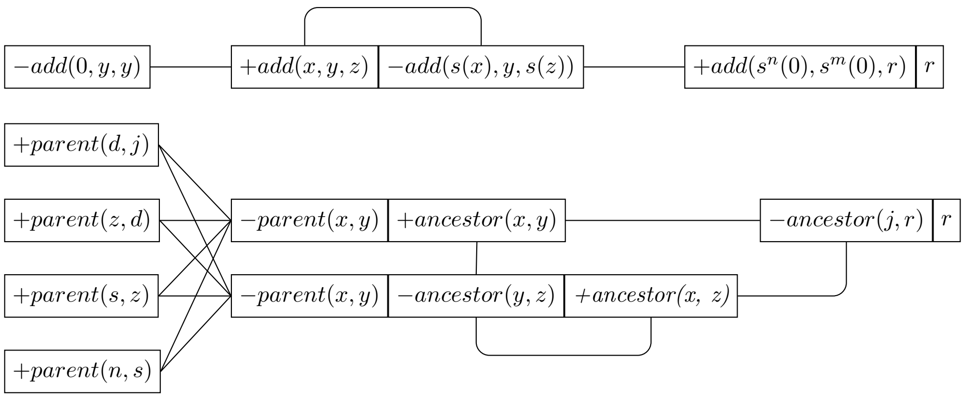
1.3 Evaluation of constellations



Graphically, we evaluate a constellation by connecting rays together when they are matchable and of opposite polarity. The two stars will fuse and the connected rays will disappear. The rays of the remaining star is affected by a "reaction" of this fusion. Think of a chemical or nuclear reaction.

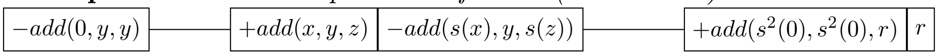
To do so, from a constellation Φ, we can construct its *dependency graph* $\mathfrak{D}[\Phi; \mathcal{A}]$ for a set of colours $\mathcal{A}$ telling us which rays can be connected together. It is a graph with stars $\phi \in \Phi$ as vertices and with edges between two stars whenever there are two rays r_1, r_2 of opposite polarity such that their underlying terms (without colour) are matchable. The edge is then labelled with the equation $r_1 \stackrel{?}{=} r_2$.

Example 2. *We give a more friendly representation of the dependency graphs corresponding to the two constellations of example 1:*

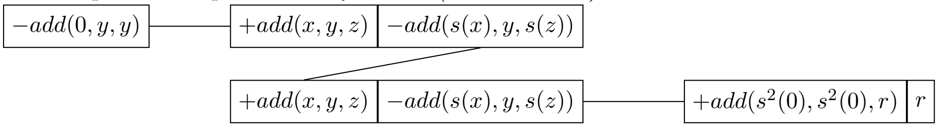


We can then make actual connexions between *occurrences* of stars following the dependency graph of a constellation. Such a connexion, called a *diagram* (written δ) have to be a tree (this condition can be relaxed in some cases) where all star variables are made distinct. They represent the possible (partial) executions of a program. Formally, it is defined by a graph homomorphism from a tree to a dependency graph.

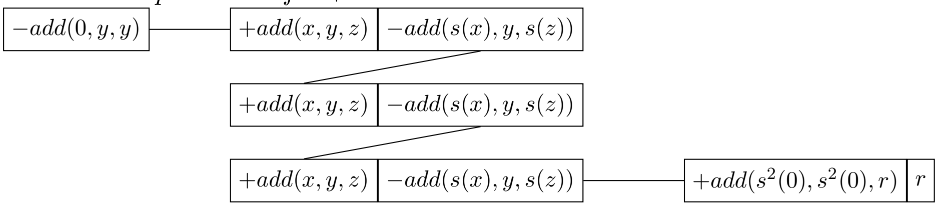
Example 3. *Partial computation of* $2+2$ *(0 recursion):*



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



Over computation of $2+2$*:*



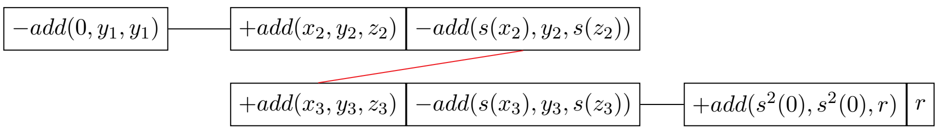
Note that in the case of $\Phi_{\mathbf{N}}^{n,m}$*, there is infinitely many diagrams.*

There are two equivalent ways of reducing diagrams by observing that each edge define a unification equation:

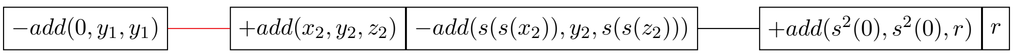
Fusion We can reduce the links step by step by solving the underlying equation, producing a solution θ. The two linked stars will fuse by making the connected rays disappear. The substitution θ is finally applied on the rays of the resulting star. This is exactly the resolution rule.

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).

Example 4 (fusion)**.** *The full fusion of the diagram representing a complete computation of* $2+2$ *from example 3 is described below (we make the exclusion of variable explicit for illustration):*



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



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

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

$$\boxed{s(s(s(s(0))))}$$

Example 5 (actualisation)**.** *If we take the diagram δ representing a complete computation in the example 3, 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 [23] 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.*

The *normalisation* or *execution* $\texttt{Ex}(\Phi)$ (figure 1) of a constellation Φ constructs the set of all possible correct (the underlying unification problem doesn't fail) saturated (no stars can be added to extend it) 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 [22] 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 6. *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 3 succeed and all other saturated diagrams represents partial or over computations and fail.*

$$\Phi \xrightarrow{\text{set of diagrams}} \cup_{k=0}^{\infty} D_k \xrightarrow{\text{restriction}} D'_1, ..., D'_n \subseteq \cup_{k=0}^{\infty} D_k \xrightarrow{\text{actualisation}} \phi_1 + ... + \phi_n$$

Figure 1: Illustration of the execution of a strongly normalising constellation where the restriction only keep the correct and saturated diagrams. The resulting constellation is $\texttt{Ex}(\Phi)$.

2 Encoding of some models of computation

It seems that our model is able to encode naturally the models of computation which are also dynamical systems but also dynamical/reducible hypergraphs (proof-nets, boolean circuits, Seiller's interaction graphs etc).

2.1 Wang tiles

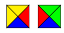
The idea is to encode a Wang tile [28] as a star of 4 rays. Two sides of matchable colours will be represented as two matchable rays. We still have to be careful with our definitions:

- the connexions of our stars are too free and may not follow the topological constraints of tiling in $\mathbf{N}^2$. We have to encode coordinates in $\mathbf{N}^2$.
- the colours of rays do not match the colours of the Wang tiles. In our model, we use the polarity together with a colour to represent a direction in the axis of $\mathbf{N}^2$: $+v, +h, -v, -h$ where v stands for "vertical axis" and h for "horizontal axis".

Let $t^i = (c^i_w, c^i_e, c^i_s, c^i_n)$, $i = 1, \ldots, k$, be a finite set of Wang tile. We encode each Wang tile in $\mathbf{N}^2$ by the star:

$$(t^i)^\bigstar = [-h(c^i_w(x), x, y), -v(c^i_s(y), x, y), +h(c^i_e(s(x)), s(x), y), +v(c^i_n(s(y)), x, s(y))]$$

A constellation will be a set of tiles and its corresponding dependency graph is simply describe the possible connexions. All finite tilings will correspond to correct saturated diagrams. Note that we have to consider a less strict definition of diagram: we allow them to be general graphs (especially, grid-like).

For more generality, we can think of an encoding of coordinates in $\mathbf{Z}^2$ or consider a bijection from $\mathbf{N}$ to $\mathbf{Z}^2$. For instance, the set $W = \{$  $\}$ will be translated as:

$$W^\bigstar = [-h(y(x), x, y), -v(b(y), x, y), +h(r(s(x)), s(x), y), +v(y(s(y)), x, s(y))]+$$

$$[-h(r(x), x, y), -v(b(y), x, y), +h(g(s(x)), s(x), y), +v(g(s(y)), x, s(y))]$$

We can see that the two red sides can be connected together because the corresponding rays have an opposite polarity but the underlying terms are matchable.

More interestingly, this definition can be extended with the more general abstract tile assembly model [29] with any temperature.

2.2 Boolean circuits

The idea is to first encode a graph representing the structure of a boolean circuit then to connect the translation with a constellation containing the implementation of the computational content of the nodes.

$$VAR(y, id) := [-val(x), y(x), +c(x, id)]$$

$$SHARE(id1, id2, id3) := [-c(x, id1), -c(x, id2), +c(x, id3)]$$

$$AND(id1, id2, id3) := [-c(x, id1), -c(y, id2), -and(x, y, r), +c(r, id3)]$$

$$OR(id1, id2, id3) := [-c(x, id1), -c(y, id2), -or(x, y, r), +c(r, id3)]$$

$$NEG(id1, id2) := [-c(x, id1), -neg(x, r), +c(r, id2)]$$

$$C(id) := [-c(x, id), r(x)] \qquad CONST(k, id) := [+c(k, id)]$$

$$QUERY(k, id) := [+c(k, id), r(k)]$$

where $id, id1, id2, id3$ 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 definition 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 of arithmetic circuits working with a particular field F. Syntax and semantics live in the same world and can interact! We can also internalise the id within the colour. For instance, $c(x, id)$ becomes $c_{id}(x)$.

We use the star C to represent the conclusion, the star $CONST(k)$ to force a variable to have a particular value and the star $QUERY(k)$ (instead of a conclusion star C) to ask for a particular output. For instance $QUERY(1)$ ask for satisfiability.

For instance, here is a circuit satisfying $x \vee \neg x$ where $\overline{n} = s^k(0)$:

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

The constellation $\Phi_{em} + \Phi^{\mathcal{PL}}$ will normalise by taking as input 0 or 1 for x. We finally have $\mathtt{Ex}(\Phi_{em} + \Phi^{\mathcal{PL}}) = [x(0)] + [x(1)]$ telling us that for the two valuations $x \mapsto 0$ and $x \mapsto 1$, the circuit produces the output 1.

2.3 Non-deterministic automata

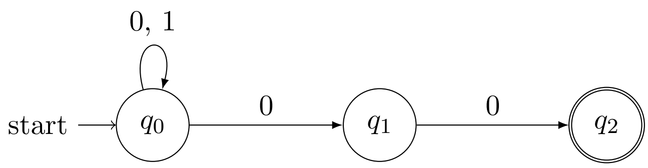
The idea is to represent the transitions by binary bipolar stars.

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 $[-a(\varepsilon, q_f), \mathit{accept}]$.
- 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:

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

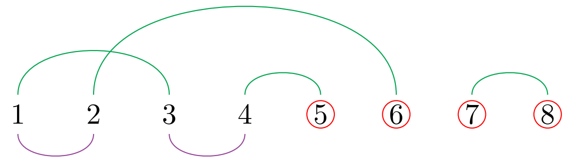
With the word $[+i(0 \cdot 0 \cdot 0)]$, there are two possible diagrams for the two possible transitions. Both will finally be unified without problems to the right transitions corresponding to the automaton and their actualisation will be *accept*. Therefore $\mathtt{Ex}(A^\bigstar + [+i(0 \cdot 0 \cdot 0)]) = [\mathit{accept}, \mathit{accept}]$.

3 The Geometry of Interaction

3.1 Cut-elimination and permutations

The geometry of interaction study generalisations of proof-nets by keeping only their essential parts. 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 $⅋/\otimes$ cuts can be seen as administrative/inessential cuts since all they do is basically a rewiring on the premises of the $⅋$ and $\otimes$ nodes connected together. Therefore, such a cut ultimately reduces as a permutation/edge between two loci as well. We observe that the loci are the support of the proof, the locations where a kind of interaction can take place.



Seiller [26] studied the geometry of interaction through connexions of graphs where the cut-elimination becomes the computation of all 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. In the case of cut-elimination, we only need edges between addresses/loci. The above graph becomes:

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

If we compute the corresponding dependency graph $\mathfrak{D}[\Phi; -]$, it will induce exactly the same graph as above. We represent edges between two locations as the matchability of two rays. The execution $\mathtt{Ex}(\Phi)$ will produce $[+c(5), +c(6)] + [+c(7), +c(8)]$ which corresponds to the expected normal form (set of maximal paths).

3.2 Correctness and partitions

If we want to handle correctness in a satisfactory and natural way, we have to shift to partitions instead of permutations [8, 1]. 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 permutations: $[-c(1), -c(2)] + [-c(3)] + [-c(4)]$ (the polarities have to be different from the constellation representing the axioms). However, in this case, all diagrams are closed (no free rays). We have to specify where the conclusions are located: $[-c(1), -c(2), A \otimes B] + [-c(3)] +$ $[-c(4), A^\perp \parr B^\perp]$. The Danos-Regnier criterion becomes "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".

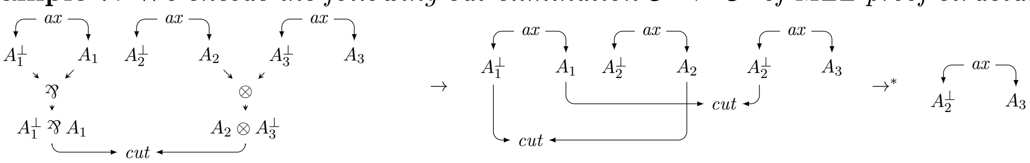
4 Interpretation of the computational content of MLL

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

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

1. A locus A becomes a ray $+c.p_A(t)$ where t represents the "address" of A relatively to the conclusions of the proof-structure (without considering cuts).
2. An axiom becomes a binary star containing the address of its formulas. It is coloured with $+c$ for "positive computation".
3. A cut between A and $A^\perp$ becomes a binary star $[-c.p_A(x), -c.p_B(x)]$ it is coloured with $-c$ for "negative computation" in order to connect them with axioms.

Example 7. *We encode the following cut-elimination $\mathcal{S} \to^* \mathcal{S}'$ of MLL proof-structures:*

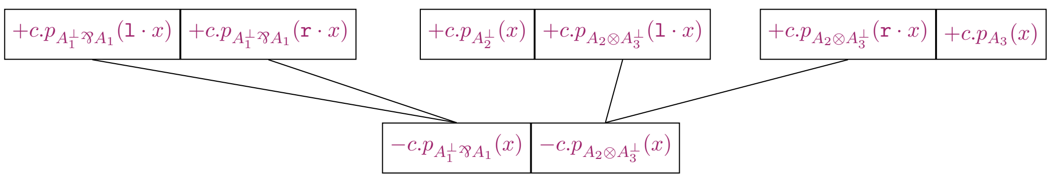


The address of $A_1^\perp$ is $p_{A_1^\perp ⅋ A_1}(\mathtt{l} \cdot x)$ because it is located on the left-hand side of $A_1^\perp ⅋ A_1$. The address of $A_3^\perp$ is $p_{A_2 \otimes A_3^\perp}(\mathtt{r} \cdot x)$ and the one for A_3 is $p_{A_3}(x)$. The proof-structure $\mathcal{S}$ is encoded as:

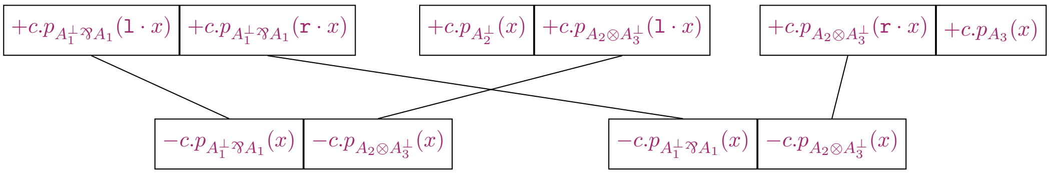
$$[+c.p_{A_1^\perp ⅋ A_1}(\mathtt{l} \cdot x), +c.p_{A_1^\perp ⅋ A_1}(\mathtt{r} \cdot x)] + [+c.p_{A_2^\perp}(x), +c.p_{A_2 \otimes A_3^\perp}(\mathtt{l} \cdot x)] +$$

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

and its dependency graph is:



Since a ray can only be connected to a unique other ray, one occurrence of cut isn't sufficient in order to make a saturated diagram. We have to duplicate the cut star, which corresponds exactly to the second step of $\mathcal{S} \to^* \mathcal{S}'$. One can check that the only correct diagram is the following one (the left matches with the left and the right with the right):

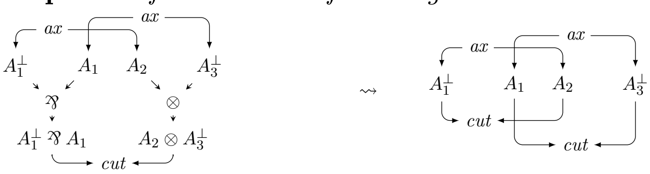


The matching is perfect so the connected rays are removed and we end up with

$$[+c.p_{A_2^\perp}(x), +c.p_{A_3}(x)]$$

corresponding to $\mathcal{S}'$. We can remark that in this case, the normalisation computes the set of all maximal paths. This corresponds to the interpretation of the cut-elimination in the GoI.

Example 8. *If we have the following reduction $\mathcal{S} \to \mathcal{S}'$:*

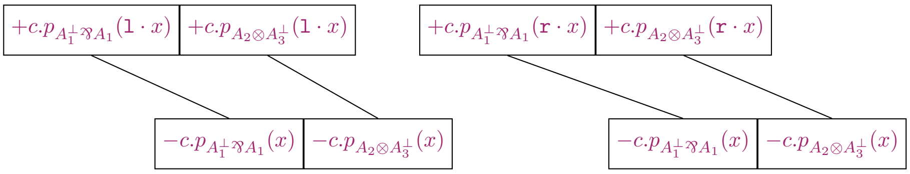


The constellation corresponding to $\mathcal{S}$ is

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

$$[-c.p_{A_1^\perp ⅋ A_1}(x), -c.p_{A_2\otimes A_3^\perp}(x)]$$

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



which contains two cycles and it is indeed the translation of $\mathcal{S}'$. These cycles can be unfolded as many times as we want by reusing some stars. It is impossible to leave a free ray since all diagrams can always be extended by adding further occurrences of stars. Therefore, all diagrams are closed and they can't define a correct diagram (the star of free is undefined because the empty star doesn't exist in our model). We finally have $\mathtt{Ex}(\mathcal{S}) = \emptyset$.

5 Interpretation of the logical content of MLL

5.1 The correctness criterion of Danos-Regnier

We translate the correctness criterion of Danos-Regnier [6] into the stellar resolution to show that it can be described very naturally by the unification of terms.

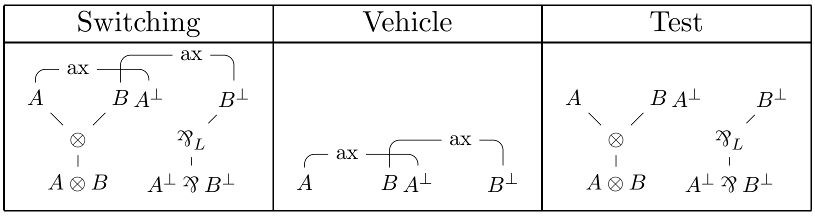
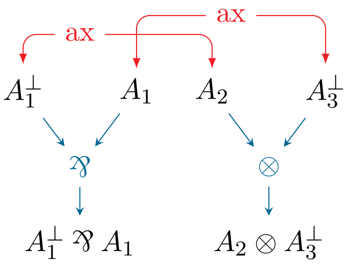


Figure 2: The vehicle and test corresponding to a Danos-Regnier switching graph.



The upper part made of axioms appearing in a proof-structure is called the *vehicle* and the lower-part is the *gabarit* (other name for *format*). With the Danos-Regnier switchings (called *ordeals*), only the lower-part (basically a syntax tree) is changed so a proof-structure may be divided into two parts: the vehicle is the *tested* and gabarit is a *set of test* against the vehicle. The vehicle holds the computational part of a proof and the test its logical part (type/formula). This test vehicle/gabarit produces a certification: if all tests pass, we have a *proof-net*. This is Girard's "usine" (factory). Note that testing is symmetric: a gabarit is also tested by a vehicle.

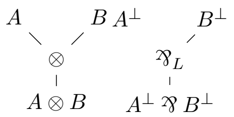
It is reminescent of testing in programming. A program is tested by another program and we expect the result of the testing to satisfy some property. It is exaclty the same here with constellations instead of programs.

Ordeals are translated in a very natural way by translating their nodes into constellations:

- $A^\bigstar = [-t.\texttt{addr}_S(C_e^d), +c.q_{C_e^d}(x)]$ where A is a conclusion of axiom,
- $(A ⅋_L B)^\bigstar = [-c.q_A(x)] + [-c.q_B(x), +c.q_{A⅋B}(x)]$,
- $(A ⅋_R B)^\bigstar = [-c.q_A(x), +c.q_{A⅋B}(x)] + [-c.q_B(x)]$,
- $(A \otimes B)^\bigstar = [-c.q_A(x), -c.q_B(x), +c.q_{A\otimes B}(x)]$,
- We add $[-c.q_A(x), p_A(x)]$ for each conclusion A

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 $[p_{A_1}(x), ..., +c.p_{A_n}(x)]$.

Example 9. *Here is an example with the switching graph and the test of figure 2:*

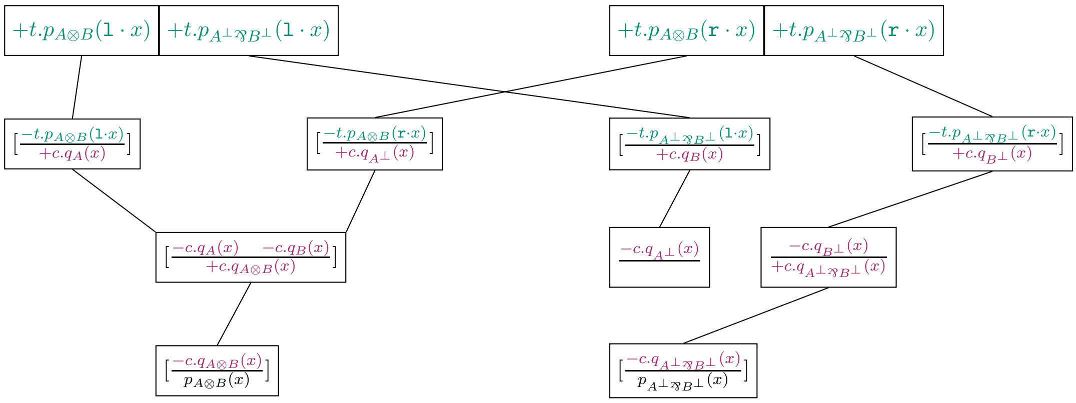


$$[\frac{-t.p_{A\otimes B}(\mathtt{l}\cdot x)}{+c.q_A(x)}] + [\frac{-t.p_{A^\perp \parr B^\perp}(\mathtt{l}\cdot x)}{+c.q_B(x)}] + [\frac{-t.p_{A\otimes B}(\mathtt{r}\cdot x)}{+c.q_{A^\perp}(x)}] +$$
$$[\frac{-t.p_{A^\perp \parr B^\perp}(\mathtt{r}\cdot x)}{+c.q_{B^\perp}(x)}]+$$
$$[\frac{-c.q_A(x) \quad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

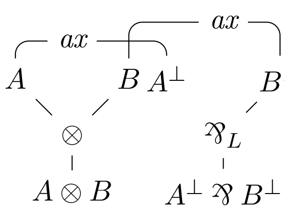
When connected to the vehicle

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

we obtain the following dependency graph:



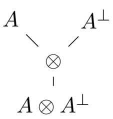
structurally corresponding to the following switching graph:



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

The essential point of the translation is that the corresponding dependency graph will have the same shape as a switching graph.

Example 10. *If we have the following test instead:*



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

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 perfect, we can construct infinitely many correct diagrams. The constellation isn't strongly normalisable.

5.2 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 proof $\vdash A$ as the constellation $\{[p_A(x)]\}$
- an n-ary axiom as the constellation $\{[p_{A_1}(x), ..., p_{A_n}(x)]\}$
- a standalone $A \otimes B$ link $[p_A(x)] + [p_B(x)]$
- a standalone $A \parr B$ link $[p_A(x), p_B(x)]$ (actually an axiom)

This is very similar to partitions, as presented by Acclavio and Maieli [1].

Note that our simulation of cut-elimination and correctness normalises into coloured constellations but it is possible to remove the colours in order to keep a canonical representation of proofs.

We can finally define what the full translation of a proof-structure is. A proof-structure is translated into a triple $\Phi_V \uplus \Phi_C \uplus \Phi_G$ where:

- Φ_V is the uncoloured vehicle made of translations of axioms
- Φ_C is the uncoloured translation of cuts
- Φ_G is coloured translation of all ordeals

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 type.

5.3 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".

We work with set of constellations corresponding to kind of "pre-types". The theory of types/formulas lies on a choosen and subjective definition of orthogonality. Which is a relation describing what constellations can be put against each other or can be tested against each other. 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)" [19] 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 11. *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.*

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

Example 12 (Automata)**.** *From the previous encoding of automata we can observe a duality between automata and words. It induces an orthogonality:* $A^\bigstar \perp w^\bigstar$ *when* $\texttt{Ex}(A^\bigstar + w^\bigstar) \neq \emptyset$. *An automaton becomes orthogonal to all the words it accepts and a word is orthogonal to all the automata which recognise it.*

Example 13 (MLL 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 gabarit* $\mathcal{G}$ *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* $\mathcal{G} \subseteq \mathbf{A}^\perp$*. 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 Danos-Regnier criterion are contained in* $\mathcal{G}^\perp$*.*

Example 14 (Logic programming)**.** *Let*

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

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* $\Phi \in \mathbf{QAdd}$, $\texttt{Ex}(\Phi \uplus \Phi^+_{\mathbf{N}}) \in \mathbf{AAdd}$ *and* $\Phi^+_{\mathbf{N}} \uplus \Phi$*. If* $\mathbf{QAdd}$ *and* $\mathbf{AAdd}$ *are proven to be types (we need a more specific orthogonality) then* $\Phi^+_{\mathbf{N}} \in \mathbf{QAdd} \multimap \mathbf{AAdd}$*.*

6 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 stellar resolution offer a large variety of addresses and a mechanism of matching allowing us to imagine exotic ways of connecting things. We may also use encoding of logic programs or imagine proof search with proof-nets.

In order to represent the exponentials (the computational part was already studied independently by Duchesne [7] and Bagnol [4]), 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 $+c.p_A(x)$ (weakening). The dereliction stops the possibility of duplication by transforming $+c.p_A(t \cdot (u \cdot y))$ into $+c.p_A(t \cdot (u \cdot \mathtt{d}))$.

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] 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.

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

[3] 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.

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

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

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

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

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

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

[10] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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