Non-commutative proof construction: a constraint-based approach

Jean-Marc Andreoli[a], Roberto Maieli[b*] and Paul Ruet[a*]
[a] CNRS - Institut de Mathématiques de Luminy
163 avenue de Luminy, Case 907, 13288 Marseille Cedex 9 (France)
[b] Università Roma Tre, Via Ostiense 234, 00144 Roma (Italy)
andreoli,ruet@iml.univ-mrs.fr,maieli@uniroma3.it

Abstract

This work presents a computational interpretation of the construction process for cyclic (CyLL) and non-commutative (NL) sequential proofs. We assume a proof construction paradigm, based on a normalisation procedure, known as *focussing* which manages efficiently the non-determinism of the construction.

Similarly to the commutative case, a new formulation of focussing for NL is used to introduce a general constraint-based technique in order to deal with partial information during proof construction. In particular, the procedure develops through construction steps propagating constraints in intermediate objects called *abstract proofs*.

1 Introduction

1.1 The proof construction paradigm

We are interested here in the computational paradigm of proof construction in logical sequent calculi. The most straightforward (and naive) proof construction algorithm starts with a single open node labelled by a given sequent (or even a single formula), then tries to incrementally construct a proof by repeatedly expanding each open node, selecting and applying an inference from the sequent calculus, thus possibly introducing new open nodes. This provides an interesting computational model, particularly adapted to capture non-deterministic processes, since proof construction itself is intrinsically non-deterministic: in the naive procedure, for example, many choices of different kinds (choice of the principal formula, choice of the inference) have to be made at each expansion step.

1.2 Focussing strategy and bipolar sequent calculus

However, it is well-known that, due to intrinsic permutability properties of the inferences in sequent calculi, some strategy is needed in order to avoid making sets of choices which lead to the same object (modulo permutation of inferences). Such a strategy, called focussing, has been proposed in [2]. It is based on the generic concept of polarity of formulas, and therefore applies to any logical system where connectives have polarities, such as linear logic or non-commutative logic (see Section 2 for an introduction to non-commutative logic). Focussing deals with two important forms of irrelevant non-determinism in proof construction: on one hand, the *instant* of the decomposition of a *negative* connective is simply irrelevant; on the other hand, the *interval* between two decompositions of *positive* connectives, if one is an immediate successor of the other in a formula, is irrelevant. The strategy to avoid these forms of irrelevant non determinism can be expressed in the sequent calculi themselves by modifying the syntax of the sequents, with the introduction of a distinguished formula called the "focus". Such focussing sequent calculi have thus been proposed in various contexts [17]. There is also an alternative presentation which does not rely on syntactic conventions while capturing exactly the same content: keep sequents as simple as possible (e.g., they are made of atoms only), but refine the inferences of the calculus themselves. Such a refined, simplified calculus, called the (focussing) *bipolar* calculus, has been presented in [3] for linear logic. It is strictly isomorphic to the focussing sequent calculus of linear logic, so proof construction can be performed equivalently in both systems. We extend here the focussing bipolar calculus to non-commutative logic, without problem given the genericity of the approach.

1.3 Dealing with partial information

Now, the naive proof construction procedure can be directly applied to the focussing bipolar calculus, but this is still unsatisfactory. Indeed, although focussing eliminates a lot of irrelevant non-determinism, there still remains sources of non-determinism which are intractable in the naive approach. The most well-known one appears in the first-order case with the

*Research supported by Vinci programme "Logique mathématique et informatique théorique" of Université Franco-Italienne.

existential quantification rule [18]: a choice of a term to assign to the quantified variable has to be made, but there are infinitely many possibilities. A random enumeration is clearly unsuitable, essentially because two different random choices may lead to essentially the same object, differing only by a term which, anyway, is irrelevant to the proof. For example, here are two proofs of the formula $\forall x\ p(x) \multimap \exists x\ p(x)$ which differ by a purely irrelevant choice (of the term to assign to x):

$$\dfrac{\dfrac{\dfrac{}{\vdash p(a), p(a)^\perp}}{\vdash \exists x\ p(x), p(a)^\perp}\ \exists}{\vdash \exists x\ p(x), \exists x\ p(x)^\perp}\ \exists \qquad \dfrac{\dfrac{\dfrac{}{\vdash p(b), p(b)^\perp}}{\vdash \exists x\ p(x), p(b)^\perp}\ \exists}{\vdash \exists x\ p(x), \exists x\ p(x)^\perp}\ \exists$$

Again, we need a strategy to avoid such irrelevant non-determinism. In the case of the existential quantification, the solution is quite familiar: it is based on unification, which performs the choice of the term for an existentially quantified variable in a lazy way, making actual choices only when they are needed somewhere in the proof. This leads the construction procedure to manipulate not ground proofs, but abstract proofs containing only partial information about the object being constructed (partiality derives from the presence of uninstantiated variables occurring in the terms). This also means that, at each step, some constraints on the different variables have to be propagated across the proof, when variables are shared between multiple nodes: this is the role of the unification process. We thus obtain a slightly less naive construction process which intertwines the usual naive expansion process and the "dual" unification process which works in the reverse direction.
There is another case of irrelevant non determinism which the naive procedure cannot avoid, namely caused by the positive multiplicative connectives: to decompose such a connective, a choice of splitting of the context has to be performed. And again, as before, a random enumeration of all the possible splits is unfeasible. A solution based on lazy splitting [9], and capable of dealing with partially defined (or even, possibly, completely undefined) sequents, has been proposed, for the focussing bipolar calculus of linear logic in [3]. It couples the naive construction procedure with a non-deterministic but finite constraint propagation procedure, thus leading to a constraint based construction mechanism.

1.4 Non-commutative logic

We study here the extension of this mechanism to non-commutative logic, NL. It is shown that it extends straightforwardly to cyclic logic, but not directly to non-commutative logic, although NL is an extension of both linear and cyclic logic. We then propose a new solution, which works on an affine version of NL where *weakening* is allowed for *negative* formulas. This is a natural modification of NL, less harmful indeed than affine logic obtained by adding unrestricted weakening, and very similar to the kind of relaxation of the syntax considered in the context of polarised linear logic [8, 13]. We show that our solution is adapted to multiplicative-additive NL (the exponential connectives are not considered in the present paper), by giving sound and complete constraint resolution algorithms in Section 4.2.
Interestingly, this work also shows that the mechanism that operates in the constraint resolution relies in no way on the specificity of the structures of order and order varieties. The characteristic properties of these structures only imposes additional constraint propagation rules (named **OV*** in Section 4.2), enforcing that the generated information always constitutes an instance of the structure. Other structures have been considered in [11] and [4]. We strongly believe that the method presented here applies directly to these cases: only the additional rules characterising the structure have to be changed. For example, in the basic case of ternary cyclic relations, no additional rule at all is needed.
On the other hand, it might well be that in the specific case of order varieties, a simpler procedure exists, in the style of that used for the commutative and cyclic cases.

1.5 Outline of the paper

Section 2 introduces the necessary preliminaries on NL, together with the focussing bipolar sequent calculus for multiplicative-additive NL. Section 3 presents the abstract inferences derived from this calculus and their associated constraints. Section 4 recalls the constraint resolution algorithm for commutative LL, presents its adaptation to cyclic LL, and presents a new algorithm for NL, together with examples.
The overall goal of the line of research pursued by this paper is two-fold: on the one hand, the definition of a proof construction mechanism capable of dealing with partial information, and on the other hand the optimisation of this mechanism by elimination of irrelevant non-determinism. Both aspects are important in modelling real situations, esp. in the context of widely distributed applications, such as Internet applications, where there is no central *locus* with a vision of the whole execution (hence the importance of dealing with partial information), and programs are strongly influenced by their environment, which, at any reasonable level of abstraction, behaves in a highly non-deterministic way. The proof construction paradigm provides an appropriate level of abstraction to understand coordination issues in such applications, as shown by the CLF system [5], a component-based middleware infrastructure built around the central notion of resource (linear logic) and coordination viewed as resource manipulation (proof construction). However, CLF exploits only a very limited fragment of linear logic. The constraint based proof construction mechanism, developed here in the non-commutative case, offers perspectives for further extensions of the CLF platform in particular, and for a better understanding of coordination issues in distributed applications in general.

2 Non-commutative logic

Non-commutative logic, NL for short, was introduced by Abrusci and the third author in [1, 21]. It generalises Girard's commutative linear logic and Yetter's cyclic linear logic [22], a classical conservative extension of the Lambek calculus [12]. In NL, the usual *tensor* and *par* exist in two versions each, one commutative and one non-commutative. This induces a structure on sequents, which become *order varieties* of formula occurrences instead of simple sets. An order variety is essentially a partially ordered set *up to cyclic permutations*, the partial order imposing a constraint on possible commutations (between the flat orders, corresponding to the commutative case, and the linear orders which represent the purely non-commutative cyclic case). This is recalled in Sections 2.1 and 2.2.
A particularly noticeable relation between order varieties is the relation of *entropy*, which irreversibly weakens the order; it corresponds to inclusion of order varieties and implies that the commutative tensor is stronger than the non-commutative one; Section 2.3 recalls some general properties of entropy. A particularly important one, used in the constraint resolution algorithm presented here, is recalled in Section 2.4. The focussing sequent calculus for non-commutative logic is recalled in Section 2.5, and its bipolar version in Section 2.6. They are shown equivalent in Section 2.7 but our constraint based proof construction procedure is defined on the latter.

2.1 Order varieties

An *order variety* on a given set D is a ternary relation α which is:

cyclic:	$\forall x, y, z \in D,$	$\alpha(x, y, z) \Rightarrow \alpha(y, z, x)$
anti-reflexive:	$\forall x, y \in D,$	$\neg\alpha(x, x, y)$
transitive:	$\forall x, y, z, t \in D,$	$\alpha(x, y, z)$ and $\alpha(z, t, x) \Rightarrow \alpha(y, z, t)$
spreading:	$\forall x, y, z, t \in D,$	$\alpha(x, y, z) \Rightarrow \alpha(t, y, z)$ or $\alpha(x, t, z)$ or $\alpha(x, y, t)$.

D is called the support set of α, denoted $|\alpha|$. For instance, any oriented cycle $(a_1 \to \cdots \to a_n \to a_1)$ induces a *total* order variety α on the set of its vertices by: $\alpha(x, y, z)$ if, and only if, y is between x and z in the cycle; this order variety is denoted by $a_1.a_2.\ldots.a_n = a_2.\ldots.a_n.a_1$, etc. The spreading condition enables to systematically give "presentations" of order varieties as orders in a reversible way, whence the name. The correspondence is as follows. Given an order variety α and $x \in |\alpha|$, we may define a partial order α_x on $|\alpha| \setminus \{x\}$ by:

$$\alpha_x(y, z) \quad \text{if, and only if,} \quad \alpha(x, y, z)$$

Conversely, given a partial order ω, let $\omega(x, y|z)$ denote the following ternary relation on $|\omega|$:

$$\begin{array}{lcl} \omega(x, y|z) & \text{if, and only if,} & (\omega(x, z) \Leftrightarrow \omega(y, z)) \\ & \text{and} & (\omega(z, x) \Leftrightarrow \omega(z, y)) \end{array}$$

expressing that z is in the same relation with x and y in ω; then we may define an order variety $\overline{\omega}$ on $|\omega|$, the *closure* of ω, by:

$$\overline{\omega}(x, y, z) \quad \text{if, and only if,} \quad \begin{array}{lcll} \omega(x, y) & \text{and} & \omega(x, y|z) & \text{or} \\ \omega(y, z) & \text{and} & \omega(y, z|x) & \text{or} \\ \omega(z, x) & \text{and} & \omega(z, x|y) & \end{array}$$

It is shown in [1] that $\overline{\omega}$ is indeed an order variety. When $\overline{\omega} = \alpha$, we say that ω *presents* α. Hence an order variety is a set of partial orders (the presentations) glued together in a convenient way.
Given two orders ω and τ, we may define the following orders on the disjoint union $|\omega| + |\tau|$ of their supports:

$$\begin{array}{ll} \text{series sum:} & \omega < \tau = \omega + \tau + |\omega| \times |\tau| \\ \text{parallel sum:} & \omega \parallel \tau = \omega + \tau. \end{array}$$

One proves easily that the closure identifies series and parallel sums:

$$\overline{\omega < \tau} = \overline{\omega \parallel \tau} = \overline{\omega > \tau}$$

The above order variety is denoted $\omega * \tau$ and called the *gluing* of ω and τ. It enjoys:

$$\omega * \tau = \overline{\omega} + \omega.|\tau| + |\omega|.\tau + \overline{\tau}$$

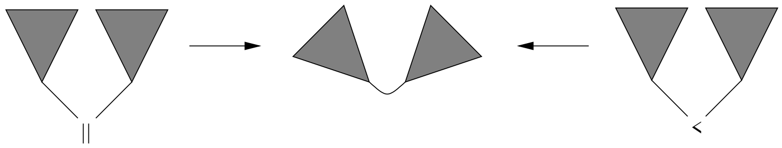
where, given an order ω and a set D disjoint from its support set, $D.\omega$ or $\omega.D$ denote the cyclic closure of $\omega \times D$. The two processes of fixing a point in an order variety and gluing orders are related by the following equations:

$$\alpha_x * x = \alpha \ \text{ and } \ (\omega * x)_x = \omega$$

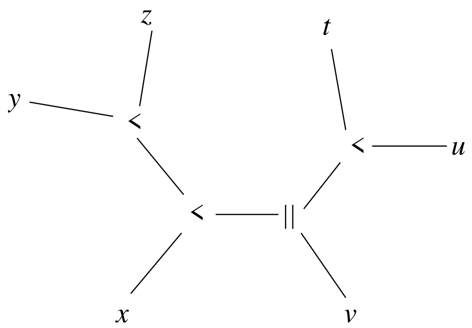
for α an order variety, $x \in |\alpha|$ and ω a order on $|\alpha| \setminus \{x\}$. These equations state that the species of order varieties in the sense of Joyal [10] has derivative the species of partial orders.

2.2 Series-parallel order varieties

Series-parallel orders are those obtained from the unique orders on singletons (the empty relation $\emptyset$ with support $\{x\}$) by series and parallel sums. For a more substantial survey, see [19].
Series-parallel order varieties are precisely those order varieties which can be presented by a series-parallel order. A series-parallel order variety α on a set D can be represented by a rootless planar tree (or *seaweed*, or *alg*) with leaves labelled by elements of D and ternary nodes labelled by $<, >$ or $\|$: take an arbitrary presentation of α as a series-parallel order ω, write ω as a — non-unique (associativity, commutativity) — planar binary tree t with leaves labelled by elements of D, and root and nodes labelled by $<$ or $\|$ for series and parallel sum respectively; then remove the root of t.



For instance $\overline{(x < y < z) \parallel v \parallel (t < u)}$ can be represented by:



To read the seaweed, take three leaves a, b, c and let $\circ$ be the node at the intersection of the three paths ab, bc and ca; then (a, b, c) is in the order variety if, and only if,

- the node $\circ$ is labelled by $<$ (resp. $>$) and
- the paths $a\circ$, $b\circ$ and $c\circ$ are in this cyclic order while moving clockwise (resp. counter-clockwise) around $\circ$.

Restrictions to a subset D are denoted $\omega\lceil_D$, $\alpha\lceil_D$; restriction preserves clearly the structures of order and order variety, and preserves series-parallelism. From now on, we consider only series-parallel orders and order varieties.

2.3 Entropy

Entropy $\trianglelefteq$ is the relation between series-parallel orders on the same given set defined by

$$\omega \trianglelefteq \tau \text{ if, and only if, } \omega \subseteq \tau \text{ and } \overline{\omega} \subseteq \overline{\tau}$$

Entropy is clearly a partial order, compatible with restriction and with the series and parallel sums of orders. In the series-parallel case, $\trianglelefteq$ is the least reflexive transitive relation between series-parallel orders on the same set such that:

$$\omega[\tau_1 \parallel \tau_2] \trianglelefteq \omega[\tau_1 < \tau_2]$$

Entropy between orders corresponds to inclusion of order varieties: given two order varieties α, β on D and $x \in D$, we have

$$\alpha \subseteq \beta \text{ if, and only if, } \alpha_x \trianglelefteq \beta_x$$

This is independent from the choice of x. Entropy is performed in the tree representation for series-parallel order varieties by changing some $<$-nodes into $\|$-nodes, i.e. by weakening the information on the nodes.

2.4 Splitting

Splitting is the following problem: given a family of pairwise disjoint sets $(D_i)_{i=1,\ldots,n}$, one of which at least being a singleton, and series-parallel order varieties α, β, respectively on $\{1, \ldots, n\}$ and on $\biguplus_i D_i$, find for each $i = 1, \ldots, n$ a series-parallel order (ω_i) on D_i such that $\beta \subseteq \alpha(\omega_1, \ldots, \omega_n)$. This problem has been solved (in a slightly different, but equivalent form) in [14] for the case of the binary splitting and then generalised in [7] to the case of the n-ary splitting:

Identity: $A^\perp$ is a positive atom.

$$\frac{}{A \blacktriangleright A^\perp}$$

Positive rules: ω, τ are orders on positive formulas and atoms.

$$\frac{\omega \blacktriangleright A \qquad \tau \blacktriangleright B}{(\tau < \omega) \blacktriangleright A \odot B}\odot \qquad \frac{\omega * A \qquad \tau * B}{(\tau \parallel \omega) \blacktriangleright A \otimes B}\otimes \qquad \frac{}{\blacktriangleright 1}1$$

$$\frac{\omega \blacktriangleright A}{\omega \blacktriangleright A \oplus B}\oplus \qquad \frac{\omega \blacktriangleright B}{\omega \blacktriangleright A \oplus B}\oplus \qquad \text{(no rule for 0)}$$

Negative rules

$$\frac{\omega * (A < B)}{\omega * A \triangledown B}\triangledown \qquad \frac{\omega * (A \parallel B)}{\omega * A \parr B}\parr \qquad \frac{\overline{\omega}}{\omega * \perp}\perp$$

$$\frac{\omega * A \qquad \omega * B}{\omega * A \,\&\, B}\& \qquad \frac{}{\omega * \top}\top$$

Focussing rule (with entropy): A positive and $|\omega|$ contains no compound negative formulas and $\alpha \subseteq \omega * A$.

$$\frac{\omega \blacktriangleright A}{\alpha}\blacktriangleright$$

Unfocussing rule: A is not positive.

$$\frac{\omega * A}{\omega \blacktriangleright A}\not\blacktriangleright$$

Table 1: Focussing sequent calculus for MANL with constants

Theorem 2.1 *The splitting problem for $(D_i)_{i=1,\ldots,n}, \alpha, \beta$ has a solution if and only if the following condition (known as admissibility) holds.*

$$\begin{array}{ll} \forall a \in D_i,\ b \in D_j,\ c \in D_k \ / \ i \neq j \neq k \neq i & \beta(a,b,c) \Rightarrow \alpha(i,j,k) \\ \forall a, b \in D_i \ c \in D_j \ c' \in D_k \ / \ j, k \neq i & \neg(\beta(a,b,c) \wedge \beta(b,a,c')) \end{array}$$

When this condition holds, the set of solutions is given by $\tau_i \trianglelefteq \omega_i$ for all i, where

$$\tau_i = \bigcup_{z \in D_j,\ j \neq i} \beta_z\restriction_{D_i}$$

Note that it is essential here that (at least) one of the D_i's be a singleton, otherwise the second part of the theorem, defining the minimal solution, is false, as shown by the following counter-example:

$$\begin{array}{l} D_1 = \{a_1, b_1, c_1\} \\ D_2 = \{a_2, b_2\} \end{array} \quad \text{and} \quad \begin{array}{l} \beta = (a_1 \parallel a_2) < (b_1 \parallel b_2) * c_1 \\ \alpha = 1 * 2 \end{array}$$

In that case, the admissibility condition is satisfied, but $\tau_1 = \{(b_1, c_1), (c_1, a_1)\}$ which is not an order.

2.5 Focussing sequent calculus for multiplicative additive NL with constants

Formulas of multiplicative additive non-commutative logic (MANL) are built from (negative) atoms $a, b, \ldots$ and their (positive) duals $a^\perp, b^\perp, \ldots$, and the following connectives:

	Positive		Negative	
Multiplicatives	1	*one*	$\perp$	*bot*
	$\otimes$	*times*	$\parr$	*par*
	$\odot$	*next*	$\triangledown$	*sequential*
Additives	0	*zero*	$\top$	*top*
	$\oplus$	*plus*	$\&$	*with*
	$\exists$	*exists*	$\forall$	*forall*

Duality is defined by usual De Morgan rules. For instance, $(A \odot B)^\perp = B^\perp \triangledown A^\perp$. A *sequent* is:

- either a finite series-parallel order variety α of occurrences of formulas,
- or a pair, denoted by $\omega \blacktriangleright F$, consisting of a finite series-parallel order ω of occurrences of positive formulas and atoms, and a single formula F called the *focus*.

The symbol $\blacktriangleright$ corresponds to the gluing of orders, so $\omega \blacktriangleright F$ stands for $\omega * F$, but the occurrence F has been syntactically distinguished. Since we can focus on any formula of an order variety, the symbol $\blacktriangleright$ marks a fixed (positive) focus and keeps track of its subformulas along the positive steps of the proof.
Observe that a sequent actually consists of two data: an order variety (possibly with a distinguished point) and a map from its support set to the set of formulas. In the sequent calculus, we omit, as is usual, the reference to the support set. Later, from Section 3.2 on, we shall need to explicitly manipulate the elements of the support set, which we shall call *places*.
The inferences of the focussing sequent calculus are presented in Table 1. The present calculus differs slightly from that given in [14] where the entropy rule is implicitly combined with the rules for the tensor connectives, and optimised so as to introduce only the minimal entropy needed by the tensor. We find it simpler here to allow explicit unconstrained entropy, since, anyway, the constraint-based approach that we take here will turn out to perform the same optimisation. However, since entropy can always be permuted so as to occur only immediately after instances of the focussing rule, we chose to combine the entropy with the focussing rule. This simplifies the shape of the constraints we manipulate.
In Section 4.2 we consider also an affine version of this sequent calculus, where a restricted form of weakening is allowed (when the formula introduced is negative).

2.6 The focussing bipolar sequent calculus

We define here a variant of the focussing sequent calculus which is better adapted to the construction process. Essentially, this *bipolar sequent calculus* is obtained by:

1. grouping together all the inferences in the topmost positive-negative layer of connectives in any principal formula,
2. naming positive subformulas at the border of such layers by fresh negative atoms.

The following definitions are trivial extensions to the non-commutative case of the corresponding definitions given for linear logic in [3].

Definition 2.2 (Monopole, bipole, flat sequent)

- *A* monopole *is a formula built from negative atoms using the negative connectives.*
- *A* bipole *is a formula built from the monopoles and the positive atoms, using the positive connectives. Furthermore, it is assumed that each bipole contains at least one positive atom (thus, bipoles and monopoles are disjoint).*
- *A* flat sequent *is an order variety over negative atoms only.*

Definition 2.3 (Focussing bipolar sequent calculus) *Given a set $\mathcal{F}$ of bipoles, the* focussing bipolar sequent calculus $\Sigma[\mathcal{F}]$ *is the set of inferences of the form*

$$\frac{\alpha_1 \qquad \cdots \qquad \alpha_n}{\alpha}\, F$$

where

- *F is a bipole of $\mathcal{F}$,*
- *$\alpha, \alpha_1, \ldots, \alpha_n$ are flat sequents,*
- *there exist an order ω over $|\alpha|$ satisfying $\alpha \subseteq \overline{\omega}$ and a proof of $\omega \blacktriangleright F$ in the focussing sequent system of non-commutative logic with proper axioms $\alpha_1, \ldots, \alpha_n$.*

The proof of $\omega \blacktriangleright F$ advocated in the above definition is necessarily maximal because its proper axioms contain only negative atoms, and hence cannot be further expanded: it is a "full decomposition" of F in the context ω.
Observe that by labelling inferences simply by bipoles, we loose the information about the choices made in the $\oplus$-inferences (between $\oplus_1$ and $\oplus_2$) in the proof of $\omega \blacktriangleright F$. This is done for sake of simplicity. We could have added another piece of information into the label to keep track of these choices. Actually, that is what we do for abstract proofs in Section 3.2 (see Definition 3.2).

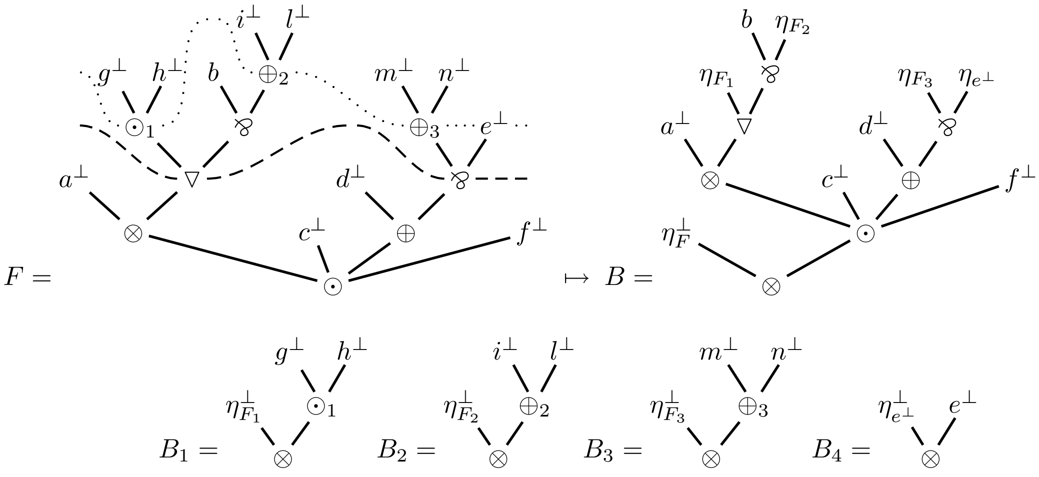


Figure 1: The multiple layers of a formula, and the resulting bipoles

Example 2.4 Consider the bipole

$$F = (p^\perp \odot q^\perp) \otimes ((\underbrace{a \parr (b \,\&\, (b' \triangledown c)))}_{F_1} \odot (\underbrace{d \triangledown e}_{F_2})) \otimes r^\perp$$

built from the monopoles F_1, F_2 and the positive atoms $p^\perp, q^\perp, r^\perp$. If $F \in \mathcal{F}$, then the inferences of $\Sigma[\mathcal{F}]$ labelled by F are exactly those of the form

$$\frac{\omega_1 * (a \parallel b) \qquad \omega_1 * (a \parallel (b' < c)) \qquad \omega_2 * (d < e)}{\alpha}\, F$$

where $\alpha \subseteq \overline{r \parallel (\omega_2 < \omega_1) \parallel (q < p)}$, and ω_1, ω_2 are orders over occurrences of negative atoms. Indeed, any full decomposition of F in the focussing sequent calculus is necessarily of the form:

$$\dfrac{\dfrac{\dfrac{}{p \blacktriangleright p^\perp} \qquad \dfrac{}{q \blacktriangleright q^\perp}}{q < p \blacktriangleright p^\perp \odot q^\perp}\odot \qquad \dfrac{\dfrac{\omega_1 * (a \parallel b) \qquad \omega_1 * (a \parallel (b' < c))}{\omega_1 \blacktriangleright a \parr (b \,\&\, (b' \triangledown c))}\triangledown\,\&\parr\cancel{\blacktriangleright} \qquad \dfrac{\omega_2 * (d < e)}{\omega_2 \blacktriangleright d \triangledown e}\triangledown\cancel{\blacktriangleright}}{\omega_2 < \omega_1 \blacktriangleright (a \parr (b \,\&\, (b' \triangledown c))) \odot (d \triangledown e)}\odot \qquad \dfrac{}{r \blacktriangleright r^\perp}}{r \parallel (\omega_2 < \omega_1) \parallel (q < p) \blacktriangleright F}\otimes\otimes$$

2.7 Equivalence with the focussing sequent calculus

We claim that the focussing bipolar sequent calculus is isomorphic to the focussing sequent calculus of NL, so that proof construction can be performed indifferently in the two systems. This is a straightforward extension of the result shown in [3] for linear logic.

Given a set $\mathcal{A}$ of negative atoms, an $\mathcal{A}$-formula (resp. $\mathcal{A}$-monopole, $\mathcal{A}$-bipole) is a formula (resp. monopole, bipole) whose atomic subfomulas (leaves) are elements of $\mathcal{A}$ or their duals.

Let $\mathcal{A}, \mathcal{A}'$ be sets of negative atoms such that $\mathcal{A} \subset \mathcal{A}'$, and η be a bijection from the set of $\mathcal{A}$-formulas onto $\mathcal{A}'$ such that $\eta_a = a$ for all $a \in \mathcal{A}$. Thus, if F is an $\mathcal{A}$-formula which is not a negative atom, then η_F is just an unambiguous "flat" name for F. The *universal program (for η)* is the set of formulas of the form $\eta_F^\perp \otimes \nu(F)$ where F ranges over $\mathcal{A}$-formulas other than negative atoms, and $\nu(F)$ denotes the $\mathcal{A}'$-formula obtained by replacing in F each positive subformula G occurring in the scope of a negative connective by η_G. It is easy to show that the universal program contains only $\mathcal{A}'$-bipoles.

Note that we could just as well have chosen $\eta_F^\perp \odot \nu(F)$ for the bipoles of the universal program. Actually, such formulas lead to exactly the same inferences in the bipolar focussing sequent calculus as $\eta_F^\perp \otimes \nu(F)$. Actually, when considering (below) the purely cyclic fragment of non-commutative logic, we make use of $\eta_F^\perp \odot \nu(F)$, so as to manipulate bipoles from this fragment only.

Example 2.5 (Universal program) Consider the formula F in Figure 1. The successive layers of connectives of identical polarities have been drawn on the figure. Starting from the root, each dashed line shows the border of a positive layer and each dotted line the border of a negative layer. In this case, the formula $\nu(F)$ is obtained by replacing in F each subformula located on the dotted line by its image by η, which is a negative atom in $\mathcal{A}'$. In the example, there are four such subformulas F_1, F_2, F_3 (with topmost connectives $\odot_1$, $\oplus_2$ and $\oplus_3$) and $e^\perp$. (The indices are only mentioned for reference purpose.) It is easy to see that, by construction, the formula $\eta_F^\perp \otimes \nu(F)$ is an $\mathcal{A}'$-bipole.

Theorem 2.6 *Let $\mathcal{U}$ be the universal program for η. For any $\mathcal{A}$-formula F, there is an isomorphism between the set of proofs of F in the focussing sequent calculus and the set of proofs of η_F in the focussing bipolar sequent calculus $\Sigma[\mathcal{U}]$.*

Demonstration: Given a proof π of F in the focussing sequent calculus, the isomorphism proceeds in three steps:

1. Remove from π all the sequents (except the root) which are not conclusion of an instance of the entropy-focussing rule.
2. Relabel each instance of the entropy-focussing rule by the bipole $\eta_G^\perp \otimes \nu(G)$ computed from its focus G.
3. In each sequent (order variety) α of the resulting tree, replace any formula H by an occurrence of η_H.

Hence, a piece of proof in the focussing sequent calculus of the form

$$\dfrac{\cdots \quad \dfrac{}{\text{pos. axioms}} \quad \cdots \quad \dfrac{\dfrac{\cdots \quad \dfrac{}{\text{neg. axioms}} \quad \cdots \quad \dfrac{\vdots}{\alpha_i^j}\blacktriangleright \quad \cdots}{\omega * N_i}\,-}{\omega \blacktriangleright N_i}\not\blacktriangleright \quad \cdots}{\dfrac{\omega \blacktriangleright G}{\alpha}\blacktriangleright}\,+$$

is replaced by

$$\dfrac{\cdots \quad \dfrac{\vdots}{\eta_{\alpha_i^j}} \quad \cdots}{\eta_\alpha}\,\eta_G^\perp \otimes \nu(G)$$

where:

- the double lines labelled by $+$ and $-$ respectively stand for groups of positive and negative rules;
- a positive axiom is either a logical axiom (identity) or a 1-axiom, a negative axiom is a $\top$-axiom;
- the variable i (indexing the premises of the group of positive rules which are not axioms) ranges over a finite set I, and for each $i \in I$, the variable j (indexing the premises of the i-th group of negative rules) ranges over a finite set J_i; hence, the indices (i, j) (labelling the order varieties α_i^j) range over the disjoint union $\biguplus_{i \in I} J_i$;
- η_α is the order variety obtained by replacing in α any occurrence of a formula H by an occurrence of η_H.

And if the conclusion sequent F of π is not the conclusion of an entropy-focussing rule, the piece of proof below the lowermost occurrences of the entropy-focussing rule

$$\dfrac{\cdots \quad \dfrac{}{\text{neg. axioms}} \quad \cdots \quad \dfrac{\vdots}{\alpha_i}\blacktriangleright \quad \cdots}{F}\,-$$

is replaced by

$$\dfrac{\cdots \quad \dfrac{\vdots}{\eta_{\alpha_i}} \quad \cdots}{\eta_F}\,\eta_F^\perp \otimes \nu(F)$$

Conversely, given a proof ϖ of η_F in the focussing bipolar sequent calculus $\Sigma[\mathcal{U}]$, a proof of F in the focussing sequent calculus is defined as follows:

1. Take any inference in ϖ:

$$\dfrac{\eta_{\alpha_1} \quad \cdots \quad \eta_{\alpha_n}}{\eta_\alpha}\,\eta_G^\perp \otimes \nu(G)$$

2. Definition 2.3 provides a "full decomposition" of $\eta_G^\perp \otimes \nu(G)$, i.e., a proof of $(\eta_\omega \parallel \eta_G) \blacktriangleright \eta_G^\perp \otimes \nu(G)$ in the focussing sequent calculus with proper axioms $\eta_{\alpha_1} \cdots \eta_{\alpha_n}$ and $\alpha \subseteq \omega * G$. This full decomposition can only end by an occurrence of the $\otimes$-rule:

$$\dfrac{\dfrac{}{\eta_G \blacktriangleright \eta_G^\perp} \qquad \dfrac{\cdots \quad \overline{\text{axioms}} \quad \cdots \quad \eta_{\alpha_1} \quad \cdots \quad \eta_{\alpha_n}}{\eta_\omega \blacktriangleright \nu(G)}\,+/-}{(\eta_\omega \parallel \eta_G) \blacktriangleright \eta_G^\perp \otimes \nu(G)}\,\otimes$$

3. By removing the last rule and replacing each formula η_H by H, one gets a proof of $\omega \blacktriangleright G$:

$$\dfrac{\cdots \quad \overline{\text{axioms}} \quad \cdots \quad \alpha_1 \quad \cdots \quad \alpha_n}{\omega \blacktriangleright G}\,+/-$$

   which can be completed into a proof of α by adding an occurrence of the entropy-focussing rule.

The two mappings are clearly inverses of each other. $\square$

Example 2.7 (Isomorphism) Here is a proof in the focussing sequent calculus of non-commutative logic and its corresponding proof in the focussing bipolar sequent calculus. Note that, for clarity purpose, in the former proof, we make explicit the entropy step ($\subseteq$) which is in fact incorporated in the focussing inference above it.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \blacktriangleright a^\perp}}{a^\perp \blacktriangleright a}\,\not\blacktriangleright\blacktriangleright \qquad \dfrac{\dfrac{\dfrac{}{d \blacktriangleright d^\perp}}{d * d^\perp \oplus e^\perp}\,\blacktriangleright\oplus \qquad \dfrac{\dfrac{}{e \blacktriangleright e^\perp}}{e * d^\perp \oplus e^\perp}\,\blacktriangleright\oplus}{d^\perp \oplus e^\perp \blacktriangleright d \mathbin{\&} e}\,\not\blacktriangleright\&}{d^\perp \oplus e^\perp < a^\perp * a \odot (d \mathbin{\&} e)}\,\blacktriangleright\odot}{a \odot (d \mathbin{\&} e) * d^\perp \oplus e^\perp \parallel a^\perp}\,\subseteq}{a \odot (d \mathbin{\&} e) \blacktriangleright (d^\perp \oplus e^\perp) \parr a^\perp}\,\not\blacktriangleright\parr}{a \odot (d \mathbin{\&} e) * c^\perp \oplus ((d^\perp \oplus e^\perp) \parr a^\perp)}\,\blacktriangleright\oplus \quad \mapsto \quad \dfrac{\dfrac{\dfrac{}{a * \eta_3}\,B_3 \quad \dfrac{}{d * \eta_4}\,B_4 \quad \dfrac{}{e * \eta_4}\,B_4}{\eta_1 * (\eta_3 \parallel \eta_4)}\,B_1}{\eta_1 * \eta_2}\,B_2$$

where the bipoles used in the resulting proof are given by

$$\begin{aligned}
B_1 &= \eta_1^\perp \otimes (a \odot (d \mathbin{\&} e)) \\
B_2 &= \eta_2^\perp \otimes (c^\perp \oplus (\eta_3 \parr \eta_4)) \\
B_3 &= \eta_3^\perp \otimes a^\perp \\
B_4 &= \eta_4^\perp \otimes (d^\perp \oplus e^\perp)
\end{aligned}$$

3 Constraint based proof construction

A generic, constraint based proof construction procedure has been proposed for Linear Logic in [3]. Its principles are recalled here and presented in a way that facilitates their application to the cyclic and non-commutative cases, presented in the next sections.

3.1 Outline of the generic approach

One of the objectives of the constraint based approach to proof construction is to be able to deal with partial information about the object being constructed. This object is therefore not a ground proof, but an "abstract proof" describing a *non-empty* set of ground proofs derived from a common pattern. Typically, an abstract proof consists of a tree structure, labelled by pairwise distinct variable identifiers, called its main variables, together with a set of constraints linking these variables. The constraints are collected during the proof construction process. They may involve side variables which do not explicitly appear as labels of the tree. The main variables range over the set of ground sequents, so that each solution of the equation system corresponds exactly to one ground proof, obtained by replacing in the tree each main variable by the sequent assigned to it in the solution. The set of ground (or "concrete") proofs attached to an abstract proof can therefore be identified to the set of solutions of its equation system.

3.1.1 An elementary case: equations on first-order terms

Consider, in a traditional logic programming setting, a fixed set of Horn clauses $\mathcal{P}$ (the program). Proof construction in that case can be performed in a simplified sequent system $\Sigma_h[\mathcal{P}]$, similar to the bipolar sequent calculus, which makes apparent the only relevant choices made during the construction (that of a clause from $\mathcal{P}$ at each step). Essentially, the sequents in $\Sigma_h[\mathcal{P}]$ are ground atoms, and the inference rules are labelled by clauses of $\mathcal{P}$. For example, the clause $D = \forall x, y,\ q(x,y) \wedge r(g(y)) \supset p(f(x))$ yields the inference figures

$$\frac{q(u,v) \qquad r(g(v))}{p(f(u))}\, D$$

for any pair of ground terms u, v.
But the construction procedure never manipulates ground proofs built from these inferences, because that would require choices for ground terms u, v to instantiate x, y which may be irrelevant. Instead it manipulates abstract proofs, built from abstract inferences encoding only the relevant information needed to perform the inference. Abstract inferences are also labelled by the clauses of $\mathcal{P}$, but their sequents are now variables, constrained by the inference. Thus clause D yields the abstract inference

$$\frac{X_1 \qquad X_2}{X}\, D$$

where X, X_1, X_2 are variables, called the main variables, ranging over sequents (here ground atoms), and constrained by the following equations:

$$\begin{array}{l} X = p(f(x')) \\ X_1 = q(x', y') \\ X_2 = r(g(y')) \end{array}$$

The side variables x', y' are renamings of the logical variables x, y of the clause. They must be fresh (i.e., not used elsewhere in the equation system) and range over ground first-order terms. It is easy to see in that case that

- each abstract proof defines a set of concrete proofs obtained by solving its equation system;
- each concrete proof can be obtained as a solution of the equation system of an abstract proof.

This means that proof construction can be performed in the abstract system rather than the concrete one, the only condition being that at any stage in the construction, one must be able to compute all the solutions of the equations of the abstract proof constructed so far, and there must be at least one such solution. In the case of the abstract version of system $\Sigma_h[\mathcal{P}]$ above, at any point in the construction, the overall system will contain, for each main variable X, exactly one equation $X = a$ where X occurs as conclusion (except if X is a leaf) and exactly one equation $X = b$ where X occurs as premise (except if X is the root). Such pairs of equations reduce to $a = b$. We therefore have to solve sets of equations on first-order terms.
The purpose of a resolution procedure is to enumerate the set of solutions of a given equation system. In most cases, we cannot expect this set to be finite: for example, the equation $x = f(y)$ obviously has infinitely many solutions on ground first-order terms. Now, if infinite enumerations are allowed, however complex the equations are, so long as their variables range over countable domains, there is always a trivial procedure, also known as "generate and test", to enumerate all the solutions: it consists in enumerating all the possible assignments of the variables (e.g., all the ground first-order terms for each variable), testing each time if they satisfy the equations. It is clear that such a procedure would not qualify as a resolution procedure. In the case of equations on first-order terms, there exists a "true" resolution procedure, which shifts paradigm from "generate and test" to "simplify and generate", as is usual in constraint programming. It proceeds in two phases, called Simplification and Generation, described below.
Notation: in the sequel, $\mathcal{E}$ denotes a multiset of equations; the semi-colon simply denotes multiset union on equations systems; the equation system obtained from $\mathcal{E}$ by substituting a variable x by a term t is denoted $(x := t)\mathcal{E}$.

- With equations on first-order terms, the Simplification phase is also known as Unification. It aims at simplifying the equation system into a so-called solved form. A system is in solved form if it contains only equations of the form $x = t$, where x is a variable and t a term possibly containing variables, with no two equations having the same left-hand side. Intuitively, it is obvious that a system of this kind has a solution, and also that its solutions could be enumerated. Simplification proceeds by rewriting the equation system, according to the usual rules of Unification [15]:

$$\begin{array}{l}
[f(t_1, \dots, t_n) = g(u_1, \dots, u_m)]\ ;\ \mathcal{E} \ \longrightarrow\ \text{DEADEND if } f \text{ and } g \text{ are distinct function symbols} \\
[f(t_1, \dots, t_n) = f(u_1, \dots, u_n)]\ ;\ \mathcal{E} \ \longrightarrow\ [t_1 = u_1]\ ;\ \dots\ ;\ [t_n = u_n]\ ;\ \mathcal{E} \\
[x = t]\ ;\ \mathcal{E} \ \longrightarrow\ \text{DEADEND if } x \text{ occurs inside } t \\
[x = t]\ ;\ \mathcal{E} \ \longrightarrow\ [x = t]\ ;\ (x := t)\mathcal{E} \ \text{ if } x \text{ does not occur inside } t \text{ and occurs in } \mathcal{E}
\end{array}$$

  where f and g are any function symbols of respective arities n and m.

- Generation proceeds from systems in solved form as obtained at the end of the Simplification procedure (when it does not fail) and aims at producing complete solutions, i.e., systems in which the equations are all of the form $x = u$, where x is a variable and u a ground term, with, of course, no two equations with the same left-hand side. Generation also proceed by rewriting the equation systems, but its rewriting steps may be non-deterministic (unlike the Simplification steps), which means that they may transform an equation system into a (possibly infinite) set of new, alternative equation systems. Thus, an equation system $\mathcal{E}$ in solved form, containing a first-order variable x not occurring in any left-hand side, rewrites into a set of alternative equation systems, with one alternative for each arbitrary choice of ground term t, yielding the equation system $(x := t)\mathcal{E}$. This is denoted as follows:

$$\mathcal{E} \longrightarrow \left|\begin{array}{ll} \rightarrow & \cdots \\ \rightarrow & (x := t)\mathcal{E} \\ \rightarrow & \cdots \end{array}\right. \left.\vphantom{\begin{array}{l} \cdots \\ \cdots \\ \cdots \end{array}}\right\} \text{one alternative for each ground term } t$$

By combining Simplification (here, unification) and Generation (here, simple arbitrary instantiation of the variables which were not assigned during the unification), we obtain a complete procedure for the resolution of equations over first-order terms. In what sense is it "better" than the trivial resolution procedure based on a "generate and test" approach ? We address this question in the next section, in the wider context of arbitrary constraint systems, beyond equations on first-order terms, since, in our context of proof construction, the constraints attached to abstract proofs can be of any type. This is non trivial as the notion of a system in "solved form", which constitutes the articulation between Simplification and Generation in the case of equations on first order terms, does not generalise easily to arbitrary constraints.

3.1.2 Constraint solving: general properties

In the sequel, we are going to manipulate constraint systems, in exactly the same way as in the example above, except that we are going to deal with more complex constraints than equations between first-order terms. As above, the constraints will be collected from the abstract proof built by the proof construction process, and will involve main variables ranging over sequents, and, possibly, other side variables. The resolution procedure will also consist of two phases, both described by possibly non-deterministic rewriting steps on constraint systems, and designed in such a way as to never mimic the "generate and test" approach to constraint resolution. Since the rewriting steps can be non-deterministic in both phases, resolution builds a tree (or "tableau") of constraint systems. The nodes of such a tableau, labelled by the constraint systems, are called states. A state is said to be open if it has not been expanded yet. The output of the resolution procedure is a tableau $\mathcal{T}$ which must satisfy the following requirements:

- The root state of $\mathcal{T}$ is labelled with the input constraint system.
- **Preservation**: Each expansion of a state in $\mathcal{T}$ must preserve the set of solutions. In other words, restricted to the main variables, the set of solutions of the constraint system at one non-open state of $\mathcal{T}$ must be equal to the union of the sets of solutions of the constraint systems at each of its successors.
- **Resolution**: The open states (leaves) of $\mathcal{T}$ must be labelled with constraint systems in fully solved form, i.e., containing only constraints of the form $X = c$ where X is a main variable and c is a ground sequent, which thus define a complete solution.
- **Termination**: $\mathcal{T}$ must have a finite depth, meaning that the resolution procedure performs only bounded sequences of rewriting steps to reach any of the solutions. However, $\mathcal{T}$ may be of infinite width, meaning that some of the choices at the non deterministic steps may involve infinitely many alternatives.

The two phases of the procedure have different properties:

- The "Simplification" phase starts from the root state and must yield a *finite* tableau, the open states of which are all *consistent*. Given the Termination property (bounded depth), being finite means that all the choices performed during that phase involve only finite alternatives (bounded width).
- The "Generation" phase continues the expansion of the tableau after the Simplification phase, but all the states created during this phase must be *consistent*. Since the Simplification phase produces only consistent open states, this means that the Generation never explores alternatives which lead to an inconsistency (no dead-end).

These two phases are illustrated in Figure 2. The main advantage of this approach, compared to the "generate and test" approach, is that it provides a *decision procedure* for the consistency of the input constraint system. Indeed, the consistency of the root state is decided by the existence of an open state at the outcome of the Simplification phase, and this phase explores only finitely many alternatives. Conversely, in the "generate and test" approach, decision can only be made after all the alternatives have been explored, and there are typically infinitely many alternatives. In other words, the Simplification

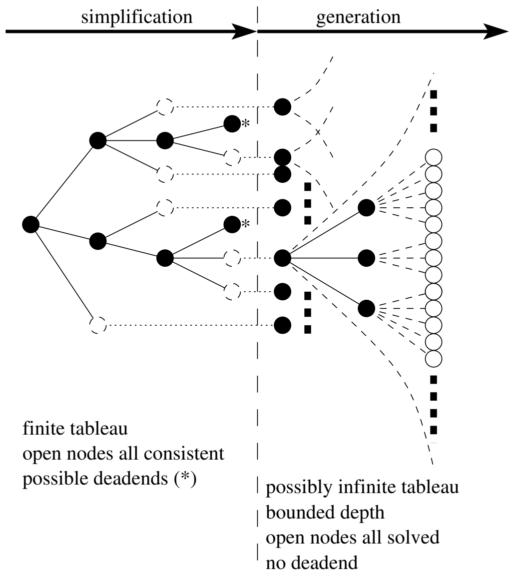


Figure 2: The two phases in the resolution procedure

phase retains only the minimal choice points which are needed for the decision and postpones, to the Generation phase, all the inessential choices. Observe here, again, the lazy construction principle at work, which motivates our whole approach.
Another advantage of the "simplify and generate" approach, which will appear below, is that it is fully incremental. Indeed, suppose that we have built the tableau for a given constraint system at the root (in practice, we only build the Simplification tableau - which is finite), and that we add new constraints at the root. The new tableau need not be re-computed from scratch, but can be derived from the old one by adding new branches and propagating new constraints at each node. In the perspective of a proof construction process, which continuously feeds new constraints, this is essential. Conversely, with a "generate and test" approach, incrementality is impossible since each test is performed on completely defined objects and need to be re-done for each new object.
It is easy to check that the Simplification and Generation procedures described in Section 3.1.1 in the case of equations on first-order terms satisfy the required properties. Note however that, unlike unification which involves rewriting steps with only one or zero alternatives (zero in case of failure), Simplification steps in general can involve an arbitrary (but finite) number of alternatives. The Generation phase, on the other hand, may explore an arbitrary (possibly infinite) but non-null number of alternatives at each step, as in the case of equations on first-order terms.

3.2 Abstract proofs in the focussing bipolar sequent calculus

We now apply the generic constraint based approach outlined in Section 3.1 to the focussing bipolar sequent calculus $\Sigma[\mathcal{F}]$ defined in Section 2.6, for some given set $\mathcal{F}$ of bipoles. Obviously, the intention is that $\mathcal{F}$ be the Universal program of Theorem 2.6, but we don't need this assumption here, as it appears that the specific form of the bipoles is irrelevant to the procedure. In the sequel, we take bipole to mean a bipole of $\mathcal{F}$ and the inferences of $\Sigma[\mathcal{F}]$ are called *concrete inferences*.

Places and sort conventions for variables

We first need to make the support sets in the concrete inferences explicit. For this, we choose a countably infinite set of *places*, and we assume that the support set of any order and order variety in concrete proofs is a (finite) set of places. Thus, places stand for formula occurrences, and we assume that each place u has a *type*, which is the formula that it holds. We write $u : F$ to mean that place u is of type F, i.e., u is an occurrence of formula F. As usual, it is assumed that for any formula F, there are infinitely many places of type F.
In order to define abstract inferences, we make use of sorted variables, taken from a countably infinite set of variables. The sort of a variable is either: place, order or order variety. For simplification purpose, we choose variable names so that their sort be implicit: except when stated otherwise, z (possibly subscripted as in $z_1, z_2, z_3, z', z''\ldots$) denotes a variable ranging over places, Y (possibly subscripted) denotes a variable ranging over orders, and X (possibly subscripted) denotes a variable ranging over order varieties. A solution of a constraint system is an assignment of all the variables to entities of the corresponding sort (i.e. places for place variables, orders for order variable, etc.). We use the letter σ (possibly subscripted) to denote such variable assignments.

We first define for each formula F a set $||F||$ of orders on occurrences of subformulas of F obtained by replacing in the topmost negative layer of F each connective $\parr$ by $\|$, each connective $\triangledown$ by $<$, and by replacing each connective $\&$ by one of its arguments. Formally:

Definition 3.1 (Orders associated to a formula) *The* set of orders associated to a formula F *is the set* $||F||$ *of orders on occurrences of formulas defined inductively as follows:*

$$
\begin{array}{lcl}
||F_1 \parr F_2|| & = & \{\omega_1 \| \omega_2 \ \textit{such that} \ \omega_1 \in ||F_1|| \ \textit{and} \ \omega_2 \in ||F_2||\} \\
||F_1 \triangledown F_2|| & = & \{\omega_1 < \omega_2 \ \textit{such that} \ \omega_1 \in ||F_1|| \ \textit{and} \ \omega_2 \in ||F_2||\} \\
||F_1 \& F_2|| & = & ||F_1|| \cup ||F_2|| \\
||\bot|| & = & \{\epsilon\} \\
||\top|| & = & \emptyset \\
||F|| & = & \textit{the singleton consisting in the unique order on } \{F\} \textit{ in all the other cases.}
\end{array}
$$

Note that, by definition of monopoles and bipoles, we have:

- if F is a monopole, then the orders in $||F||$ involve only occurrences of negative atoms;
- if F is a bipole, then the orders in $||F^\perp||$ involve only occurrences of duals of either positive atoms or monopoles.

Definition 3.2 (Abstract inference) *An* abstract inference *is of the form*

$$
\frac{\cdots \qquad X_{i,\tau} \qquad \cdots}{X} \, F, \omega \tag{1}
$$

where X and the $X_{i,\tau}$ are the main variables (which range over order varieties and are assumed to be distinct), F is a bipole, $\omega \in ||F^\perp||$ and the premises $X_{i,\tau}$ form a family indexed by:

- *$i \in |\omega|$ such that $i : G^\perp$ and G is a monopole,*
- *$\tau \in ||G||$.*

The constraint system attached to such an abstract inference expresses that the variables $X, X_{i,\tau}$ can only be assigned order varieties which, when substituted in the abstract inference, yield a concrete inference labelled by F and ω.

Definition 3.3 (Constraint system attached to an abstract inference) *The* constraint system attached to *the abstract inference (1) is built as follows.*

- *Introduce one side variable Y_i ranging over orders for each place i in $|\omega|$, and add the* conclusion constraint[1]*:*

$$\boxed{X \subseteq \overline{\omega}((Y_i)_{i \in |\omega|})}$$

  The variables Y_i are assumed to be distinct.

- *For each place i in $|\omega|$, we know that the type of i is the dual of either a positive atom or a monopole.*
  - *If the type of i is the dual of a positive atom $a^\perp$, then introduce a fresh variable z_i ranging over places and add the* terminal premise constraint*:*

$$\boxed{Y_i = z_i, \quad z_i : a}$$

  - *If the type of i is the dual of a monopole G, then, for each order τ in $||G||$, add the* non-terminal premise constraint*:*

$$\boxed{X_{i,\tau} = \tau * Y_i}$$

In the case of a terminal premise constraint, the notation $Y = z$ is a slight abuse, since the two variables do not have the same sort and hence do not range over the same objects; what is meant is in fact that Y is the (unique) order on the singleton support set $\{z\}$. It is often convenient to even omit the place variable z and the type constraint $z : a$, and write directly $Y = a$. Observe the different treatments between positive and negative atoms in the bipoles: positive atoms are assigned place variables while negative atoms are directly assigned places (because their places can be arbitrarily chosen). The resolution procedure matches positive atoms with negative ones, thus instantiating the corresponding place variables.

[1]Here, we slightly abuse notation, by identifying any order variety α (here $\overline{\omega}$) with the function that maps any family indexed by $|\alpha|$ of orders into the order variety obtained by substituting in α each place by the corresponding order in the family.

Definition 3.4 (Abstract proof) *An* abstract proof *is a tree built in the usual way by assembling abstract inferences. The* constraint system *attached to an abstract proof is the conjunction of the constraint systems attached to its inferences, in which the side variables are renamed apart so as to be all distinct. Furthermore, the orders τ occurring in the non-terminal premise constraints $X_{i,\tau} = \tau * Y_i$ are assumed to have pairwise disjoint support sets; this is harmless, as places can be renamed.*

Note that all the main variables range over order varieties while all the side variables range over orders.

Example 3.5 Consider the bipole

$$F \;=\; \underbrace{p^\perp}_{1} \otimes \Big((\underbrace{a \parr ((b \triangledown c) \,\&\, (b' \parr c'))}_{2}) \odot \underbrace{q^\perp}_{3} \odot (\underbrace{r^\perp}_{4} \otimes (\underbrace{d \triangledown e}_{5})) \Big).$$

Then $||F^\perp||$ consists of a single order:

$$\Big(((\underbrace{d \triangledown e}_{5})^\perp \parallel (\underbrace{r^\perp}_{4})^\perp) < (\underbrace{q^\perp}_{3})^\perp < (\underbrace{a \parr ((b \triangledown c) \,\&\, (b' \parr c'))}_{2})^\perp \Big) \parallel (\underbrace{p^\perp}_{1})^\perp.$$

Hence we introduce 5 distinct side variables Y_1, Y_2, Y_3, Y_4, Y_5, and the constraints become:

$$\frac{X_{21} \qquad X_{22} \qquad X_5}{X} \qquad \begin{cases} X \subseteq \overline{((Y_5 \parallel Y_4) < Y_3 < Y_2) \parallel Y_1} \\ Y_1 = p \\ X_{21} = Y_2 * a \parallel (b < c) \\ X_{22} = Y_2 * a \parallel b' \parallel c' \\ Y_3 = q, \quad Y_4 = r \\ X_5 = Y_5 * (d < e) \end{cases}$$

Variables X_{21} and X_{22} are the two premises indexed by the subformula numbered 2 in F and the two orders computed from it:

$$||a \parr ((b \triangledown c) \,\&\, (b' \parr c'))|| \;=\; \{\ a \parallel (b < c)\ ,\ a \parallel b' \parallel c'\ \}.$$

The following theorem, which may be shown by a straightforward induction on F, states that abstract proofs are indeed abstract representations of the concrete proofs.

Theorem 3.6 *Let F be a bipole and $\alpha_1, \ldots, \alpha_n, \alpha$ be order varieties. The concrete inference*

$$\frac{\alpha_1 \quad \cdots \quad \alpha_n}{\alpha}\, F$$

is in the focussing bipolar sequent calculus if, and only if, for some $\omega \in ||F^\perp||$, the assignment $X = \alpha; (X_i = \alpha_i)_{i=1}^n$ is a solution (in the main variables) of the constraint system attached to the abstract inference

$$\frac{X_1 \qquad \cdots \qquad X_n}{X}\, F, \omega.$$

4 Constraint resolution algorithms for non-commutative logic

We now come to the problem of defining a resolution algorithm for the constraint systems generated by proof construction in the focussing bipolar sequent calculus (and hence, indirectly, in non-commutative Logic). Such an algorithm can be used to ensure that the constraint system at any stage in the construction is consistent. This algorithm is an instance of the generic method presented in Section 3.1.2. It therefore consists of a non-deterministic rewriting procedure building a tableau of constraint systems, with two distinct phases of Simplification and Generation, and satisfying the properties of Preservation, Resolution and Termination. For presentation purpose, we first recall the algorithm in the commutative case viewed here as a fragment of non-commutative logic. We then show that a similar approach applies to the other important fragment: cyclic logic. Unfortunately, the method does not extend easily to the whole system, and we propose a different algorithm capable of dealing with the whole system.
Although we have presented here the sequent calculus in the propositional case only, the algorithms presented below work just as well in the first-order case. A complete treatment of the first-order commutative case is given in [3]. It extends directly to the algorithms presented below, both in the cyclic and full non-commutative cases. Indeed, all these algorithms essentially work by matching positive occurrences with negative occurrences of atoms, producing equations of the form $[a = b]$ between atoms. In the propositional case, such equations either disappear immediately, if a and b are actually the same atom, or generate a failure (DEADEND) otherwise. In the first order case, the atoms may contain variables ranging over first-order

ground terms, introduced by existential quantification, and these equations are treated by the unification procedure recalled in the Section 3.1.1. The treatment of universal quantification is based on the ideas of [16, 6, 20] and slightly alters the unification procedure to account for so called eigen-variables. The interested reader is referred to [3] for a complete description, which, again, applies just as well here. Note that the easiness with which the first-order case is accounted for should not come as a surprise: the whole method presented here elaborates on the notion of unification (more generally constraint solving), which is an intrinsic first-order mechanism.

4.1 The degenerated cases: commutative and cyclic logic

In both the commutative and cyclic fragments, the resolution algorithm is particularly simple, because it is possible to actually reduce the constraints when matching positive and negative occurrences of atoms, by simply removing these occurrences. In both cases, the Simplification procedure propagates information downward in the abstract proof, while the Generation procedure propagates information upward. The finiteness of the proof ensures that both procedures are bounded (in steps, but of course, the Generation procedure makes unbounded choices).

4.1.1 The commutative case

We first consider the commutative fragment of Non-commutative logic, where formulas use no multiplicative connectives other than the commutative ones: $\otimes, \parr$. When F is a formula in this fragment, $||F||$ contains orders of the form $F_1 \parallel \cdots \parallel F_n$, which we write, to simplify, $F_1 \cdots F_n$ (omitting the symbol $\parallel$, implicitly associative commutative). Here, both order varieties and orders simplify to multisets.
For a given bipole F, the conclusion constraint $X \subseteq \varpi(Y_1, \dots, Y_m)$ becomes in fact a multiset equality $X = Y_1 \cdots Y_m$ (indeed, remember that the inclusion concerns the structure, which is here void, not the support sets, which must be equal). Using commutativity, it is always possible to assume that the terminal premise constraints concern the last p side variables $Y_{n+1}, \dots, Y_{n+p}$ (where $n + p = m$). They are of the form $Y_{n+1} = a_1, \dots, Y_{n+p} = a_p$ and can be directly reported in the conclusion constraint. It becomes $X = \Delta Y_1 \cdots Y_n$ where Δ is the multiset $a_1, \dots, a_p$. The side variables $Y_1, \dots, Y_n$ on the other hand correspond to monopoles of F. For each such monopole G and each order τ in $||G||$, there is a main variable X' together with the non-terminal premise constraint $X' = \tau * Y$, which, again, becomes the multiset constraint $X' = \Gamma Y$ (where Γ is the multiset τ).
Each main variable X which is both the conclusion of an abstract inference and the premise of another

$$\cfrac{\cdots \qquad \cfrac{\cdots}{X} \rightsquigarrow [X = \Delta Y_1 \cdots Y_n] \qquad \cdots}{\cdots} \rightsquigarrow [X = \Gamma Y]$$

thus leads to two constraints $X = \Gamma Y = \Delta Y_1 \cdots Y_n$. The system can first be solved in the side variables only, using the constraints of the form $\Gamma Y = \Delta Y_1 \cdots Y_n$. The solutions in the main variables are then straightforwardly derived using the constraints of the form $X = \Gamma Y$.
The resolution algorithm for such constraint systems has been completely described in [3]. It deals with (oriented) constraints of the following general form[2]:

$$\Gamma Z_1 \cdots Z_m = \Delta Y_1 \cdots Y_n$$

where $Y_1, \dots, Y_n, Z_1, \dots, Z_m$ are side variables ranging over multisets of atoms and Γ, Δ are multisets of atoms. The orientation follows that of the proof: the variables in the left-hand side of a constraint are attached to an inference which is just below the inference to which the variables in the right-hand side are attached. The Simplification phase aims at reducing the system to a system of "generators", which are constraints of this form in which Δ is empty and $n \geq 1$. The Generation phase then uses these generators to produce all the solutions of the system.

- Simplification procedure: main rule

$$
[\Gamma Z_1 \cdots Z_m = \Delta a Y_1 \cdots Y_n] \; ; \; \mathcal{E}
\longrightarrow
\begin{cases}
\left.\begin{array}{l}
\mapsto \;\cdots \\
\mapsto \;[\Gamma' Z_1 \cdots Z_m = \Delta Y_1 \cdots Y_n] \; ; \; [a = b] \; ; \; \mathcal{E} \\
\mapsto \;\cdots
\end{array}\right\} (i) \\
\left.\begin{array}{l}
\mapsto \;\cdots \\
\mapsto \;[\Gamma Z_1 \cdots Z'_j \cdots Z_m = \Delta Y_1 \cdots Y_n] \; ; \; (Z_j := a Z'_j)\mathcal{E} \\
\mapsto \;\cdots
\end{array}\right\} (ii)
\end{cases}
$$

with (i) one output state for each b such that $\Gamma = b, \Gamma'$ and (ii) one output state for each $j = 1, \dots, m$, introducing a fresh side variable Z'_j.

[2] It is easier to formulate the algorithm in the general case, although in the case we are dealing with, we always have $m = 0, 1$.

Thus, each element a of Δ is sent either onto an element b of Γ (triggering the unification $a = b$ with potential failure) or into Z_j for some $j = 1, \ldots, m$, in which case Z_j must be of the form aZ'_j. Obviously, there are finitely many alternatives for that choice.

- Simplification procedure: other rules

$$\begin{array}{lcll} [\Gamma Z_1 \cdots Z_m = \epsilon] \ ; \ \mathcal{E} & \longrightarrow & \text{DEADEND} & \text{if } \Gamma \text{ is non empty} \\ [\epsilon Z_1 \cdots Z_m = \epsilon] \ ; \ \mathcal{E} & \longrightarrow & (Z_1 := \epsilon) \cdots (Z_m := \epsilon)\mathcal{E} & \end{array}$$

- Generation procedure: if $n \geq 1$ and $Z_1, \ldots, Z_m$ do not occur in $\mathcal{E}$ on a right-hand side

$$[\Gamma Z_1 \cdots Z_m = \epsilon Y_1 \cdots Y_n] \ ; \ \mathcal{E} \ \longrightarrow \left| \begin{array}{l} \mapsto \ \cdots \\ \mapsto \ (Y_i := \Delta_i)_{i=1}^{n} (Z_j := \Gamma_j)_{j=1}^{m} \mathcal{E} \\ \mapsto \ \cdots \end{array} \right. \left. \vphantom{\begin{array}{l} a \\ b \\ c \end{array}} \right\} (i)$$

with (i) one output state for each arbitrary choice of multisets $\Gamma_1, \ldots, \Gamma_m$ and each multiset partition $\Delta_1, \ldots, \Delta_n$ of $\Gamma\Gamma_1 \cdots \Gamma_m$. Thus, Generation just works as Simplification, but in a reverse direction: each element of $\Gamma\Gamma_1 \cdots \Gamma_m$ is sent into Y_i for some $i = 1, \ldots, n$. Note that, assuming $n \geq 1$, there is always a finite but non null number of ways to partition any given multiset into n pieces. The case $n = 0$ would make the partition impossible, but has already been treated in the Simplification and hence cannot occur here. Note also that if $m \geq 1$, there are infinitely many alternatives for the choice of a m-uple of multisets, and a single one if $m = 0$ (in fact, with constraint systems coming from abstract proofs, m is always either 0 or 1).

It is shown in [3] that the Simplification phase produces a finite tableau with no inconsistent open states, that the Generation phases produces only consistent states, and that the overall resolution procedure satisfies the expected properties (Preservation, Termination, Resolution). All these properties hold only if the initial constraint system is obtained from an abstract proof, ensuring a certain regularity in the shape of the system (see [3] for a precise definition of "regularity"): the resolution algorithm defined here is not a general multiset constraint resolution procedure. In particular, with regularity, the Simplification procedure propagates information downwards in the (finite) abstract proof, so that the output tableau is always bounded by the size of the abstract proof.

4.1.2 The cyclic case

We now consider the cyclic fragment of Non-commutative logic, where the bipoles involve no multiplicative connectives other than the non-commutative ones: $\odot, \triangledown$. When F is a formula in this fragment, $||F||$ contains orders of the form $F_1 < \cdots < F_n$, which we write, to simplify, $F_1 \cdots F_n$ (omitting the symbol $<$ implicitly associative non-commutative). Here, orders are simple lists while order varieties are cycles. If ω is a list, $\overline{\omega}$ is the corresponding cycle (obtained by joining the two ends of the list).
For a given bipole F, the conclusion constraint $X \subseteq \overline{\omega}(Y_1, \ldots, Y_m)$ becomes in fact a cycle equality $X = \overline{Y_1 \cdots Y_m}$. However, unlike the commutative case, it is not possible here to group together the side variables corresponding to terminal premise constraints. Instead, one obtains constraints of the form:

$$X = \overline{\Gamma Y} = \overline{\Delta_1 Y_1 \cdots \Delta_n Y_n}$$

where $\Gamma, \Delta_1, \ldots, \Delta_n$ are lists of atoms (each Δ_i, possibly empty, accounting for the possible presence of side variables corresponding to terminal premise constraints before each side variable corresponding to a non-terminal premise constraint). The first step in the Simplification procedure reduces equalities on cycles to equalities on lists:

$$\begin{array}{l} [\overline{\Gamma Y} = \overline{\Delta_1 Y_1 \cdots a\Delta_i Y_i \cdots \Delta_n Y_n}] \ ; \ \mathcal{E} \\ \longrightarrow \left| \begin{array}{ll} \mapsto \ \cdots & \\ \mapsto \ [\Gamma'' Y \Gamma' = \Delta_i Y_i \cdots \Delta_n Y_n \Delta_1 Y_1 \cdots \Delta_{i-1} Y_{i-1}] \ ; \ [a = b] \ ; \ \mathcal{E} & \\ \mapsto \ \cdots & \\ \mapsto \ [Y'' \Gamma Y' = \Delta_i Y_i \cdots \Delta_n Y_n \Delta_1 Y_1 \cdots \Delta_{i-1} Y_{i-1}] \ ; \ (Y := Y' a Y'')\mathcal{E} & \} \ (ii) \end{array} \right. \end{array}$$

with (i) one output state for each b such that $\Gamma = \Gamma' b \Gamma''$ and (ii) one output state introducing fresh side variables Y', Y''. Thus, an arbitrary element a of $\Delta_1, \ldots, \Delta_n$ is sent either onto an element b of Γ producing the constraint $[a = b]$ or into Y, which must in that case be of the form $Y' a Y''$. Note that the choice of a, although arbitrary, is always possible, since $\Delta_1, \ldots, \Delta_n$ cannot be empty. This is due to the fact that bipoles always contain at least one positive atom.
After the initial step, the resolution algorithm manipulates (oriented) constraints of the form

$$\Gamma_1 Z_1 \cdots \Gamma_m Z_m \Gamma = \Delta_1 Y_1 \cdots \Delta_n Y_n$$

where $Z_1, \ldots, Z_m, Y_1, \ldots, Y_n$ are side variables ranging over lists of atoms, and $\Gamma_1, \ldots, \Gamma_m, \Delta_1, \ldots, \Delta_n$ are lists of atoms.
The resolution algorithm for such constraint systems is similar to that in the commutative case.

- Simplification procedure: main rule

$$
\begin{array}{l}
[\Gamma_1 Z_1 \cdots \Gamma_m Z_m \Gamma = \epsilon Y_1 \cdots \epsilon Y_{i-1} a \Delta_i Y_i \cdots \Delta_n Y_n]\ ;\ \mathcal{E} \\
\longrightarrow \left|
\begin{array}{l}
\mapsto\ \cdots \\
\left.\begin{array}{l}
\mapsto\ [\Gamma_1 Z_1 \cdots \Gamma_{j-1} Z_{j-1} \Gamma'_j = \epsilon Y_1 \cdots \epsilon Y_{i-1}]\ ;\ [\Gamma''_j Z_j \cdots \Gamma_m Z_m \Gamma = \Delta_i Y_i \cdots \Delta_n Y_n]\ ; \\
\quad [a = b]\ ;\ \mathcal{E} \\
\mapsto\ \cdots
\end{array}\right\} (i) \\
\left.\begin{array}{l}
\mapsto\ \cdots \\
\mapsto\ [\Gamma_1 Z_1 \cdots \Gamma_j Z'_j \epsilon = \epsilon Y_1 \cdots \epsilon Y_{i-1}]\ ;\ [\epsilon Z''_j \Gamma_{j+1} Z_{j+1} \cdots \Gamma_m Z_m \Gamma = \Delta_i Y_i \cdots \Delta_n Y_n]\ ; \\
\quad (Z_j := Z'_j a Z''_j) \mathcal{E} \\
\mapsto\ \cdots
\end{array}\right\} (ii)
\end{array}\right.
\end{array}
$$

with (i) one output state for each $j = 1, \dots, m$ and each b such that $\Gamma_j = \Gamma'_j b \Gamma''_j$, and (ii) one output state for each $j = 1, \dots, m$, introducing fresh side variables Z'_j, Z''_j.

Thus, again, each element a of $\Delta_1 \cdots \Delta_n$ is sent either onto an element b of Γ_j (for some $j = 1, \dots, m$, triggering the unification $a = b$ with potential failure) or into Z_j (for some $j = 1, \dots, m$), in which case Z_j must be of the form $Z'_j a Z''_j$.

- Simplification procedure: other rules

$$
\begin{array}{lll}
[\Gamma_1 Z_1 \cdots \Gamma_m Z_m \Gamma = \epsilon]\ ;\ \mathcal{E} & \longrightarrow & \text{DEADEND if } \Gamma_1 \cdots \Gamma_m \Gamma \text{ is non empty} \\
[\epsilon Z_1 \cdots \epsilon Z_m \epsilon = \epsilon]\ ;\ \mathcal{E} & \longrightarrow & (Z_1 := \epsilon) \cdots (Z_m := \epsilon) \mathcal{E}
\end{array}
$$

- Generation procedure: if $n \geq 1$ and $Z_1, \dots, Z_m$ do not occur in $\mathcal{E}$ on a right-hand side

$$
[\Gamma_1 Z_1 \cdots \Gamma_m Z_m \Gamma = \epsilon Y_1 \cdots \epsilon Y_n]\ ;\ \mathcal{E} \longrightarrow \left| \left.\begin{array}{l}
\mapsto\ \cdots \\
\mapsto\ (Y_i := \Delta_i)_{i=1}^{n} (Z_j := {\Gamma'}_j)_{j=1}^{m} \mathcal{E} \\
\mapsto\ \cdots
\end{array}\right\} (i) \right.
$$

with (i) one output state for each arbitrary choice of lists $\Gamma'_1, \dots, \Gamma'_m$ and each list partition $\Delta_1, \dots, \Delta_n$ of $\Gamma_1 \Gamma'_1 \cdots \Gamma_m \Gamma'_m \Gamma$.

Following exactly the same kind of argument as in [3], it can be shown that the Simplification phase produces a finite tableau with consistent open states (if any), that the Generation phases produces only consistent states, and that the overall resolution procedure satisfies the expected properties (Preservation, Termination, Resolution). Again, this assumes that the initial constraint system is obtained from an abstract proof, not just any constraint system on lists.

4.2 The general case

4.2.1 The problem

In the general case, the situation is much more complex than in the previous cases, for two reasons:

- The general position of a point in a series-parallel order structure is more difficult to capture than in the case of multiset and list structures. Indeed, in the multiset case, $a \in \Gamma$ iff there exists a multiset Γ_1 such that $\Gamma = a\Gamma_1$; and in the list case, $a \in \Gamma$ iff there exist lists Γ_1, Γ_2 such that $\Gamma = \Gamma_1 a \Gamma_2$. This property is essential to the proper execution of the Simplification procedure. In the case of order structures, we would like to have, similarly, a fixed order τ such that $a \in |\omega|$ iff there exist a family of orders $(\omega_i)_{i=1,\dots,n}$ and $\omega = \tau(a, \omega_1, \dots, \omega_n)$. Unfortunately, there is no such fixed τ.

- Furthermore, in the multiset case, the resolution of the constraint systems has a monotonicity property, in the sense that if a constraint $\Gamma Y = \Delta Y_1 \cdots Y_n$ has a solution σ, ie.$\Gamma \sigma(Y) = \Delta \sigma(Y_1) \cdots \sigma(Y_n)$, then any constraint $\Gamma' Y = \Delta Y_1 \cdots Y_n$ where $\Gamma \subseteq \Gamma'$ also has solutions which coincide with σ on Γ, obtained by arbitrarily distributing the elements of Γ' which are not in Γ among $\sigma(Y_1) \cdots \sigma(Y_n)$. This is exactly how the Generation procedure works, to build complete solutions from the partial ones obtained at the outcome of the Simplification procedure. Monotonicity also holds in the case of lists. In the case of arbitrary orders, we would need to infer solutions of $\omega' * Y \subseteq \alpha(Y_1, \dots, Y_n)$ from any solution of $\omega * Y \subseteq \alpha(Y_1, \dots, Y_n)$ whenever $\omega' \lceil_{|\omega|} = \omega$. This is unfortunately impossible, as shown by the following example. Consider the system

$$a \parallel (b < c) * Z = X \subseteq \overline{(c < a) \parallel b \parallel Y}$$

After matching the atoms, we obtain a solution where $Z = Y = \epsilon$. However, if we modify the constraint by expanding its left-hand side with a single atom, preserving the existing order, we obtain

$$(d < a) \parallel (b < c) * Z = X \subseteq \overline{(c < a) \parallel b \parallel Y}$$

which has no solution, since the triple (d, b, c) is in the left-hand side, but d must be in $|Y|$ and the triple (d, b, c) is not in the right-hand side. The importance of this problem in the Generation procedure is illustrated in Example 4 of Section 4.5.

4.2.2 Weakening of negative formulas

In the multiset and list cases, the monotonicity property ensures that Simplification only needs to deal with atoms which appear as focus in the abstract proof: the other atoms cannot cause problems, once in the Generation procedure. Although, as shown above, this monotonicity property does not hold in NL, it holds in an "affine" version of NL in which the weakening rule is allowed when the introduced formula is negative:

$$\frac{\overline{\omega}}{\omega * A}\ \text{W, if } A \text{ negative}$$

It is a structural rule, just as entropy, and, like entropy, it can always (by permutation) be incorporated into the focussing inference rule. Observe that the weakening rule applies only to negative formulas, very much as in polarised LL [13]. This means that it is not allowed to cancel focalised formulas (which are positive). In a proof construction perspective, this is essential, otherwise any bipole would be applicable in any sequent.
This slightly alters the side condition on the focussing inference rule in the focussing sequent calculus: instead of requiring $\alpha \subseteq \omega * A$, we now have $\alpha \Subset \omega * A$ where the $\Subset$ relation is defined as follows.

Definition 4.1 *Let α, β be order varieties: $\alpha \Subset \beta$ if and only if $|\beta| \subseteq |\alpha|$ and $\alpha\lceil_{|\beta|} \subseteq \beta$.*

The condition $|\beta| \subseteq |\alpha|$ characterises weakening while $\alpha\lceil_{|\beta|} \subseteq \beta$ characterises entropy.
The bipolar sequent calculus is similarly modified: in Definition 2.3, the condition $\alpha \subseteq \overline{\omega}$ (third bullet) is simply replaced by $\alpha \Subset \overline{\omega}$. Finally, the constraint system attached to an abstract inference is modified only for the conclusion constraint, which becomes:

$$\boxed{X \Subset \overline{\omega}((Y_i)_{i \in |\omega|})}$$

It is now obvious that monotonicity holds with constraint systems of this form, since the extended constraint can always be solved by cancelling the added points.

4.2.3 Decomposition of the constraints

In order to solve the first problem mentioned above, each constraint on order varieties is first decomposed into two constraints: one in terms of support set information (elements of the support set) and one in terms of ordering information (pairs or cyclic triples of the graph).

- Non terminal premise constraints of the form $X = \omega * Y$ become:

$$\begin{aligned} |X| &= |\omega| + |Y| \quad (2)\\ X &= \overline{\omega} + \omega|Y| + |\omega|Y + \overline{Y} \quad (3) \end{aligned}$$

- Conclusion constraints of the form $X \Subset \alpha(Y_1, \ldots, Y_n)$ where α is a variety (say over $\{1, \ldots, n\}$: here, the specific places used by α are irrelevant) become

$$\begin{aligned} |X| &\supseteq \sum_i |Y_i| \quad (4)\\ X\lceil_{\sum_i |Y_i|} &\subseteq \sum_i \overline{Y_i} + \sum_{i \neq j} |Y_i|Y_j + \sum_{\alpha(i,j,k)} |Y_i||Y_j||Y_k| \quad (5) \end{aligned}$$

In the case of non-commutative logic, it is not possible, as in commutative and cyclic logic, to express the Resolution procedure as rewrite rules directly on constraint systems of the initial form above. Instead, the rewrite rules describe transformations on multisets of "infons" which are constraints of a more general form. Apart from constraints of the initial form, there are three types of infons, expressing constraints on, respectively, places, support sets and ordering. The place infons allow to instantiate place variables with corresponding places. They are the exact counterpart of the matching of a right-hand side atom into the left-hand side in the commutative and cyclic case. They trigger the unification of the corresponding type formulas.

The resolution procedure propagates the support set infons exactly as in the commutative and cyclic cases, i.e., downwards in the Simplification procedure and upwards in the Generation procedure. The ordering infons are also propagated but in the opposite directions.
In fact, the rewriting steps in the resolution algorithm act on states consisting of two multisets $\langle \mathcal{C}, \mathcal{H} \rangle$ of infons: the idea is that $\mathcal{H}$ acts as a journal logging the infons obtained so-far, i.e., the initial constraints plus all the infons produced by rewriting steps before the current state on the same branch of the tableau, while $\mathcal{C}$ contains those infons of $\mathcal{H}$ which are available for consumption by the further rewrite rules. Thus, although the multiset $\mathcal{H}$ always grows, its submultiset $\mathcal{C}$ is destructively modified, so that the rewriting procedure eventually terminates. A solution at a state $\langle \mathcal{C}, \mathcal{H} \rangle$ will be an assignment of the variables satisfying all the infons in $\mathcal{H}$ (and hence in $\mathcal{C}$).
We use the following notation to denote the rewrite rules:

$$C \ \textbf{when} \ C_0 \quad \longrightarrow \left| \begin{array}{l} \mapsto \ \cdots \\ \mapsto \ C_i \ \textbf{with} \ \theta_i \\ \mapsto \ \cdots \end{array} \right.$$

where C, C_0, C_i are multisets of infons and θ_i is a renaming of the place variables. Each of the clauses **when**, and **with** can be omitted. The corresponding rewrite rule is

$$\langle \mathcal{C} + C, \mathcal{H} \rangle \quad \longrightarrow \left| \begin{array}{l} \mapsto \ \cdots \\ \mapsto \ \theta_i \langle \mathcal{C} + C_i, \mathcal{H} + C_i \rangle \\ \mapsto \ \cdots \end{array} \right.$$

for any multisets $\mathcal{C}$ and $\mathcal{H}$ such that $\mathcal{H}$ contains each of the infons of C_0, and, if C is empty, $\mathcal{H}$ does not contain any of the C_i (this prevents the rules in that case from applying indefinitely). The operator $+$ stands for multiset union.

4.3 The Simplification procedure

Definition 4.2 (Infon) *The* infons manipulated by the Simplification procedure *are of the following form:*

Place infons:	$[z = u]$
Support set infons:	$[z \in \|X\|]$; $[z \in \|Y\|]$
Ordering infons:	$[X(z_1, z_2, z_3)]$; $[\overline{Y}(z_1, z_2, z_3)]$; $[Y(z_1, z_2)]$

where z, z_1, z_2, z_3 are variables ranging over places, u is a place, X is a main variable (ranging over order varieties) and Y is a side variable (ranging over orders).

Note that, since all the ternary relations we consider are cyclic, we do not distinguish between the infons $[X(z_1, z_2, z_3)]$ and $[X(z_2, z_3, z_1)]$ (and similarly for $\overline{Y}$). Let $\mathcal{E}$ be the constraint system obtained from an abstract proof, the Simplification procedure starts with a single state labelled by the pair $\langle \emptyset, \mathcal{E} \rangle$. It then proceeds by applying the rewrite rules of Figure 3.
Rules **SP** and **SC** propagate support set infons, in essentially the same way as in the commutative or cyclic case, i.e., downwards in the abstract proof tree. Rule **SP** is the only one that has multiple output states. It sends each place z of X either into ω or into Y (it is the essential step which is also present in the commutative and cyclic case). When z is matched with a place u of ω yielding infon $[z = u]$, it is explicitly required that no other place variable be already matched with u. This ensures linearity. When z is send into Y, there are two possibilities.

- Either the infon $[z \in |Y|]$ is generated.
- Or z is identified with another place z' already sent into Y; in that case, to preserve linearity, it is explicitly required that z' do not come from the same X as z, ie. $[z' \in |X|]$ does not hold. This is only possible if the bipole originating the premise constraint $X = \omega * Y$ contained an additive connective &, yielding another premise constraints $X' = \omega' * Y$ sharing the same side variable Y. In that case, z' may come from X' by a previous application of rule **SP** having produced infon $[z' \in |Y|]$.

The same phenomenon occurs in the commutative (and cyclic) case when two constraints

$$\Gamma Y = \Delta a Y_1 \ldots Y_n \qquad \Gamma' Y = \Delta' a' Y'_1 \ldots Y'_m$$

share the same side variable Y. First, in the second constraint, a' can be sent either into Γ' or into Y. In the latter case, Y is identified with $a'Y'$ so that the first constraint becomes $\Gamma a' Y' = \Delta a Y_1 \ldots Y_m$. From that, a can either be sent into Γ or into Y' or matched with a'. These correspond to the three kinds of output states of rule **SP**, except that in the commutative case, the first and third cases do not need to be explicitly distinguished.
Rule **INIT** starts the propagation of support set infons downward by turning each terminal premise constraint $Y = z$ into an infon $[z \in |Y|]$ ready for propagation. Rules **OP*** and **OC*** deal with ordering infons and propagate them upwards, but only

⋆ For each premise constraint $X = \omega * Y$

$$
\textbf{SP:}\quad [z \in |X|] \longrightarrow
\begin{array}{l}
\mapsto \cdots \\
\mapsto [a=b]\ ,\ [z=u] \\
\mapsto \cdots \\
\mapsto [z \in |Y|] \\
\mapsto \cdots \\
\mapsto [a=b] \textbf{ with } z := z' \\
\mapsto \cdots
\end{array}
\begin{array}{l}
\left.\begin{array}{l}\text{one output state for each } u \in |\omega| \\ \text{with typing constraints } z : a\ \ u : b \\ \text{such that } \forall z'\ [z' = u] \notin \mathcal{H}\end{array}\right\} \\
\\
\left.\begin{array}{l}\text{one output state for each } [z' \in |Y|] \in \mathcal{H} \\ \text{with typing constraints } z : a\ \ z' : b \\ \text{such that } [z' \in |X|] \notin \mathcal{H}\end{array}\right\}
\end{array}
$$

OP1: **when** $[z_1 = u_1][z_2 = u_2][z_3 = u_3]$ where $(u_1, u_2, u_3) \in \overline{\omega} \longrightarrow [X(z_1, z_2, z_3)]$
OP2: **when** $[z_1 = u_1][z_2 = u_2][z_3 \in |X|][z_3 \in |Y|]$ where $(u_1, u_2) \in \omega \longrightarrow [X(z_1, z_2, z_3)]$
OP3: **when** $[z_1 = u_1][z_2 \in |X|][z_3 \in |X|][Y(z_2, z_3)]$ where $u_1 \in |\omega| \longrightarrow [X(z_1, z_2, z_3)]$
OP4: **when** $[z_1 \in |X|][z_2 \in |X|][z_3 \in |X|][\overline{Y}(z_1, z_2, z_3)] \longrightarrow [X(z_1, z_2, z_3)]$

⋆ For each conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$

SC: $[z \in |Y_i|] \longrightarrow [z \in |X|]$

OC1: $[X(z_1, z_2, z_3)]$ **when** $[z_1 \in |Y_i|][z_2 \in |Y_i|][z_3 \in |Y_i|] \longrightarrow [\overline{Y_i}(z_1, z_2, z_3)]$
OC2: $[X(z_1, z_2, z_3)]$ **when** $[z_1 \in |Y_i|][z_2 \in |Y_i|][z_3 \in |Y_j|]$ where $i \neq j \longrightarrow [Y_i(z_1, z_2)]$
OC3: $[X(z_1, z_2, z_3)]$ **when** $[z_1 \in |Y_i|][z_2 \in |Y_j|][z_3 \in |Y_k|]$ where $i \neq j \neq k \neq i$ and $\neg\alpha(i, j, k)$
$\longrightarrow$ DEADEND

⋆ Structural rules

OV1: **when** $[Y(z_1, z_2)][Y(z_2, z_1)] \longrightarrow$ DEADEND
OV2: **when** $[Y(z_1, z_2)][Y(z_2, z_3)] \longrightarrow [\overline{Y}(z_1, z_2, z_3)]$

⋆ Initialisation: let $\mathcal{C}$ be the multiset of infons of the form $[z \in |Y|]$ for each terminal premise constraint $Y = z$.

INIT: $\longrightarrow \mathcal{C}$

Figure 3: Simplification rules

when sufficient support set information has been propagated downwards. Finally, rules **OV*** ensure that all the orderings respect the axioms of orders and order varieties (in fact only two of them, since the others are obtained for free, as shown below).

As the astute reader may have guessed, the names of the rules have not been chosen arbitrarily: the first letter **S** or **O** indicates whether the rule deals with support set infons or ordering infons; the second letter **P** or **C** indicates whether the rule pertains to premise or conclusion constraints.

We now have to show that Simplification satisfies all the "good" properties listed in Section 3.1.2.

Theorem 4.3 *Simplification satisfies Termination, Preservation, and produces a finite tableau.*

Demonstration:

- Simplification satisfies Preservation: for example, rule **SP** satisfies Preservation as a direct consequence of (2). All the other cases are treated similarly: rule **SC** is justified by (4), rules **OP*** by (3) and rules **OC*** by (5). The structural rules are trivial.

- Simplification satisfies Termination: the argument is very similar to the commutative case; the rewrite rules propagate information in a uniform direction through the abstract proof tree (towards the root for support set infons, towards the leaves for ordering infons), and hence all propagations must terminate. Note that the rules which produce without consuming infons do not threaten Termination, since they cannot apply if their conclusion has already been produced.

- Simplification produces a finite tableau: obvious, since all the rewrite rules involve only finite choices, and we already have shown Termination.

□

The main difficulty is therefore to show that the open states at the end of the Simplification procedure are consistent. This is the purpose of the rest of this section, where we assume given an open state $\langle ., \mathcal{H}_o \rangle$ obtained at the end of Simplification. Hence, no Simplification rule further applies to it. In the sequel, we simply write $[\ldots]$ to mean that the infon $[\ldots]$ belongs to $\mathcal{H}_o$, i.e., has been produced in the sequence of rewriting steps which led to $\langle ., \mathcal{H}_o \rangle$.

Lemma 4.4 *Let $\mathcal{P} = \{u \mid \exists z\ [z = u]\}$. There exists a unique bijection $u \mapsto \dot{u}$ from $\mathcal{P}$ into a subset of place variables such that for all place u and place variable z*

$$z = \dot{u} \ \Leftrightarrow\ [z = u]$$

Demonstration: This lemma essentially states that the graph of the relation $[z = u]$ is a bijection (from its domain into its codomain). In other words, we have to show that:

- If $[z = u]$ and $[z' = u]$ then $z = z'$: this results from the fact that when a place variable is matched to a place, it is explicitly required in rule **SP** that the latter be not already matched by another place variable.
- If $[z = u]$ and $[z = u']$ then $u = u'$: this results from the fact that, by construction of the Simplification procedure, a place variable is never matched more than once since, once it is matched, the propagation of its support set information is stopped. Special care has to be taken since distinct place variables can be identified during the Simplification procedure (3rd group of output states in rule **SP**), so it may a priori happen that $[z = u]$ and $[z' = u']$ be produced and then later z and z' be identified. But that never happens because it would require that support set information about z and z' be propagated further down until the identification step, whereas no support set infon concerning z or z' is propagated after they are matched to u and u'.

□

Any structure R on a subset of $\mathcal{P}$ (places), can be transported into a structure (abusively written) $\dot{R}$ on place variables via the bijection $u \mapsto \dot{u}$. Note that if ω is any order on places, we have (by simple transport of structure):

$$\overline{\omega\dot{\restriction}_{\mathcal{P}}} = \overline{\omega}\dot{\restriction}_{\mathcal{P}}$$

Definition 4.5 *Let X be a main variable and Y a side variable. We define the sets of places $\mathcal{D}_X, \mathcal{D}_Y$ as well as the relations $\dot{X}$, $\dot{Y}$ and $\ddot{Y}$ of arity, respectively, 3, 2 and 3, on place variables as follows:*

$$\begin{array}{lll}
\mathcal{D}_X = \{z \mid [z \in |X|]\} & |\dot{X}| = \mathcal{D}_X & \dot{X}(z_1, z_2, z_3) \ \Leftrightarrow\ [X(z_1, z_2, z_3)] \\
\mathcal{D}_Y = \{z \mid [z \in |Y|]\} & |\dot{Y}| = \mathcal{D}_Y & \dot{Y}(z_1, z_2) \ \Leftrightarrow\ [Y(z_1, z_2)] \\
 & |\ddot{Y}| = \mathcal{D}_Y & \ddot{Y}(z_1, z_2, z_3) \ \Leftrightarrow\ [\overline{Y}(z_1, z_2, z_3)]
\end{array}$$

Lemma 4.6 *For any main (resp. side) variable X (resp. Y), the relation $\dot{X}$ (resp.$\dot{Y}$) is a series-parallel order variety (resp. order). Furthermore:*

- *For any conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$ we have $\dot{X} \subseteq \alpha(\dot{Y}_1, \ldots, \dot{Y}_n)$.*
- *For any premise constraint $X = \omega * Y$ we have $\dot{X} = \omega\dot{\restriction}_{\mathcal{P}} * \dot{Y}\restriction_{\mathcal{D}_X}$*
- *For any side variable Y we have $\overline{\dot{Y}} = \ddot{Y}$.*

Demonstration: In fact, the rewrite rules of the Simplification phase have been exactly designed to obtain this lemma. Let $\mathcal{X}$ (resp. $\mathcal{Y}$) be the set of main (resp. side) variables X (resp. Y) such that $\dot{X}$ is a series-parallel order variety (resp. $\dot{Y}$ is a series-parallel order and $\overline{\dot{Y}} = \ddot{Y}$). All we need to show is that:

P^0: $X_o \in \mathcal{X}$ where X_o is the main variable at the root of the abstract proof.

P$^+$: If $X \in \mathcal{X}$ then for any conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$ we have $Y_i \in \mathcal{Y}$ and $\dot{X} \subseteq \alpha(\dot{Y}_1, \ldots, \dot{Y}_n)$.

P$^-$: If $Y \in \mathcal{Y}$ then for any premise constraint $X = \omega * Y$ we have $X \in \mathcal{X}$ and $\dot{X} = \omega\dot{\restriction}_{\mathcal{P}} * \dot{Y}\restriction_{\mathcal{D}_X}$.

Indeed, by repeatedly applying these properties upward in the proof, we obtain that $X \in \mathcal{X}$ (resp. $Y \in \mathcal{Y}$) for of all the main (resp. side) variables X (resp. Y), and hence Lemma 4.6.

P^0: Observe that, since Simplification propagates ordering infons upward, no such infon can be obtained for the main variable X_o at the root of the abstract tree, and hence, $\dot{X}_o$ is the empty order variety on $\mathcal{D}_{X_o}$, which is series-parallel. Hence $X_o \in \mathcal{X}$.

P$^+$: Now, let $X \Subset \alpha(Y_1, \ldots, Y_n)$ be a conclusion constraint and assume $X \in \mathcal{X}$, ie. $\dot{X}$ is a series-parallel order variety. We must show that $Y_i \in \mathcal{Y}$ and $\dot{X} \subseteq \alpha(\dot{Y}_1, \ldots, \dot{Y}_n)$.

1. Let's first show that $\mathcal{D}_{Y_i} \cap \mathcal{D}_{Y_j} = \emptyset$ for $i \neq j$. Reason by contradiction, and assume $z \in \mathcal{D}_{Y_i} \cap \mathcal{D}_{Y_j}$. Hence $[z \in |Y_i|]$ and $[z \in |Y_j|]$ which is only possible if $[z \in |Y_i|]$ and $[z' \in |Y_j|]$ were obtained at some point and z' was identified to z by application of rule **SP** at some node X' further down in the proof with constraint $X' = \omega' * Y'$. Hence, just before the identification, we must be in the following situation:

$$\begin{array}{ccccc}
[z \in |Y_i|] & & & & [z' \in |Y_j|] \\
\mathbf{SC} \searrow & & & & \swarrow \mathbf{SC} \\
& [z \in |X|] & & [z' \in |X|] & \\
& \vdots\downarrow & & \downarrow\vdots & \\
& [z \in |X'|] & & [z' \in |X'|] & \\
& \mathbf{SP}\downarrow & & \boxed{\begin{array}{c}\downarrow \mathbf{SP} \\ \{z' := z\}\end{array}} & ? \\
& [z \in |Y'|] & & &
\end{array}$$

   Now the identification $\{z' := z\}$ (framed on the figure) applies rule **SP** from $[z' \in |X'|]$ when $[z \in |Y'|]$, but explicitly requires that $[z \in |X'|]$ be not produced, which is impossible since it is needed to produce $[z \in |Y'|]$. Contradiction.

2. The set $\mathcal{D}_{Y_i}$ is a singleton for at least one i. Indeed, recall that the conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$ was obtained from the positive layer of a bipole, which contains at least one positive atom, which in turns yields a terminal premise constraint $Y_i = z$ for some $i = 1, \ldots, n$. By rule **INIT**, we get $[z \in |Y_i|]$ hence $z \in \mathcal{D}_{Y_i}$. Since propagation of support set goes downward, no other infon $[. \in |Y_i|]$ can be produced, so that $\mathcal{D}_{Y_i} = \{z\}$.

3. Let $z \in |\dot{X}|$. Hence we have $[z \in |X|]$, which must have been obtained by rule **SC** from $[z \in |Y_i|]$ for some i, hence $z \in \mathcal{D}_{Y_i}$. Thus, $|\dot{X}| \subseteq \sum_i \mathcal{D}_{Y_i}$. The converse inclusion is shown in a similar way.

4. Consequently, $(\mathcal{D}_{Y_i})_{i=1,\ldots,n}, \dot{X}, \alpha$ forms a splitting problem as defined in Section 2.4. Let's show that admissibility holds for that problem.
   - Let $z_1 \in \mathcal{D}_{Y_i}$, $z_2 \in \mathcal{D}_{Y_j}$ and $z_3 \in \mathcal{D}_{Y_k}$ with $i \neq j \neq k \neq i$ and assume $\dot{X}(z_1, z_2, z_3)$. Hence $[X(z_1, z_2, z_3)]$. Hence necessarily $\alpha(i, j, k)$, otherwise rule **OC3** would have lead to a deadend.
   - Let $z_1, z_2 \in \mathcal{D}_{Y_i}$, $z \in \mathcal{D}_{Y_j}$ and $z' \in \mathcal{D}_{Y_k}$ with $i \neq j, k$ and assume $\dot{X}(z_1, z_2, z)$ and $\dot{X}(z_2, z_1, z')$. Hence $[X(z_1, z_2, z)]$ and $[X(z_2, z_1, z')]$. Hence, by rule **OC2** we have $[Y_i(z_1, z_2)]$ and $[Y_i(z_2, z_1)]$. This is impossible, because rule **OV1** would have lead to a deadend.

5. Hence, by Theorem 2.1, we know that τ_i defined by

$$\tau_i = \bigcup_{j \neq i, z \in \mathcal{D}_{Y_j}} \dot{X}_z \restriction_{\mathcal{D}_{Y_i}}$$

   is a series-parallel order for each $i = 1, \ldots, n$, and that furthermore,

$$\dot{X} \subseteq \alpha(\tau_1, \ldots, \tau_n)$$

   It is easy to show, using rule **OC2** and the definition of τ_i above, that τ_i is exactly $\dot{Y}_i$ for all i. Hence $\dot{Y}_i$ is a series-parallel order for all i and

$$\dot{X} \subseteq \alpha(\dot{Y}_1, \ldots, \dot{Y}_n)$$

6. Let's show that $\ddot{Y}_i \subseteq \overline{\dot{Y}_i}$. Assume $\ddot{Y}_i(z_1, z_2, z_3)$. Hence $[\overline{Y_i}(z_1, z_2, z_3)]$ which must have been obtained in either of two ways:
   - By application of rule **OV2**, hence we have $[Y_i(z_1, z_2)]$ and $[Y_i(z_2, z_3)]$. Hence $\dot{Y}_i(z_1, z_2)$ and $\dot{Y}_i(z_2, z_3)$. Hence, $\overline{\dot{Y}_i}(z_1, z_2, z_3)$.
   - By application of rule **OC1**, hence we have $[X(z_1, z_2, z_3)]$. Hence $\dot{X}(z_1, z_2, z_3)$. Hence $\overline{\dot{Y}_i}(z_1, z_2, z_3)$ since $\dot{X} \subseteq \alpha(\dot{Y}_1, \ldots, \dot{Y}_n)$.

   Hence in both cases we have proved $\overline{\dot{Y}_i}(z_1, z_2, z_3)$.

7. Conversely, let's show that $\overline{\dot{Y}_i} \subseteq \ddot{Y}_i$. Assume $\overline{\dot{Y}_i}(z_1, z_2, z_3)$. We can assume, without loss of generality, that we are in the case where $A : \dot{Y}_i(z_1, z_2)$ and $B : \dot{Y}_i(z_1, z_2 | z_3)$. From A, we get $[Y_i(z_1, z_2)]$ which must have been obtained by rule **OC2** from some $[X(z_1, z_2, z)]$ where $[z \in |Y_j|]$ for $j \neq i$. Hence $z \in \mathcal{D}_X$ and $\dot{X}(z_1, z_2, z)$. Let's apply the spreading property of order varieties to this cycle with z_3. There are three possibilities:

– $\dot{X}(z_3, z_2, z)$, hence $[X(z_3, z_2, z)]$ and by rule **OC2**, we have $[Y_i(z_3, z_2)]$, hence $\dot{Y}_i(z_3, z_2)$. Hence, by B we get that $\dot{Y}_i(z_3, z_1)$ and $[Y_i(z_3, z_1)]$. By application of rule **OV2** to this and A, we get that $[\overline{Y_i}(z_1, z_2, z_3)]$.
– $\dot{X}(z_1, z_3, z)$, hence $[X(z_1, z_3, z)]$ and by rule **OC2**, we have $[Y_i(z_1, z_3)]$, hence $\dot{Y}_i(z_1, z_3)$. Hence, by B we get that $\dot{Y}_i(z_2, z_3)$ and $[Y_i(z_2, z_3)]$. By application of rule **OV2** to this and A, we get that $[\overline{Y_i}(z_1, z_2, z_3)]$.
– $\dot{X}(z_1, z_2, z_3)$, hence $[X(z_1, z_2, z_3)]$ and by rule **OC1**, we have $[\overline{Y_i}(z_1, z_2, z_3)]$.

Hence, in all cases we have proved $[\overline{Y_i}(z_1, z_2, z_3)]$ hence $\ddot{Y}_i(z_1, z_2, z_3)$.

Therefore, we have shown that $\dot{Y}_i$ is a series-parallel order satisfying $\overline{\dot{Y}_i} = \ddot{Y}_i$ for each i, and that

$$\dot{X} \subseteq \alpha(\dot{Y}_1, \ldots, \dot{Y}_n)$$

P$^-$: Finally, let $X = \omega * Y$ be a premise constraint and assume $Y \in \mathcal{Y}$, ie. $\dot{Y}$ is a series-parallel order satisfying $\overline{\dot{Y}} = \ddot{Y}$. We have to show that $X \in \mathcal{X}$ and $\dot{X} = \omega\dot{\upharpoonright}_{\mathcal{P}} * \dot{Y}\upharpoonright_{\mathcal{D}_X}$.

1. Let's first show that $|\omega\dot{\upharpoonright}_{\mathcal{P}}| \cap |\dot{Y}\upharpoonright_{\mathcal{D}_X}| = \emptyset$. Reason by contradiction and assume $z \in |\omega\dot{\upharpoonright}_{\mathcal{P}}| \cap |\dot{Y}\upharpoonright_{\mathcal{D}_X}|$. From $z \in |\omega\dot{\upharpoonright}_{\mathcal{P}}|$ we get $z = \dot{u}$ for some $u \in |\omega\upharpoonright_{\mathcal{P}}|$, hence $[z = u]$ for $u \in |\omega|$. From $z \in |\dot{Y}\upharpoonright_{\mathcal{D}_X}|$ we have $[z \in |Y|]$ and $[z \in |X|]$, which means that $[z \in |X|]$ has been propagated by rule **SP** at X into $[z \in |X|]$. But in that case, $[z = u]$ cannot have been produced: it can only be produced in the other alternatives of rule **SP**. Contradiction.
2. Let $z \in \mathcal{D}_X$ hence $[z \in |X|]$. By application of rule **SP**, we get either $[z = u]$ with $u \in |\omega|$, in which case $z = \dot{u} \in |\omega\dot{\upharpoonright}_{\mathcal{P}}|$, or $[z \in |Y|]$ hence $z \in |\dot{Y}|$ hence $z \in |\dot{Y}\upharpoonright_{\mathcal{D}_X}|$. Thus, $\mathcal{D}_X \subseteq |\omega\dot{\upharpoonright}_{\mathcal{P}}| \cup |\dot{Y}\upharpoonright_{\mathcal{D}_X}|$. The converse inclusion is shown in a similar way.
3. Let $z_1, z_2, z_3 \in \mathcal{D}_X$ such that $(\omega\dot{\upharpoonright}_{\mathcal{P}} * \dot{Y})(z_1, z_2, z_3)$. Hence we have one of the following cases:
   – $[z_1 = u_1]$ and $[z_2 = u_2]$ and $[z_3 = u_3]$ with $u_1, u_2, u_3 \in |\omega|$ and $\overline{\omega}(u_1, u_2, u_3)$ and we are in the conditions of application of rule **OP1**.
   – $[z_1 = u_1]$ and $[z_2 = u_2]$ and $[z_3 \in |Y|]$ with $u_1, u_2 \in |\omega|$ and $\omega(u_1, u_2)$ and we are in the conditions of application of rule **OP2**.
   – $[z_1 = u_1]$ and $[z_2 \in |Y|]$ and $[z_3 \in |Y|]$ with $u_1 \in |\omega|$ and $\dot{Y}(z_2, z_3)$. Hence $[Y(z_1, z_2)]$ and we are in the conditions of application of rule **OP3**.
   – $[z_1 \in |Y|]$ and $[z_2 \in |Y|]$ and $[z_3 \in |Y|]$ with $\overline{\dot{Y}}(z_1, z_2, z_3)$. Hence $\ddot{Y}(z_1, z_2, z_3)$ — this is the essential step — hence $[\overline{Y}(z_1, z_2, z_3)]$ and we are in the conditions of application of rule **OP4**.

   Thus, in all cases, by application of one of the rule **OP**, we get $[X(z_1, z_2, z_3)]$, hence $\dot{X}(z_1, z_2, z_3)$. Thus, we have shown, $\omega\dot{\upharpoonright}_{\mathcal{P}} * \dot{Y}\upharpoonright_{\mathcal{D}_X} \subseteq \dot{X}$. The converse inclusion is shown in a similar way.

Therefore, we have shown that

$$\dot{X} = \omega\dot{\upharpoonright}_{\mathcal{P}} * \dot{Y}\upharpoonright_{\mathcal{D}_X}$$

and hence, since $\dot{Y}$ and ω are series-parallel orders, $\dot{X}$ is a series-parallel order variety.

□

Definition 4.7 *A* safe *place assignment is an injective mapping ϕ from place variables into places such that*

$$\forall u \in \mathcal{P} \;\; \phi(\dot{u}) = u$$

Any structure R on place variables, can be transported into a structure (abusively written) R^ϕ on places via the injection ϕ. We can now complete the study of Simplification.

Theorem 4.8 *The open nodes obtained at the end of the Simplification procedure are consistent.*

Demonstration: First let ϕ be a safe place assignment. Now consider the following assignment $\hat{\phi}$ for all the variables:

- For each place variable z, let $\hat{\phi}(z) = z^\phi$.
- For each side variable Y, let $\hat{\phi}(Y) = \dot{Y}^\phi$.
- For the main variable X_o at the root, let $\hat{\phi}(X_o)$ be the empty order variety with support set $\mathcal{D}^\phi_{X_o}$.
- For each main variable other than X_o, hence attached to a unique premise constraint $X = \omega * Y$, let $\hat{\phi}(X) = \omega * \dot{Y}^\phi$.

By Lemma 4.6, it is easy to show that $\hat{\phi}(X)$ is an order variety and $\hat{\phi}(Y)$ is an order, both series-parallel.

- For the main variable X_o at the root, we have that $\hat{\phi}(X_o)\restriction_{\mathcal{D}^\phi_{X_o}}$ is the empty order variety on support set $\mathcal{D}^\phi_{X_o}$, which is equal to the order variety $\dot{X_o}^\phi$, with the same support set and also empty. Hence $\hat{\phi}(X_o)\restriction_{\mathcal{D}^\phi_{X_o}} = \dot{X_o}^\phi$.
- For each premise constraint $X = \omega * Y$, we have, trivially,

$$\hat{\phi}(X) = \omega * \dot{Y}^\phi = \omega * \hat{\phi}(Y)$$

but also (using Lemma 4.6):

$$\hat{\phi}(X)\restriction_{\mathcal{D}^\phi_X} = \omega\restriction_{\mathcal{D}^\phi_X} * \dot{Y}^\phi\restriction_{\mathcal{D}^\phi_X} = \omega\restriction_{\mathcal{P}} *(\dot{Y}\restriction_{\mathcal{D}_X})^\phi = \dot{\omega\restriction_{\mathcal{P}}}^\phi * (\dot{Y}\restriction_{\mathcal{D}_X})^\phi = (\dot{\omega\restriction_{\mathcal{P}}} * \dot{Y}\restriction_{\mathcal{D}_X})^\phi = \dot{X}^\phi$$

because, since ϕ is safe, $\mathcal{D}^\phi_X$ coincides with $\mathcal{P}$ on $|\omega|$, and for any structure R on a subset of $\mathcal{P}$, we obviously have $R = \dot{R}^\phi$ (transportation of structure forward and backward).
- For each conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$, we have (using Lemma 4.6):

$$\hat{\phi}(X) \Subset \hat{\phi}(X)\restriction_{\mathcal{D}^\phi_X} = \dot{X}^\phi \subseteq \alpha(\dot{Y_1}, \ldots, \dot{Y_n})^\phi = \alpha(\dot{Y_1}^\phi, \ldots, \dot{Y_n}^\phi) = \alpha(\hat{\phi}(Y_1), \ldots, \hat{\phi}(Y_n))$$

Note here the first step, which discards, by Weakening, all the places of $\hat{\phi}(X)$ which are not in $\mathcal{D}^\phi_X$. It would have been very difficult to account for these places in the Simplification procedure, because checking that they can be consistently ordered would have required uncontrolled non-deterministic choices to simply assign them among $|\hat{\phi}(Y_1)|, \ldots, |\hat{\phi}(Y_n)|$, and recursively upward (while the places of $\mathcal{D}^\phi_X$ are assigned by rules **SP** and **SC**). Even if we allow these non-deterministic choices, the same problem will re-appear in the Generation procedure, with places which are not even known in the Simplification procedure.

Therefore, $\hat{\phi}$ satisfies all the initial constraints in $\mathcal{H}_o$. It also trivially satisfies all the other infons in that state. For example, let $[z \in |Y|]$ belong to $\mathcal{H}_o$. Hence $z \in \mathcal{D}_Y$, hence $\hat{\phi}(z) = z^\phi \in \mathcal{D}^\phi_Y = |\hat{\phi}(Y)|$. Hence $\hat{\phi}$ satisfies the infon $[z \in |Y|]$. □

Theorems 4.3 and 4.8 thus show all the expected properties of Simplification stated in Section 3.1.2.
We have shown that $\hat{\phi}$ is a solution at open state $\langle ., \mathcal{H}_o\rangle$, but we can be more precise. Simplification computes, at its open states, the minimal solution:

Corollary 4.9 *Let $\langle ., \mathcal{H}_o\rangle$ be an open state at the end of the Simplification procedure and σ be a solution of $\mathcal{H}_o$, then $\sigma = \hat{\phi}$ on the place variables of $\dot{\mathcal{P}}$, and can be made equal on the others. Furthermore, for all variables X, Y we have*

$$\begin{array}{ll} |\hat{\phi}(Y)| = \mathcal{D}^\phi_Y \subseteq |\sigma(Y)| & |\hat{\phi}(X)| = \mathcal{D}^\phi_X \subseteq |\sigma(X)| \\ \hat{\phi}(Y) \trianglelefteq \sigma(Y)\restriction_{\mathcal{D}^\phi_Y} & \hat{\phi}(X) \subseteq \sigma(X)\restriction_{\mathcal{D}^\phi_X} \end{array}$$

Demonstration: Let σ be a solution of $\mathcal{H}_o$. For each place variable $z \in \dot{\mathcal{P}}$, we have in $\mathcal{H}_o$ an infon $[z = u]$ for some $u \in \mathcal{P}$, which is satisfied by σ, so that $\sigma(z) = u = \hat{\phi}(z)$. The other place variables are assigned places which do not occur in $\mathcal{H}_o$, and which can be renamed so that $\hat{\phi}, \sigma$ coincide on all place variables. Now, let Y be a side variable and $u \in |\hat{\phi}(Y)| = \mathcal{D}^\phi_Y$. Hence $u = z^\phi$ for some $z \in \mathcal{D}_Y$. Since ϕ is safe, we have $\dot{u}^\phi = u = z^\phi$ hence $z = \dot{u}$ hence $[z = u] \in \mathcal{H}_o$. Since σ is a solution of $\mathcal{H}_o$, we have $\sigma(z) = u$. Now, from $z \in \mathcal{D}_Y$ we get $[z \in |Y|] \in \mathcal{H}_o$. Again, since σ is a solution of $\mathcal{H}_o$, we have $\sigma(z) \in |\sigma(Y)|$. Hence, $u \in |\sigma(Y)|$. Therefore, we have proved $|\hat{\phi}(Y)| \subseteq |\sigma(Y)|$. The other statements (on the main variables, and on ordering) are proved similarly. □

4.4 The Generation procedure

The Generation procedure produces all the solutions by extending the minimal solutions computed by the Simplification procedure. Let $\langle ., \mathcal{H}_o\rangle$ be a given open (hence consistent) node at the end of the Simplification procedure. The Generation procedure first chooses an arbitrary safe assignment ϕ for the place variables. It then generates all the solutions which coincide with ϕ on place variables. In the sequel, recall that ϕ is therefore fixed. Note that there is no need to consider non-deterministic alternatives for the choice of ϕ: they would lead to the same solutions, modulo renaming of places.
The Generation procedure starts from the state $\langle \emptyset, \mathcal{H}_o\rangle$ then proceeds by applying the rewrite rules of Figure 4. These rules manipulate infons of the following forms:

Definition 4.10 (Infon) *The* infons manipulated by the Generation procedure *are of the following forms:*

$$\begin{array}{lll} \textit{Partial infons} & [X\restriction_{\mathcal{D}_X} \subseteq \beta] \;\; ; \;\; [Y\restriction_{\mathcal{D}_Y} \trianglelefteq \tau] \\ \textit{Total infons} & [X = \beta] \;\; ; \;\; [Y = \tau] \end{array}$$

$\star$ For each side variable Y, let $\{X_j = \omega_j * Y\}_{j=1}^m$ be all the premise constraints involving Y,

$$\textbf{PP:}\quad [X_j\restriction_{\mathcal{D}_{X_j}} \subseteq \beta_j]_{j=1}^m \quad \longrightarrow \left|\begin{array}{l}\mapsto\ \ldots \\ \mapsto\ [Y\restriction_{\mathcal{D}_Y} \trianglelefteq \tau] \\ \mapsto\ \ldots\end{array}\right. \left.\vphantom{\begin{array}{l}a\\a\\a\end{array}}\right\} \begin{array}{l}\text{one output state for each } \tau \\ \text{such that } \dot{Y}^\phi \trianglelefteq \tau \\ \text{and } (\omega_j * \tau)\restriction_{\mathcal{D}^\phi_{X_j}} \subseteq \beta_j \text{ for all } j = 1, \ldots, m\end{array}$$

$$\textbf{TP:}\quad [Y = \tau'] \quad \longrightarrow \quad [X_j = \omega_j * \tau']_{j=1}^m$$

$\star$ For each conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$,

$$\textbf{PC:}\quad [Y_i\restriction_{\mathcal{D}_{Y_i}} \trianglelefteq \tau_i]_{i=1}^n \quad \longrightarrow \quad [X\restriction_{\mathcal{D}_X} \subseteq \alpha(\tau_1, \ldots, \tau_n)]$$

$$\textbf{TC:}\quad [X = \beta] \textbf{ when } [Y_i\restriction_{\mathcal{D}_{Y_i}} \trianglelefteq \tau_i]_{i=1}^n \quad \longrightarrow \left|\begin{array}{l}\mapsto\ \ldots \\ \mapsto\ [Y_i = \tau'_i]_{i=1}^n \\ \mapsto\ \ldots\end{array}\right. \left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \begin{array}{l}\text{one output state for each } n\text{-uple } \tau'_1, \ldots, \tau'_n \\ \text{such that } \beta \Subset \alpha(\tau'_1, \ldots, \tau'_n) \text{ and } \forall i \\ \tau'_i = z^\phi \text{ if } Y_i = z \text{ is a terminal constraint} \\ \dot{Y_i}^\phi \trianglelefteq \tau'_i\restriction_{\mathcal{D}^\phi_{Y_i}} \trianglelefteq \tau_i \text{ otherwise}\end{array}$$

$\star$ Initialisations: let $\mathcal{C}$ be the multiset of infons of the form $[Y\restriction_{\mathcal{D}_Y} \trianglelefteq z^\phi]$ for each terminal premise constraint $Y = z$ and $[X\restriction_{\mathcal{D}_X} \subseteq \mathcal{D}^\phi_X \times \mathcal{D}^\phi_X \times \mathcal{D}^\phi_X]$ for each main variable X which occurs in no conclusion constraint; let X_o be the main variable at the root.

$$\textbf{P-INIT:}\quad \longrightarrow \quad \mathcal{C}$$

$$\textbf{T-INIT:}\quad [X_o\restriction_{\mathcal{D}_{X_o}} \subseteq \beta] \quad \longrightarrow \left|\begin{array}{l}\mapsto\ \ldots \\ \mapsto\ [X_o = \beta'] \\ \mapsto\ \ldots\end{array}\right. \left.\vphantom{\begin{array}{l}a\\a\\a\end{array}}\right\} \begin{array}{l}\text{one output state for each } \beta' \\ \text{such that } \beta'\restriction_{\mathcal{D}^\phi_{X_o}} \subseteq \beta\end{array}$$

Figure 4: Generation rules

where τ is an order and β is a ternary relation on places (always an order variety, except in Rule **P-INIT***).*

Rules **PP** and **PC** propagate partial infons initially produced by **P-INIT** downwards in the abstract proof tree while rules **TP** and **TC** propagate total infons initially produced by **T-INIT** upwards, i.e., in both cases, in the opposite direction of the Simplification procedure (just as in the commutative or cyclic case). More precisely, Generation proceeds in two phases:

- First, rule **P-INIT** is applied and all the propagations by rules **PP** and **PC** are performed. At the end of this phase, a partial infon $[X_o\restriction_{\mathcal{D}_{X_o}} \subseteq \beta]$ is produced at the root.
- Then, this infon triggers rule **T-INIT** followed by all the propagations by rules **TP** and **TC**. At the end of this phase, there is an infon $[X = \beta]$ and an infon $[Y = \tau]$ for each main (resp. side) variable X (resp. Y). These total infons entirely define the generated solution.

Theorem 4.11 *Generation satisfies Resolution, Termination and Preservation.*

Demonstration:

- Termination and Resolution are straightforward: the propagations are always performed in a uniform direction in the abstract proof tree, which is finite, so they must terminate. Furthermore, at the end (of the Generation), there is a total infon $[X = \beta]$ for each main variable X, with no two such constraints sharing their left-hand side, so the state is in fully solved form.
- Preservation: let's show it here for rule **PP**. The other rules are treated in the same way. Since the application of a rule does not decrease the set of infons, it is obvious that a solution of the right-hand side of a rule is also a solution of its left-hand side, hence we only need to prove the converse. Let σ be a solution of the input state of rule **PP** which coincides with ϕ on place variables, and let $\tau = \sigma(Y)\restriction_{\mathcal{D}^\phi_Y}$. Then by construction σ is a solution for the infon $[Y\restriction_{\mathcal{D}_Y} \trianglelefteq \tau]$, hence of the output state of the rule corresponding to that choice of τ. We just have to show that this choice is possible, i.e., that $\dot{Y}^\phi \trianglelefteq \tau$ and $(\omega_j * \tau)\restriction_{\mathcal{D}^\phi_{X_j}} \subseteq \beta_j$ for all $j = 1, \ldots, m$. The former is a direct consequence

of Lemma 4.9. For the latter, observe that σ satisfies the constraint $X_j = \omega_j * Y$, hence

$$\sigma(X_j)\lceil_{\mathcal{D}^\phi_{X_j}} = (\omega_j * \sigma(Y))\lceil_{\mathcal{D}^\phi_{X_j}} = (\omega_j * \tau)\lceil_{\mathcal{D}^\phi_{X_j}}$$

since $\mathcal{D}^\phi_{X_j}$ and $\mathcal{D}^\phi_Y$ coincide outside $|\omega_j|$. But σ also satisfies the infon $[X_j\lceil_{\mathcal{D}^\phi_{X_j}} \subseteq \beta_j]$, hence $(\omega_j * \tau)\lceil_{\mathcal{D}^\phi_{X_j}} \subseteq \beta_j$.

□

The main difficulty is therefore to show that all the states produced by the Generation procedure are consistent. This is the purpose of the rest of this section. Recall that Generation consists of two phases performed sequentially: propagations downwards by **P-INIT**, **PP** and **PC**; propagations upwards by **T-INIT**, **TP** and **TC**.
The first phase is trivially consistent, since, in any state, it is possible to choose the minimal solution given by $\hat{\phi}$. From now on, consider a state $\langle ., \mathcal{H}_1\rangle$ at the end of the first phase, having produced infon $[X_o\lceil_{\mathcal{D}_{X_o}} \subseteq \beta_o]$ at the root. We know that $\hat{\phi}$ is a solution at that state. We are going to build another solution, which is the entropy-maximal solution consistent with the choices made to obtain that state. Consider the variable assignment σ_1 defined as follows:

- For each place variable z, let $\sigma_1(z) = z^\phi$.
- For each side variable Y, it is easy to see that in the first phase of Generation, rule **PP** fired exactly once for Y, producing exactly one infon $[Y\lceil_{\mathcal{D}_Y} \trianglelefteq \tau]$. Let $\sigma_1(Y) = \tau$.
- For the root X_o, let $\sigma_1(X_o) = \beta_o$.
- For each main variable X occurring in a premise constraint $X = \omega * Y$, let $\sigma_1(X) = \omega * \sigma_1(Y)$.

Lemma 4.12 *σ_1 is the entropy-maximal solution at state $\langle ., \mathcal{H}_1\rangle$, i.e., it is a solution and for any solution σ,*

$$\sigma(Y)\lceil_{\mathcal{D}^\phi_Y} \trianglelefteq \sigma_1(Y)$$

Demonstration:

- By construction, σ_1 satisfies all the infons of the form $[Y\lceil_{\mathcal{D}_Y} \trianglelefteq \tau]$ where Y is a side variable (in fact, the two terms are even equal rather than in the entropy relation).
- Also by construction, σ_1 satisfies all the premise constraints of the form $X = \omega * Y$. In that case, we know that an infon $[Y\lceil_{\mathcal{D}_Y} \trianglelefteq \tau]$ has been produced by rule **PP**, hence $\sigma_1(Y) = \tau$ and $\sigma_1(X) = \omega * \tau$. We also know that $(\omega * \tau)\lceil_{\mathcal{D}^\phi_X} \subseteq \beta$ for some β such that the infon $[X\lceil_{\mathcal{D}_X} \subseteq \beta]$ has also been produced. Hence $\sigma_1(X)\lceil_{\mathcal{D}^\phi_X} \subseteq \beta$, and σ_1 satisfies the infon $[X\lceil_{\mathcal{D}_X} \subseteq \beta]$.
- Finally, if X is also involved in a conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$, then the infon $[X\lceil_{\mathcal{D}_X} \subseteq \beta]$ has been produced by rule **PC**, hence we know that $\beta = \alpha(\tau_1, \ldots, \tau_n)$, for some $\tau_1, \ldots, \tau_n$ such that the infons $[Y_i\lceil_{\mathcal{D}_{Y_i}} \trianglelefteq \tau_i]$ have also been produced. Hence $\sigma_1(Y_i) = \tau_i$ and

  $$\sigma_1(X) \Subset \sigma_1(X)\lceil_{\mathcal{D}^\phi_X} \subseteq \beta = \alpha(\tau_1, \ldots, \tau_n) = \alpha(\sigma_1(Y_1), \ldots, \sigma_1(Y_n))$$

  Hence, σ_1 also satisfies the conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$.
- Note that σ_1 also satisfies all the infons produced in the Simplification procedure, because of the condition $\dot{Y} \trianglelefteq \tau$ in rule **PP**.
- σ_1 is by construction entropy-maximal, since $\sigma_1(Y) = \tau$ implies that the infon $[Y\lceil_{\mathcal{D}_Y} \trianglelefteq \tau]$ has been produced, hence any solution σ must satisfy it, hence $\sigma(Y)\lceil_{\mathcal{D}^\phi_Y} \trianglelefteq \tau = \sigma_1(Y)$.

□

Theorem 4.13 *The Generation procedure produces no inconsistent state.*

Demonstration: We have already seen that the first phase of the procedure does not produce any inconsistencies. Now consider the second phase of the Generation procedure, and let's show that it produces no inconsistent state either. The only difficulty is with rule **TC**, in the case of a conclusion constraint $X \Subset \alpha(Y_1, \ldots, Y_n)$. Assume we have a solution σ for the input state of that rewrite rule. In particular, σ satisfies $[X = \beta]$ and $[Y_i\lceil_{\mathcal{D}_{Y_i}} \trianglelefteq \tau_i]_{i=1}^n$. Choose arbitrarily one of its output states, corresponding to a choice of $\tau'_1, \ldots, \tau'_n$ such that $\beta \Subset \alpha(\tau'_1, \ldots, \tau'_n)$ and $\dot{Y_i}^\phi \trianglelefteq \tau'_i\lceil_{\mathcal{D}^\phi_{Y_i}} \trianglelefteq \tau_i$. Consider the variable assignment σ' defined on side variables as follows (and extended to the main variables using the premise constraints):

- $\sigma'(Y) = \sigma(Y)$ for any side variable Y attached to an inference below the conclusion X.
- $\sigma'(Y_i) = \tau'_i$ for $i = 1, \ldots, n$.
- $\sigma'(Y) = \sigma_1(Y)$ in all the other cases.

By considering each case, we show that σ' is a solution of the constraint system of the selected output state of the rewrite rule.

- Below X, we use the fact that σ is a solution of the input state.
- At the conclusion constraint X, we use the hypothesis on τ'_i:

$$\begin{array}{l}\sigma'(X) = \omega * \sigma'(Y) = \omega * \sigma(Y) = \sigma(X) \\ = \beta \Subset \alpha(\tau'_1, \ldots, \tau'_n) = \alpha(\sigma'(Y_1), \ldots, \sigma'(Y_n))\end{array}$$

- In all the other cases, we use Lemma 4.12 and the fact that σ_1 is an entropy-maximal solution, and the only non trivial cases are the conclusion constraints $X' = \omega' * Y_i \Subset \alpha'(Y'_1, \ldots, Y'_m)$ involving some Y_i. Then

$$\begin{array}{l}\sigma'(X') = \omega' * \sigma'(Y_i) = \omega' * \tau'_i \Subset \omega' * \tau_i = \omega' * \sigma_1(Y_i) \\ = \sigma_1(X') \Subset \alpha'(\sigma_1(Y'_1), \ldots, \sigma_1(Y'_m)) = \alpha'(\sigma'(Y'_1), \ldots, \sigma'(Y'_m))\end{array}$$

  Note that the support set of $\sigma'(Y_i) = \tau'_i$ may be larger than that of $\sigma_1(Y_i)$, i.e., $\mathcal{D}^{\phi}_{Y_i}$, and here again the "affine" hypothesis is used, to cancel the places which are not in $\mathcal{D}^{\phi}_{Y_i}$. It would be absolutely impossible to account for these places and their ordering constraints in the Simplification procedure, which may not even be aware of them.

$\square$

Theorems 4.11 and 4.13 thus show all the expected properties of Generation stated in Section 3.1.2.

4.5 Examples

In the following examples, atoms sharing the same letter (e.g., $p, p', \ldots$) are assumed unifiable. To simplify notations, negative atoms in bipoles are "primed" (as in $p', p'', \ldots$) and are also used to denote their corresponding places; positive atoms $p^\perp$ are also used to denote their corresponding place variable. This introduces an ambiguity, since the notation $[p = p']$ may mean either the unification of two first-order terms (the atoms p, p'), or the place infon produced by matching the place variable attached to the positive atom $p^\perp$ with the place attached to the negative atom p'. Unifications will simply be ignored here (i.e. we only match unifiable atoms, otherwise a deadend would be immediately produced), hence we adopt the second meaning.

1. Consider the following abstract proof

$$\cfrac{\cfrac{X_2}{X_1}\,[q^\perp \otimes a^\perp \otimes b^\perp \otimes r']}{X}\,[p^\perp \otimes (a' \bigtriangledown b')]$$

   We want to show that it is not possible to obtain a concrete proof by matching the occurrences of a, b in the upper bipole with the occurrences of a', b' in the lower bipole. The constraint system is

$$\cfrac{\cfrac{r' * Y_2 = X_2}{(a' < b') * Y_1 = X_1 \Subset Y_2 * (Y_b \parallel Y_a \parallel Y_q)}}{X \Subset Y_1 * Y_p}$$

   together with the terminal premise constraints $Y_p = p; Y_q = q; Y_a = a; Y_b = b$. The following transitions (at node X_1) are then possible. They constitute the branch of the tableau that leads to the deadend we want to show; of course, other branches may succeed (e.g., leaving a, b unmatched at step 3).

	Rule	Result
1	**INIT**	$[q \in \lvert Y_q\rvert]; [a \in \lvert Y_a\rvert]; [b \in \lvert Y_b\rvert]$
2	**SC**	$[q, a, b \in \lvert X_1\rvert]$
3	**SP**	$[a = a']; [b = b']; [q \in \lvert Y_1\rvert]$
4	**OP2**	$[X_1(a, b, q)]$
5	**OC3**	DEADEND

Step 3 corresponds to the choice of matching a with a', and b with b', and of not matching q at this level. The failure at Step 5 is caused by the cycle $X_1(a,b,q)$ obtained at Step 4 and contradicted by $X_1 \Subset Y_2 * (Y_b \parallel Y_a \parallel Y_q)$, which requires $X_1\restriction_{a,b,q} \subseteq \overline{b \parallel a \parallel q}$.

2. Another interesting case leading to a deadend is when two branches created by $\&$ induce contradictory ordering information:

$$\frac{\dfrac{X_3}{X_1}\,[q^\perp \otimes (a_1^\perp \odot b_1^\perp) \otimes q''] \qquad \dfrac{X_4}{X_2}\,[r^\perp \otimes (b_2^\perp \odot a_2^\perp) \otimes r'']}{\dfrac{X_0}{X}\,[m^\perp \otimes (b' \triangledown a')]}\,[p^\perp \otimes (q' \,\&\, r')]$$

The constraint system (in addition to the terminal premise constraints $Y_x = x$ for each atom x) is:

$$\frac{\dfrac{q'' * Y_1 = X_3}{q' * Y_0 = X_1 \Subset Y_1 * (Y_{b_1} < Y_{a_1}) \parallel Y_q} \qquad \dfrac{r'' * Y_2 = X_4}{r' * Y_0 = X_2 \Subset Y_2 * (Y_{a_2} < Y_{b_2}) \parallel Y_r}}{\dfrac{(b' < a') * Y = X_0 \Subset Y_0 * Y_p}{X \Subset Y * Y_m}}$$

Failure is obtained by the following sequence of transitions:

	Rule	Result
1	**INIT**	$[q \in \lvert Y_q\rvert]; [r \in \lvert Y_r\rvert]; [a_i \in \lvert Y_{a_i}\rvert]; [b_i \in \lvert Y_{b_i}\rvert]$
2	**SC** at X_1	$[q, a_1, b_1 \in \lvert X_1\rvert]$
3	**SC** at X_2	$[r, a_2, b_2 \in \lvert X_2\rvert]$
4	**SP** at X_1	$[a_1, b_1 \in \lvert Y_0\rvert]; [q = q']$
5	**SP** at X_2	$a_2 := a_1; b_2 := b_1$ and $[r = r']$
6	**SC** at X_0	$[p, a_1, b_1 \in \lvert X_0\rvert]$
7	**SP** at X_0	$[a_1 = a']; [b_1 = b']; [p \in \lvert Y\rvert]$
8	**OP2** at X_0	$[X_0(b_1, a_1, p)]$
9	**OC2** at X_0	$[Y_0(b_1, a_1)]$
10	**OP1** at X_2	$[X_2(r, b_1, a_1)]$
11	**OC3** at X_2	DEADEND

Step 4 chooses to match q with q' leaving a_1, b_1 unmatched at this level. Step 5 chooses to match r with r' and to substitute a_2, b_2 with a_1, b_1. This is different from matching say a_2 with a_1, which does not make sense since matching can only occur between a place and a place variable, not between two place variables. Substitution is a global operation which replaces a_2 by a_1 everywhere in the constraint system. Step 7 chooses to match a_1 with a' and b_1 with b' and to leave p unmatched at this level. Applying **OP1** at X_1 instead of X_2 after step 9 would be useless, as no contradiction would be detected on that side. Note that the order of application of the rewrite rules is, to a great extent, arbitrary; but any other order would lead to the same failure.

3. The following example illustrates importance of the "admissibility" condition in Theorem 2.1.

$$\frac{\dfrac{X_3}{X_1}\,[q^\perp \otimes (a^\perp \odot c^\perp) \otimes q''] \qquad \dfrac{X_4}{X_2}\,[r^\perp \otimes (b^\perp \otimes d^\perp) \otimes r'']}{\dfrac{X_0}{X}\,[m^\perp \otimes ((a' \triangledown b') \parr (c' \triangledown d'))]}\,[p^\perp \otimes q' \otimes r']$$

Its constraint system is

$$\frac{\dfrac{q'' * Y_3 = X_3}{q' * Y_1 = X_1 \Subset Y_3 * (Y_c < Y_a) \parallel Y_q} \qquad \dfrac{r'' * Y_4 = X_4}{r' * Y_2 = X_2 \Subset Y_4 * (Y_d \parallel Y_b) \parallel Y_r}}{\dfrac{((a' < b') \parallel (c' < d')) * Y_0 = X_0 \Subset Y_2 \parallel Y_1 * Y_p}{X \Subset Y_0 * Y_m}}$$

Trying to match a, b, c, d with a', b', c', d' fails. Indeed, this matching induces the order $(a < b) \parallel (c < d)$ on a, b, c, d, for which the split $\{a, c\}\{b, d\}$ is not admissible. The following sequence of transitions (not detailed here) shows this.

$$\begin{array}{l}\text{START} \to \left\{\begin{array}{l}[q, a, c \in |X_1|] \to [q = q']; [a, c \in |Y_1|] \to [a, c \in |X_0|] \to [a = a']; [c = c'] \\ [r, b, d \in |X_2|] \to [r = r']; [b, d \in |Y_2|] \to [b, d \in |X_0|] \to [b = b']; [d = d']\end{array}\right\} \to \\ {[X_0(a, c, d)]; [X_0(a, b, c)] \to [Y_1(a, c)]; [Y_1(c, a)] \to \text{DEADEND}}\end{array}$$

Note that the failure would not have been avoided, had the connective between $a^\perp$ and $c^\perp$ (here $\odot$) been reversed or replaced by $\otimes$ (and likewise for the connective between $b^\perp$ and $d^\perp$). It is the split $\{a, c\}\{b, d\}$ itself which is not admissible, not the way a, c and b, d are ordered on each side. Rule **OV1** here, detects the violation of the admissibility condition.

4. Finally, we give an example of Generation, illustrating the importance of the "affine" assumption.

$$\dfrac{\dfrac{\dfrac{X_3}{X_2}\ [b^\perp \otimes (d^\perp \odot r^\perp) \otimes g']}{X_1}\ [(a^\perp \odot c^\perp \odot q^\perp) \otimes (e' \triangledown d')]}{X}\ [p^\perp \otimes (c' \triangledown (a' \parr b'))]$$

The constraint system is:

$$\begin{array}{c}
g' * Y_3 = X_3 \\ \hline
(e' < d') * Y_2 = X_2 \Subset Y_3 * (Y_r < Y_d) \parallel Y_b \\ \hline
c' < (a' \parallel b') * Y_1 = X_1 \Subset Y_2 * (Y_q < Y_c < Y_a) \\ \hline
X \Subset Y_1 * Y_p
\end{array}$$

Matching a, b, c, d with a', b', c', d', we get the following open state at the end of the Simplification procedure:

$$\begin{array}{l}
[b, d, r \in |X_2|]; [b, r \in |Y_2|]; [a, b, c, q, r \in |X_1|]; [q, r \in |Y_1|]; [p, q, r \in |X|]; \\
[a = a']; [b = b']; [c = c']; [d = d']; \\
[X_1(c, \substack{a \\ b}, \substack{q \\ r})]; [Y_2(b, r)]; [X_2(d, b, r)]
\end{array}$$

From this state, the Generation procedure leads to:

	Rule	Result
1	**P-INIT**	$[Y_a\lceil_a \trianglelefteq a]; \ldots$
2	**PP**	$[Y_3\lceil_\emptyset \trianglelefteq \epsilon]$
3	**PC**	$[X_2\lceil_{b,d,r} \subseteq \epsilon * (r < d) \parallel b]$
4	**PP**	$[Y_2\lceil_{b,r} \trianglelefteq b < r]$
5	**PC**	$[X_1\lceil_{a,b,c,q,r} \subseteq b < r * q < c < a]$
6	**PP**	$[Y_1\lceil_{q,r} \trianglelefteq q \parallel r]$
7	**PC**	$[X\lceil_{p,q,r} \subseteq (q \parallel r) * p]$

$\downarrow \quad \uparrow$

	Rule	Result
14	**TP**	$[X_3 = g]$
13	**TC**	$[Y_3 = \epsilon]$
12	**TP**	$[X_2 = e < d * b < r]$
11	**TC**	$[Y_2 = b < r]$
10	**TP**	$[X_1 = c < (a \parallel b) * q \parallel r]$
9	**TC**	$[Y_1 = q \parallel r]$
8	**T-INIT**	$[X = (q \parallel r) * p]$

Here, steps 2 and 4 are deterministic. In particular, there is no alternative at step 4, because $Y_2\lceil_{b,r}$ must contain $\dot{Y}_2 = b < r$. At step 6, $Y_1\lceil_{q,r}$ must be an order τ satisfying the condition $c < (a \parallel b) * \tau \subseteq (q < c < a) * (b < r)$. The choice made at Step 6 is $\tau = q \parallel r$, which works, but $\tau = r < q$ would also have been possible. Steps 9 to 13 choose the smallest solutions authorised by steps 6 to 2. The Generation procedure guarantees that at least that solution works, but there may be other solutions. In the end, the following concrete proof has been generated:

$$\dfrac{\dfrac{\dfrac{g * \epsilon}{e < d * b < r}\ [b^\perp \otimes (d^\perp \odot r^\perp) \otimes g]}{c < (a \parallel b) * q \parallel r}\ [(a^\perp \odot c^\perp \odot q^\perp) \otimes (e \triangledown d)]}{(q \parallel r) * p}\ [p^\perp \otimes (c \triangledown (a \parr b))]$$

The middle inference implicitly uses the Entropy $c<(a\|b)*q\|r \subseteq q<c<a*b<r$. The top inference implicitly uses the Weakening of e from $e<d*b<r$ to $d*b<r$ and then the equality $d*b<r = (r<d)\|b*\epsilon$. Note that the Weakening step is essential. It is not possible to perform the top inference if e is not discarded. In fact, the Simplification procedure propagates information only on places which have been matched. Here, e' has not been matched, and therefore, its capacity to fail the top inference could not be detected by the Simplification procedure. The capacity at all time to get rid of unmatched places by Weakening solves the problem. Note that the problem did not occur in the fragments considered in the previous sections (linear logic and cyclic logic): in both cases, the Simplification procedure deals only with matched places, but the unmatched ones happen not to create undetected failure.

5 Conclusion

In this paper, we have investigated proof construction in the framework of non-commutative logic. We have extended the constraint based approach to proof construction proposed for linear logic in [3], first to the cyclic fragment of NL, then to an affine variant of full multiplicative, additive, first-order NL.
NL is a particular case of "coloured" linear logic [4], which is linear logic with structure. One problem with such logic is that the choice of structure is somehow arbitrary, and that is not satisfactory: logical rules should express necessity, not some a priori choice of structure. It is therefore important to characterise the different structures by the global properties they confer to the logic, in particular in terms of proof construction.
The present paper describes a constraint based proof construction algorithm for the structures of orders and order varieties. It identifies the splitting property (Theorem 2.1) of these structures as conferring to the associated logic a global property of effectiveness in the proof construction. It can quite straightforwardly be generalised to other structures which, like order varieties, are fully defined by their restrictions to triples. Arbitrary ternary cyclic relations or pre-order varieties are examples of such structures. Future work includes the investigation of other classes of structures for which this assumption does not hold.

References

[1] V. M. Abrusci and P. Ruet. Non-commutative logic I: the multiplicative fragment. *Annals of Pure and Applied Logic*, 101(1):29–64, 2000.

[2] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.

[3] J.-M. Andreoli. Focussing and proof construction. *Annals of Pure and Applied Logic*, 107(1-3):131–164, 2001.

[4] J.-M. Andreoli. An axiomatic approach to structural rules for locative linear logic. In T. Ehrhard, J.-Y. Girard, P. Ruet, and Ph. J. Scott, editors, *Linear logic in computer science*, volume 316 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 2004.

[5] J.-M. Andreoli, S. Freeman, and R. Pareschi. The coordination language facility: coordination of distributed objects. *Journal of Theory and Practice of Object Systems*, 2(2):77–94, 1996.

[6] S. Cerrito. Herbrand methods in sequent calculi: Unification in ll. In K.R. Apt, editor, *International Conference on Logic Programming*, pages 607–621, Washington, DC, U.S.A., 1992. MIT Press.

[7] C. Faggian. Proof construction and non-commutativity: a cluster calculus. In *Principles and Practice of Declarative Programming*. ACM Press, 2000.

[8] J.-Y. Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Sciience*, 1(3):255–296, 1991.

[9] J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[10] A. Joyal. Une théorie combinatoire des séries formelles. *Advances in Mathematics*, 42:1–82, 1981.

[11] F. Lamarche. On the algebra of structural contexts. *Manuscript*, 2002.

[12] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.

[13] O. Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, 2002.

[14] R. Maieli and P. Ruet. Non-commutative logic III: focusing proofs. *Information and Computation*, 185(2):233–262, 2003.

[15] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[16] D. Miller. Lexical scoping as universal quantification. In G. Levi and M. Martelli, editors, *International Conference on Logic Programming*, Lisbon, Portugal, 1989. MIT Press.

[17] D. Miller. A multiple-conclusion meta-logic. *Theoretical Computer Science*, 165:201–232, 1996.

[18] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[19] R. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–193. Kluwer Academic Publishers, 1989.

[20] G. Nadathur. A proof procedure for the logic of hereditary harrop formulas. *Journal of Automated Reasoning*, 11(1):115–145, 1993.

[21] P. Ruet. Non-commutative logic II: sequent calculus and phase semantics. *Mathematical Structures in Computer Science*, 10(2):277–312, 2000.

[22] D.N. Yetter. Quantales and (non-commutative) linear logic. *Journal of Symbolic Logic*, 55(1), 1990.