ECOLE NORMALE SUPERIEURE

Concurrent Constraint Programming and Non-Commutative Linear Logic

(Extended Abstract)

P. RUET
F. FAGES

LIENS - 96 - 25

Département de Mathématiques et Informatique

CNRS URA 1327

Concurrent Constraint Programming and Non-Commutative Linear Logic

(Extended Abstract)

P. RUET
F. FAGES

LIENS - 96 - 25

December 1996

Laboratoire d'Informatique de l'Ecole Normale Supérieure
45 rue d'Ulm 75230 PARIS Cedex 05

Tel : (33)(1) 44 32 00 00

Adresse électronique : ... @dmi.ens.fr

Concurrent Constraint Programming and Non-Commutative Linear Logic (Extended Abstract)

Paul Ruet, François Fages
{ruet,fages}@dmi.ens.fr
LIENS-CNRS, Ecole Normale Supérieure, Paris, France

Abstract

This paper presents a precise connection between a non-commutative version of intuitionistic linear logic (INLL) and concurrent constraint programming (cc). The contribution of this paper is twofold:

• on the one hand, we refine existing logical characterizations of operational aspects of concurrent constraint programming, by providing a logical interpretation of finer observable properties of cc programs, namely successes and suspensions.

• on the other hand, this work stems in the line of research aiming at establishing close correspondences between linear logic and concurrent programming, by showing that, for formulas satisfying a certain simple condition, *deductions* correspond exactly to *transitions* between agents, without any extra-logical operator nor any restriction on the form of proofs.

1 Introduction

This paper presents a precise connection between a non-commutative version of intuitionistic linear logic (INLL) and concurrent constraint programming (cc). The contribution of this paper is twofold:

• on the one hand, we refine existing logical characterizations of operational aspects of concurrent constraint programming, by providing a logical interpretation of finer observable properties of cc programs, namely successes and suspensions.

• on the other hand, this work stems in the line of research aiming at establishing close correspondences between linear logic and concurrent programming, by showing that, for formulas satisfying a certain simple condition, *deductions* correspond exactly to *transitions* between agents. Examples of such correspondences have been proposed for LO [2] and the π-calculus [20] at the expense of extra-logical operators, for concurrent logic programming [23, 14] at the expense of restrictions on the form of proofs, and for Petri nets and a propositional fragment of linear logic [19, 3]. One aspect of our correspondence between cc and first-order non-commutative linear logic is that it does not assume such restrictions.

Concurrent constraint programming [29] is a model of concurrent computation, where concurrent agents communicate through a shared store, represented by a constraint, which expresses some *partial information* on the values of the variables involved in the computation. An agent may add a constraint c to the store, or ask the store to entail a given constraint $(c \to A)$. Communication is *asynchronous*: agents can remain idle, and senders (constraints c) are not blocking.

Syntactically, concurrent constraint programming is an extension of *constraint logic programming* [11, 18] with a suspension mechanism $c \to A$, and the operational semantics of cc is the same as that of constraint logic programming, except for $c \to A$ which blocks until the amount of

accumulated information (the store) is strong enough to entail c, in which case $c \rightarrow A$ evolves to A. This gives rise to a natural form of data-driven computation, that generalizes the delay mechanism of Prolog with freeze and of early CLP systems such as CHIP [34] [10], and is now central in the applications of constraint programming to complex system modelling and combinatorial optimization problems [17].

Inherited from the process algebra tradition, cc languages can be presented very simply by a set of operators for parallel composition $\|$, non-deterministic choices $+$ (angelic or demoniac, blind or one-step), hiding $\exists$, blocking ask $\rightarrow$, and by a transition system expressing the operational semantics of the agents. The operational semantics can be defined either in the SOS style [28] or in the chemical abstract machine [4] style which we shall retain here (see table 1 and section 2 for the definition of $\longrightarrow_{cc}$ transitions). Computation is *monotonic* (the constraints in the store are not consumed): this allows to provide cc with a denotational semantics, viewing agents as closure operators on the semi-lattice of constraints [31, 12].

From the logic programming tradition however, the operational aspects of cc programming should also be closely connected to proof theory, via the *computation-as-proof-search* paradigm. This paradigm, first introduced for the Horn clause fragment of classical logic, has been smoothly applied to constraint logic programming (with, or without negation by failure, or constructive negation), where the logical nature of the constraint system extends to the goals and program declarations, and states strong connections between operational semantics and entailment [11, 18, 32, 6]. For instance, success constraints (i.e. final states of computations) can be observed logically: any success entails the initial state (modulo the logical translation of the program $P^\star$ and the constraint system $\mathcal{C}$); conversely any constraint c entailing a goal G is covered (again modulo $P^\star$ and $\mathcal{C}$) by a finite set of successes $c_1 \ldots c_n$, i.e. $\mathcal{C} \vdash \forall(c_1 \ldots c_n \Rightarrow c)$. Such results make easier the design and understanding of programs, and provide useful tools for reasoning about them.

Maher in [18] was the first to suggest that the synchronization mechanism in concurrent logic programming could be given a logical interpretation. In [16] Lincoln and Saraswat give an interesting connection between the observation of the stores of cc agents and entailment in intuitionistic logic (IL). The basic idea is to express agents and observations by formulas and to read a sequent $\Gamma \vdash \Delta$ as "the agent Γ satisfies the test Δ". Their main result establishes a logical interpretation of the observation of the *stores* entailed in each branch of the derivation tree: for any constraint c and any (formula associated to a cc) agent Γ, $\Gamma \vdash_{IL} c$ iff $\Gamma \longrightarrow_{cc} (c_1 \wedge B_1) \vee \ldots \vee (c_n \wedge B_n)$ and for all $1 \leq i \leq n$ $\mathcal{C} \vdash \forall(c_i \rightarrow c)$.

However such a logical semantics does not accommodate other notions of observations. Actually let a *success* of an agent A be a store c such that A evolves to c, and let a *suspension* be an agent $B = c \parallel (d \rightarrow A)$ such that A evolves to B and c does not entail d (the exact definition is slightly longer). The three programs $p(x) = x \geq 1$, $p(x) = x \geq 1 \parallel p(x)$, and $p(x) = x \geq 1 \parallel (false \rightarrow A)$ have the same stores and are therefore indistinguishable, although the first succeeds, the second loops and the third suspends. As shown in [26] through examples the observation of successes or suspensions is in fact not expressible in intuitionistic logic. Roughly speaking, the interpretation of cc agents as intuitionistic formulas stumbles against the structural rule of (left) *weakening*:

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

Girard's linear logic [8] enables a control on the weakening and *contraction* structural rules of classical and intuitionistic logics. It seems therefore natural to interpret concurrent constraint programs in linear logic. While moving to linear logic, it is very natural to move to a *non-monotonic* version of cc at the same time, where constraints are consumed, but where monotonic cc can be easily encoded. Such variants have been introduced by Saraswat and Lincoln in a higher-order setting [30], further studied in [5, 33], where the logic of constraints is linear logic: in this version, constraints can be consumed, and the language is therefore closer to process calculi like Milner's π-calculus [21].

In [26], a first-order non-monotonic variant, lcc, is defined in which the successes are characterized in ILL (intuitionistic multiplicative and additive linear logic). On the contrary, suspensions cannot be characterized in ILL, because from

$$A \otimes (B \multimap C) \vdash B \multimap (A \otimes C)$$

one cannot conclude that $A \parallel (B \to C) \longrightarrow B \to (A \parallel C)$ suspends: A might add enough information into the store to unblock the constraint c (for instance $c \otimes (c \multimap 1) \vdash c \multimap (c \otimes 1)$ but $c \otimes (c \multimap 1) \vdash 1$ as well, and indeed the agent $c \parallel c \to 1$ succeeds with 1, and does not suspend).

There is a lack of a "sequential" connective, that is a non-commutative one. The actual non-commutative versions of linear logic do not provide any immediate solution (for instance Retoré's *before* connective $<$ [25] enjoys the same 'porosity' property as linear implication: $A \otimes (B < C) \vdash B < (A \otimes C)$).

In this paper we show that the intuitionistic fragment of a new non-commutative version of linear logic (INLL) copes with this difficulty (Section 3): we show that the successes and the suspensions of an lcc computation can be characterized in INLL (Section 4). We then refine this connection between INLL and cc, by showing that, when focusing on formulas satisfying a certain condition (absence of existentially quantified variables in the final agent), deductions correspond exactly to transitions between agents.

Note that these results hold for (usual) monotonic cc as well, because, as we shall see, cc can be faithfully translated into lcc.

2 Preliminaries on concurrent constraint programming

2.1 Non-monotonic lcc

A *linear constraint system* is a pair $(\mathcal{C}, \Vdash_{\mathcal{C}})$ where: $\mathcal{C}$ is a set of formulas (the *linear constraints*) built from a set V of variables, a set Σ of function and relation symbols, and logical operators 1, the multiplicative conjunction $\otimes$ and the existential quantifier $\exists$; and $\Vdash_{\mathcal{C}} \subseteq \mathcal{C}^\star \times \mathcal{C}$. We assume $\otimes$ has neutral 1. Instead of $((c_1 \ldots c_n), c) \in \Vdash_{\mathcal{C}}$, we write $c_1 \ldots c_n \Vdash_{\mathcal{C}} c$.

$\vdash_{\mathcal{C}}$ is the least reflexive and transitive relation $\subseteq \mathcal{C}^\star \times \mathcal{C}$ containing $\Vdash_{\mathcal{C}}$ and closed by the following rules ($\mathrm{fv}(A)$ denotes the set of free variables occurring in A):

$$\frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c} \qquad \frac{\Gamma \vdash c_1 \qquad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \qquad \frac{\Gamma, c \vdash d \qquad \Delta \vdash c}{\Gamma, \Delta \vdash d}$$

$$\frac{\Gamma \vdash c}{\Gamma \vdash \exists x c} \qquad \frac{\Gamma, A \vdash c}{\Gamma, \exists x A \vdash c}\ x \notin fv(\Gamma, c)$$

They are the rules of intuitionistic linear logic (ILL) for $\otimes$ and $\exists$, plus the cut rule. The syntax of lcc agents is given by the following grammar:

$$A ::= p(\vec{x}) \mid c \mid (A \parallel A) \mid A + A \mid \exists x A \mid c \to A$$

where $\parallel$ stands for parallel composition and $\to$ for blocking ask. In an agent written $A = c \parallel A_1 \parallel \cdots \parallel A_n$, c is a constraint, the main constructor of each A_i is not $\parallel$, and no A_i is a constraint. We call c the *store* of A, it is the global available information.

Recursion is obtained with declarations: $D ::= \epsilon \mid p(\vec{x}) = A \mid D, D$

As usual, the precise operational semantics of cc programs depends on the choice of observables. In this paper we shall first consider successes and suspensions. A *success* for an lcc agent A is a

linear constraint c such that $A \longrightarrow c$. A *suspension* for A is an agent $B = c \parallel (d_1 \to A_1) \parallel \cdots \parallel (d_n \to A_n)$ such that $A \longrightarrow B$ and for no i, $c > d_i$, where the relation between constraints "$c > d$" is the least relation containing $\Vdash_{\mathcal{C}}$ and closed by the rule $c > d \;\Rightarrow\; (c \otimes e) > d$.

We shall define the operational semantics by a transition system which abstracts from specific evaluation strategies. The transition system is given in table 1 in the style of the Chemical Abstract Machine [4] (see also [24]). This presentation, though different from a logic programming one, has the advantage of keeping track of the variable bindings, and we find it therefore cleaner to manage logically. We also chose to define the transition relation as a congruence (thus $c \to A \;\longrightarrow c \to B$ whenever $A \;\longrightarrow B$), although such transitions can obviously be discarded (as expected) without affecting the observation of successes and suspensions.

• The *structural congruence* $\equiv$ is the least congruence relation such that $(\mathcal{A}/\equiv, \parallel, 1)$ is an abelian monoid, $(\mathcal{A}/\equiv, +)$ is an abelian semi-group, and such that, for all agents A and B:

$$\frac{A \text{ and } B \text{ are } \alpha\text{-convertible}}{A \equiv B}$$

$$A + A \equiv A \qquad 1 \to A \equiv A$$

$$\exists x\, 1 \equiv 1 \qquad \exists x \exists y A \equiv \exists y \exists x A$$

$$\frac{x \text{ is not free in } A}{\exists x (A \parallel B) \equiv A \parallel \exists x B} \qquad \frac{x \text{ is not free in } c}{\exists x (c \to A) \equiv c \to \exists x A}$$

• The *transition* relation between agents $\longrightarrow$ is the least reflexive transitive congruence such that:

$$(d \otimes c) \parallel (c \to A) \longrightarrow (d \parallel A)$$

$$\frac{c \vdash_{\mathcal{C}} d}{c \longrightarrow d} \qquad \frac{c \vdash_{\mathcal{C}} d}{(d \to A) \longrightarrow (c \to A)} \qquad (c \to d) \parallel (d \to A) \longrightarrow (c \to A) \qquad c \to (d \to A) \longrightarrow (c \otimes d) \to A$$

$$\frac{(p(\vec{x}) = A) \in P}{p(\vec{x}) \longrightarrow A} \qquad \frac{A' \equiv A \qquad A \longrightarrow B \qquad B \equiv B'}{A' \longrightarrow B'}$$

$$A + B \longrightarrow A \qquad A + B \longrightarrow B$$

Table 1: Transition system of `lcc` with blind choice.

Remarks:

▶ In the *communication* rule $(d \otimes c) \parallel (c \to A) \longrightarrow (d \parallel A)$, the constraint c is consumed by the agent $c \to A$. Therefore communication is intrinsically non-deterministic since several constraints may satisfy the condition of the rule.

▶ The non-deterministic choice $A + B$, called *blind choice*, can behave either like A or like B, it has both capabilities. This is in slight contrast with the *one-step guarded choice*, defined by $\frac{A \longrightarrow A'}{A + B \longrightarrow A'}$ and $\frac{B \longrightarrow B'}{A + B \longrightarrow B'}$. As remarked in [12, 7] the difference between (angelic, backtracking) non-determinism and (demonic, committed-choice) in-determinism arises in the way observations are defined, however angelic non-determinism refers generally to the blind choice rule and demonic non-determinism to the one-step committed choice rule. Only the blind choice rule will be considered in this paper. Agents and declarations not involving $+$ are said *deterministic*.

2.2 Translation of monotonic `cc` into `lcc`

The monotonic (original) version of `cc` can be recovered very simply. The basic idea is to restitute the consumed constraint after communication, in the translation of a suspension $c \to A$. This leads to the following translation:

Let $(\mathcal{C}, \Vdash_{\mathcal{C}})$ be a constraint system. We define the linear constraint system $(\mathcal{C}, \Vdash_{\mathcal{C}})^\bullet$ and the translation of `cc` agents into `lcc` agents:

$$c^\bullet = c, \text{ if } c \text{ is an atomic constraint other than } \top \qquad \top^\bullet = 1$$
$$p(\vec{x})^\bullet = p(\vec{x}) \qquad (A + B)^\bullet = A^\bullet + B^\bullet$$
$$(A \parallel B)^\bullet = A^\bullet \parallel B^\bullet \qquad (\exists x\, A)^\bullet = \exists x A^\bullet$$
$$(c \to A)^\bullet = c \to (c \parallel A^\bullet)$$

The deduction relation $\Vdash^\bullet_{\mathcal{C}}$ is defined by: $c \Vdash_{\mathcal{C}} d \Rightarrow c^\bullet \Vdash^\bullet_{\mathcal{C}} d^\bullet$, and for any constraint c, $c^\bullet \Vdash^\bullet_{\mathcal{C}} c^\bullet \otimes c^\bullet$ and $c^\bullet \Vdash^\bullet_{\mathcal{C}} 1$.

The transition relation $\longrightarrow^\bullet$ is the transition relation of monotonic `cc`, i.e. the same as that of `lcc`, except for the communication rule $(d \otimes c) \parallel (c \to A) \longrightarrow_{lcc} (d \parallel A)$ which is replaced by $c \parallel (c \to A) \longrightarrow_{cc} (c \parallel A)$.

Remark:
▶ In order to translate the deduction of constraints, we could naturally use of the so-called exponential connectives of linear logic (specifically !), and avoid this way the *ad hoc* axioms of $\Vdash^\bullet_{\mathcal{C}}$, but then we should take care to limitate the use of ! to constraints only[1], and this would not be very smooth. Our translation has the advantage to rely on the simple multiplicative-additive fragment.

The following proposition is then easily proved:

Proposition 1 *Let c and d be constraints: $c \vdash d$ iff $c^\bullet \vdash^\bullet d^\bullet$. Let A and B be `cc` agents: $A \equiv B$ iff $A^\bullet \equiv^\bullet B^\bullet$, $A \longrightarrow B$ iff $A^\bullet \longrightarrow^\bullet B^\bullet$.*

3 Non-Commutative Linear Logic

The complete presentation of that non-commutative linear logic is the topic of another paper [27], and is partially recalled in Appendix. We just present here the intuitionistic fragment of interest for the present paper.

The formulas are built from atoms $p, q, \ldots$, the *constant* 1, the *existential quantifier* $\exists$ and connectives: a (multiplicative) *commutative* conjunction $\otimes$, a (multiplicative) *non-commutative* implication $\rightsquigarrow$ and the *additive* conjunction &. Like Yetter [35], and contrary to Lambek [15] and Abrusci [1], we simply consider one implication.

Defining a sequent calculus for a linear logic mixing both commutative and non-commutative multiplicatives raises the problem of representing the information on the way the formulas in the sequent must be combined (either by $\otimes$ or by $\rightsquigarrow$). Following Retoré [25], we shall represent this information by an ordering (on occurrences of formulas) associated to the sequent.

Sequents are of the form $\Gamma \vdash A$ [i], where i is a simple oriented graph (at most one edge from a vertex to another) indexed by the multiset of formulas Γ, A. $A \vdash B$ is defined by $\vdash A^\perp, B$ $[A^\perp \leftrightarrow B]$. We define the following operations and relations on i:

• if x and y are two vertices of i, we note $\mathfrak{i}\{z/x, y\}$ the graph obtained by identifying in i the vertices x and y, renamed z (in case $x \leftrightarrow_{\mathfrak{i}} y$, the two edges from z to z should be identified as well);

• we note $\mathfrak{i} \cup \mathfrak{j}$ the graph with edges and vertices those of i and j;

• if x is a vertex of i and y a vertex of j, we note $\mathfrak{i} \asymp_{x,y} \mathfrak{j}$ the graph obtained from $\mathfrak{i} \cup \mathfrak{j}$ by adding an edge from t to s, $\forall s \in \mathfrak{i}$ such that $x \mathfrak{i} s$ and $\forall s \in \mathfrak{i}$ such that $t \mathfrak{j} y$, adding an edge from s to t, $\forall s \in \mathfrak{i}$ such that $s \mathfrak{i} x$ and $\forall t \in \mathfrak{j}$ such that $y \mathfrak{j} t$, and then erasing the vertices x and y and all edges from or to x or y; this will be used in the cut rule;

[1] Usually the replication operator of process calculi (like the π-calculus [21], where it is written ! as well) does not have the same operational behaviour as the exponential connective: it enables duplication ($!A \to (!A \parallel !A)$) but not erasing ($!A \not\to 1$).

• if x is a vertex of $\mathfrak{i}$ and y a vertex of $\mathfrak{j}$, we note $\mathfrak{i}\curlyvee_{x,y}\mathfrak{j}$ the graph obtained from $\mathfrak{i}\cup\mathfrak{j}\{x \rightsquigarrow y/x, y\}$ by adding an edge from t to s, $\forall s \in \mathfrak{i}$ such that $x\mathfrak{i}s$ and $\forall s \in \mathfrak{i}$ such that $t\mathfrak{j}y$, and then erasing the vertices from x or to y; this will be used in the $\rightsquigarrow$ left introduction;

• if x is a vertex of $\mathfrak{i}$ and y a vertex of $\mathfrak{j}$, we note $\mathfrak{i}\otimes_{x,y}\mathfrak{j}$ the graph obtained from $\mathfrak{i}\cup\mathfrak{j}\{x \otimes y/x, y\}$ by adding an edge from t to s, $\forall s \in \mathfrak{i}$ such that $x\mathfrak{i}s$ and $\forall s \in \mathfrak{i}$ such that $t\mathfrak{j}y$, adding an edge from s to t, $\forall s \in \mathfrak{i}$ such that $s\mathfrak{i}x$ and $\forall t \in \mathfrak{j}$ such that $y\mathfrak{j}t$, and then erasing all edges from or to x or y; this will be used in the $\otimes$ right introduction;

• if x and y are two vertices of $\mathfrak{i}$, we shall say $x \prec_{\mathfrak{i}} y$ iff $\mathfrak{i}$ contains an edge from x to y and no other egde starting from x or arriving at y.

The rules of the sequent calculus are given in table 2.

Axiom / cut

$$A^x \vdash A^y \ [A^x \leftrightarrow A^y] \qquad \frac{\Gamma \vdash A^x \ [\mathfrak{i}] \qquad \Delta, A^y \vdash B \ [\mathfrak{j}]}{\Gamma, \Delta \vdash B \ [\mathfrak{i} \asymp_{A^x, A^y} \mathfrak{j}]}$$

Commutative

$$\frac{\Gamma \vdash A \ [\mathfrak{i}] \qquad \Delta \vdash B \ [\mathfrak{j}]}{\Gamma, \Delta \vdash A \otimes B \ [\mathfrak{i} \otimes_{A,B} \mathfrak{j}]} \qquad \frac{\Gamma, A, B \vdash C \ [\mathfrak{i}]}{\Gamma, A \otimes B \vdash C \ [\mathfrak{i}\{A \otimes B/A, B\}]}$$

Non-commutative

$$\frac{\Gamma, A \vdash B \ [\mathfrak{i}]}{\Gamma \vdash A \rightsquigarrow B \ [\mathfrak{i}\{A \rightsquigarrow B/A, B\}]} A \prec_{\mathfrak{i}} B \qquad \frac{\Gamma \vdash A \ [\mathfrak{i}] \qquad \Delta, B \vdash C \ [\mathfrak{j}]}{\Delta, \Gamma, A \rightsquigarrow B \vdash C \ [\mathfrak{i} \curlyvee_{A,B} \mathfrak{j}]}$$

Additive

$$\frac{\Gamma \vdash A \ [\mathfrak{i}] \qquad \Gamma \vdash B \ [\mathfrak{i}\{B/A\}]}{\Gamma \vdash A\&B \ [\mathfrak{i}\{A\&B/A\}]} \qquad \frac{\Gamma, A \vdash C \ [\mathfrak{i}]}{\Gamma, A\&B \vdash C \ [\mathfrak{i}\{A\&B/A\}]} \qquad \frac{\Gamma, B \vdash C \ [\mathfrak{i}]}{\Gamma, A\&B \vdash C \ [\mathfrak{i}\{A\&B/B\}]}$$

Existential quantifier

$$\frac{\Gamma \vdash A \ [\mathfrak{i}]}{\Gamma \vdash \exists x A \ [\mathfrak{i}\{\exists x A/A\}]} \qquad \frac{\Gamma, A \vdash B \ [\mathfrak{i}]}{\Gamma, \exists x A \vdash B \ [\mathfrak{i}\{\exists x A/A\}]} x \notin fv(\Gamma, B)$$

Constant

$$\vdash 1 \ [1 \leftrightarrow 1] \qquad \frac{\Gamma \vdash C \ [\mathfrak{i}]}{\Gamma, 1 \vdash C[\mathfrak{i}\{(A \leftrightarrow 1 \leftrightarrow B \leftrightarrow A)/(A \leftrightarrow B)\}]} (A \leftrightarrow_{\mathfrak{i}} B) \in (\Gamma, C)$$

Table 2: Sequent calculus for an intuitionistic fragment of INLL.

Theorem 1 *The sequent calculus given in Table 2 enjoys cut-elimination.*

This is a direct consequence of the cut-elimination theorem given in [27] for the sequent calculus

of *classical* non-commutative linear logic, as the cut-elimination procedure preserves the intuitionistic nature of sequents (i.e. at most one formula on the right of a sequent).

Comments:

▶ The graph associated to a sequent enables to express sequentiality constraints on formulas (for instance $A \prec_{\mathfrak{i}} B$). Some rules are subject to such constraints, most notably the right introduction $\vdash\rightsquigarrow$ of the non-commutative implication $\rightsquigarrow$, whereas other rules are not sensitive to these constraints, for instance $\otimes\,\vdash$. The interplay of sequentiality and parallelism in the sequent enables for instance to prove $c \otimes (c \rightsquigarrow A) \vdash A$ but, as expected, $A \otimes (d \rightsquigarrow B) \not\vdash d \rightsquigarrow (A \otimes B)$.

In fact this is the most important, the rest of the sequent calculus just keeps the system coherent (in particular w.r.t. cut-elimination).

▶ Most of the rules of this calculus will be read as transitions of processes: *axiom* expresses a form of asynchrony (agents can idle indefinitely), *cut* expresses the ability to compose executions, and the rules for the additive conjunction expresses non-deterministic choice. The rules for $\otimes$ (in particular the left introduction) tell just that two formulas A and B can always be read as the single formula $A \otimes B$. The left introduction of $\exists$ expresses hiding. The interpretation of other rules is not so easy, and is the subtelity of the next section.

We need the following lemma:

Lemma 1 *In a provable sequent* $\Gamma \vdash A$ [i],
(i) for any formula B *in* Γ, A, $\mathfrak{i}$ *contains an edge from* B *and an edge to* B*;*
(ii) the unoriented graph associated to $\mathfrak{i}$ *is connected.*

4 The correspondence cc - INLL

Let $(\mathcal{C}, \Vdash_{\mathcal{C}})$ be a constraint system, and $\mathcal{D}$ be a set of declarations. The `lcc` agents are translated into formulas as follows[2]:

$$c^\dagger = c, \text{ if } c \text{ is a constraint} \qquad p(\vec{x})^\dagger = p(\vec{x}) \qquad (\exists x\, A)^\dagger = \exists x\, A^\dagger$$
$$(c \to A)^\dagger = c \rightsquigarrow A^\dagger \qquad (A \parallel B)^\dagger = A^\dagger \otimes B^\dagger \qquad (A + B)^\dagger = A^\dagger \& B^\dagger$$

Let INLL($\mathcal{C}$,$\mathcal{D}$) be the deduction system obtained by extending INLL with:
- for each $c \Vdash_{\mathcal{C}} d$ in $\Vdash_{\mathcal{C}}$, $c \vdash d$ $[c \leftrightarrow d]$ as a non-logical axiom,
- for each declaration $p(\vec{x}) = A$ in $\mathcal{D}$, the sequent $p(\vec{x}) \vdash A^\dagger$ $[p(\vec{x}) \leftrightarrow A^\dagger$ as a non-logical axiom.

By a simple induction on $\equiv$ and $\longrightarrow$, we can easily prove the following:

Theorem 2 (Soundness) *Let* A *and* B *be* `lcc` *agents.*
If $A \equiv B$ *then* $A\dagger \vdash\dashv_{INLL(\mathcal{C},\mathcal{D})} B\dagger$. *If* $A \longrightarrow B$ *then* $A^\dagger \vdash_{INLL(\mathcal{C},\mathcal{D})} B^\dagger$.

This is the minimum, and holds in fact in any (suitable) logical interpretation (intuitionistic, or commutative linear logic). The interesting point of our refined logic is that the logical reading of agents captures finer operational aspects of (`l`)`cc` computations: indeed a converse (completeness) theorem holds for the observation of successes and suspensions.

We first need the following result, where $\longrightarrow^+$ denotes the transition system obtained from $\longrightarrow$ by adding the following rule:

$$A \longrightarrow^+ \exists x A$$

[2]It might be tempting to try to translate as well the operator for sequential composition ";" proposed for `cc` in [29] by the non-commutative tensor $\times$ (see Appendix A). The operational rule $\frac{A \longrightarrow A'}{A;B \longrightarrow A';B}$ raises however the same kind of difficulty as the one-step guarded choice rule $\frac{A \longrightarrow A'}{A + B \longrightarrow A'}$ which has not been translated logically, and seems deeply non-logical. For similar reasons the sequential composition is thus not considered in this paper.

Lemma 2 *If x is a variable not occurring free in A and B, and $A \longrightarrow^+ c \to B$, then $A \longrightarrow^+ (\exists x c) \to B$,*

The lemma is used in the proof of the following lemma (it is of minor interest in itself, can be easily proved, and holds for $\longrightarrow$ as well).

Lemma 3 *Let A and B be `lcc` agents. If $A^\dagger \vdash_{INLL(\mathcal{C},\mathcal{D})} B^\dagger$ then $A \longrightarrow^+ B$.*

The proof is given in the Appendix.

Let us say an agent A *suspends with store c and blocking constraints $d_1, ..., d_n$*, if there exists a suspension for A of the form $B = c \parallel (d_1 \to A_1) \parallel \cdots \parallel (d_n \to A_n)$.

Theorem 3 (Observation of successes and suspensions) *Let A be any `lcc` agent.*

(1) If c is a constraint such that $A^\dagger \vdash_{INLL(\mathcal{C},\mathcal{D})} c$, then c is a success for A, i.e. $A \longrightarrow c$.

(2) If $A^\dagger \vdash_{INLL(\mathcal{C},\mathcal{D})} (c \parallel (d_1 \to A_1) \parallel \cdots \parallel (d_n \to A_n))^\dagger$ and for no i, $c > d_i$, then A has a suspension with store c and blocking constraints $d_1 \ldots d_n$.

The new transition relation $\longrightarrow^+$ and Lemma 3 are not just tools in the proof of Theorem 3: Lemma 3 tells that the only difference between transitions between `cc` agents and deductions between the translations of agents is the logical rule of right introduction of the existential quantifier (otherwise said $A \vdash \exists x A$). In terms of process calculi, the rule $A \longrightarrow^+ \exists x A$ would enable any process to hang-up communication on any variable (channell), it would be a kind of uncontroled "time-out" that is dictated by the logic here.

On the other side, if we restrict ourselves to transitions $A \longrightarrow B$, where B does not contain the existential quantifier, we can refine the above result and prove that `cc` transitions then correspond exactly to deductions in INLL; it is an easy corollary of Lemma 3:

Theorem 4 *Let A and B be any `lcc` agents. If $A^\dagger \vdash_{INLL(\mathcal{C},\mathcal{D})} B^\dagger$ and if B contains no hiding (no existential quantifier), then $A \longrightarrow B$.*

5 Conclusion and Perspectives

We believe that the correspondence between concurrent constraint programming and non-commutative linear logic is a fundamental one. Besides the logical characterization of precious operational aspects of `cc` computations, the presented work offers new perspectives to the connection between concurrency and proof theory.

The intuition behind `cc` computations has served to define a new non-commutative linear logic which combines both commutative and non-commutative connectives. That logic thus extends the pure non-commutative (cyclic) linear logic of Girard and Yetter [9, 35], and differs from other proposals made by Retoré [25] to combine both kinds of connectives. A sequent calculus in the style of Retoré has been given. The existence of proof nets for this logic, and the proof of cut-elimination shows that this non-commutative linear logic is an interesting logic to study in its own right [27].

In particular the development of a phase semantics for NLL should provide in turn new denotational semantics of `cc` programs that capture finer observable properties of `cc` computations than the ones currently available [31, 12, 29]. Perrier [23] has considered a concurrent language based on linear logic and he proposed a denotational semantics based on the phase semantics, to model the interaction capability of a process. From both viewpoints of proof theory and concurrency, it would be thus worth developping the phase semantics of non-commutative linear logic (extending [1]) in connection with the present work on concurrent constraint programming. It should also be very interested to study the relationship between the execution of `cc` agents and the proof nets' syntax (which is intrisically parallel, thus naturally more suited for the representation of concurrent conputations than are sequent calculi).

The equivalence result between `ncc` transitions and INLL deductions suggests also a study of the usual semantics of concurrency, and specifically that of bisimulations, in the light of proof theory. The paper of Kobayashi and Yonezawa [13] is one of the most interesting works done so far to compare bisimulations to a logical semantics. We believe the strong correspondence between logic and concurrent agents established in the paper should encourage further investigation in this cross-fertilizating direction. The comparison of the true concurrency semantics developped in [22] with our approach provides another subject in this direction.

References

[1] M. Abrusci. Phase semantics and sequent calculus for pure non-commutative classical linear logic. *Journal of Symbolic Logic*, 56(4), 1991.

[2] J.M. Andreoli and R. Pareschi. Linear objects: logical processes with built-in inheritance. *New Generation Computing*, 9, 1991.

[3] A. Asperti. *Categorical Topics in Computer Science*. PhD thesis, Università di Pisa, March 1990.

[4] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96, 1992.

[5] E. Best, F.S. de Boer, and C. Palamidessi. Concurrent constraint programming with information retrieval. Esprit project ACCLAIM final report, 1994.

[6] F. Fages. Constructive negation by pruning. *To appear in J. of Logic Programming*, 1996.

[7] F.S. de Boer, M. Gabbrielli, and C. Palamidessi. Proving correctness of constraint logic programming with dynamic scheduling. In *Proceedings of SAS'96, Springer LNCS 1145*, 1996.

[8] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.

[9] J.Y. Girard. Towards a geometry of interaction. *Categories in Computer Science and Logic*, 1988.

[10] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language `cc(fd)`. In *Constraint Programming: Basics and Trends*, pages 293–316, 1995.

[11] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*. ACM, January 1987.

[12] R. Jagadeesan, V. Shanbhogue, and V.A. Saraswat. Angelic non-determinism in concurrent constraint programming. Technical report, Xerox Parc, 1991.

[13] N. Kobayashi and A. Yonezawa. Logical, testing and observation equivalence for processes in a linear logic programming. Technical Report 93-4, Dep. of Comp. Science, University of Tokyo, 1993.

[14] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, 3, 1994.

[15] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3), 1958.

[16] P. Lincoln and V.A. Saraswat. Proofs as concurrent processes. Manuscript, 1992.

[17] M. Maher and J. Jaffar. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20, 1994.

[18] M.J. Maher. Logic semantics for a class of committed-choice programs. In *Proceedings of ICLP'87, International Conference on Logic Programming*, 1987.

[19] N. Marti-Oliet and J. Meseguer. From petri nets to linear logic. In *Proceedings of Category Theory and Computer Science*, pages 313–340, Springer LNCS 389, 1989.

[20] D. Miller. The π-calculus as a theory in linear logic: preliminary results. In *Proceedings Workshop on Extensions of Logic Programming*, Springer LNCS 660, 1992.

[21] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1), 1992.

[22] U. Montanari and F. Rossi. Graph rewriting for a partial-ordering semantics of concurrent constraint programming. *Theoretical Computer Science*, 109:225–256, 1993.

[23] G. Perrier. Concurrent programming in linear logic. Technical Report CRIN 95-R-052, INRIA-Lorraine, 1995.

[24] A. Podelski and G. Smolka. Operational semantics of constraint logic programming with coroutining. In *Proceedings of ICLP'95, Int. Conf. on Logic Programming*, Tokyo, 1995.

[25] Ch. Retoré. *Réseaux et séquents ordonnés*. PhD thesis, Université Paris 7, 1993.

[26] P. Ruet. Logical semantics of concurrent constraint programming. In *Proceedings of CP'96, 2^{nd} International Conference on Constraint Programming, Cambridge, MA, Springer LNCS 1118*, 1996.

[27] P. Ruet. Non-commutative linear logic with mobilities. *Presented at the Logic Colloquium'96, San Sebastian, Spain*, LIENS Technical Report in preparation, 1996.

[28] V.A. Saraswat. Concurrent constraint programming. In *Proceedings 17th ACM Symposium on Principles of Programming Languages*, pages 232–245, 1990.

[29] V.A. Saraswat. *Concurrent constraint programming*. ACM Doctoral Dissertation Awards. MIT Press, 1993.

[30] V.A. Saraswat and P. Lincoln. Higher-order linear concurrent constraint programming. Manuscript, 1992.

[31] V.A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL'91: Proceedings 18th ACM Symposium on Principles of Programming Languages*, 1991.

[32] P.J. Stuckey. Constructive negation for constraint logic programming. *Information and Computation*, 118(1), 1995.

[33] C. Tse. The design and implementation of an actor language based on linear logic. Master Thesis, MIT, 1994.

[34] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge (MA), 1989.

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

Appendix: classical NLL

We recall here the definition of the formulas and the sequent calculus of the classical version of NLL (in [27], the logic was presented with two negations, a "right" negation and a "left" negation, like Abrusci's pure non-commutative linear logic [1]; here we find it simpler to present it with only one negation: this is a minor difference, which is merely the choice of cyclic linear logic [9, 35]).

Formulas are built from litterals $p, q, \ldots p^\perp, q^\perp, \ldots$, connectives:

- (multiplicative) *commutatives*: conjunction ($\otimes$) and disjunction ($\wp$),
- (multiplicative) *non-commutatives*: conjunction ($\times$) and disjunction ($+$),
- *addittives*: conjunction ($\&$) and disjunction ($\oplus$),
- *constants*: commutatives (1, $\perp$) and additives ($\top$, 0),

and *quantifiers*: universal ($\forall$) and existential ($\exists$).
As usual negation is defined by: $(p)^\perp = p^\perp$ et $(p^\perp)^\perp = p$, and De Morgan rules:

$$\begin{array}{cc} (A \otimes B)^\perp = B^\perp \wp A^\perp & (A \wp B)^\perp = B^\perp \otimes A^\perp \\ (A \times B)^\perp = B^\perp + A^\perp & (A + B)^\perp = B^\perp \times A^\perp \\ (A \& B)^\perp = B^\perp \oplus A^\perp & (A \oplus B)^\perp = B^\perp \& A^\perp \\ (\forall x\, A)^\perp = \exists x (A^\perp) & (\exists x A)^\perp = \forall x (A^\perp) \end{array}$$

$$i^\perp = e \qquad e^\perp = i \qquad 1^\perp = \perp \qquad \perp^\perp = 1 \qquad \top^\perp = 0 \qquad 0^\perp = \top$$

Negation is then an involution.

Sequents are of the form $\vdash \Gamma$ [i], where i is a simple oriented graph (at most one edge from a vertex to another) indexed by the multiset of formulas Γ. The definition of the operations and relations over i is the same as in the intuitionistic case.

The rules of the sequent calculus are:

Axiom / Cut

$$\vdash A^\perp, A \ [A^\perp \leftrightarrow A] \qquad \frac{\vdash \Gamma, A \ [\mathrm{i}] \qquad \vdash \Delta, A^\perp \ [\mathrm{j}]}{\vdash \Gamma, \Delta \ [\mathrm{i} \asymp_{A, A^\perp} \mathrm{j}]}$$

Commutatives

$$\frac{\vdash \Gamma, A \ [\mathrm{i}] \qquad \vdash B, \Delta \ [\mathrm{j}]}{\vdash \Gamma, A \otimes B, \Delta \ [\mathrm{i} \otimes_{A,B} \mathrm{j}]} \qquad \frac{\vdash \Gamma, A, B \ [\mathrm{i}]}{\vdash \Gamma, A \wp B \ [\mathrm{i}\{A \wp B / A, B\}]}$$

Non-commutatives

$$\frac{\vdash \Gamma, A \ [\mathrm{i}] \qquad \vdash B, \Delta \ [\mathrm{j}]}{\vdash \Gamma, A \times B, \Delta \ [\mathrm{i} \curlyvee_{A,B} \mathrm{j}]} \qquad \frac{\vdash \Gamma, A, B \ [\mathrm{i}]}{\vdash \Gamma, A + B \ [\mathrm{i}\{A + B / A, B\}]} \ A \prec_{\mathrm{i}} B$$

Additives

$$\frac{\vdash \Gamma, A\ [\mathrm{i}] \qquad \vdash \Gamma, B\ [\mathrm{i}\{B/A\}]}{\vdash \Gamma, A\&B\ [\mathrm{i}\{A\&B/A\}]} \qquad \frac{\vdash \Gamma, A\ [\mathrm{i}]}{\vdash \Gamma, A \oplus B\ [\mathrm{i}\{A \oplus B/A\}]} \qquad \frac{\vdash \Gamma, B\ [\mathrm{i}]}{\vdash \Gamma, A \oplus B\ [\mathrm{i}\{A \oplus B/B\}]}$$

Constants

$$\vdash 1\ [1 \leftrightarrow 1] \qquad \frac{\vdash \Gamma, A, B\ [\mathrm{i}]}{\vdash \Gamma, A, B, \bot\ [\mathrm{i}\{(A \leftrightarrow \bot \leftrightarrow B \leftrightarrow A)/(A \leftrightarrow B)\}]}\ A \leftrightarrow_{\mathrm{i}} B$$

$$\vdash \Gamma, \top\ [\top \leftrightarrow_{\mathrm{i}} \Gamma] \qquad \text{(no rule for 0)}$$

Quantifiers

$$\frac{\vdash \Gamma, A\ [\mathrm{i}]}{\vdash \Gamma \forall x A\ [\mathrm{i}\{\forall x A/A\}]}\ x \notin vl(\Gamma) \qquad \frac{\vdash \Gamma, A\ [\mathrm{i}]}{\vdash \Gamma \exists x A\ [\mathrm{i}\{\exists x A/A\}]}$$

$A \vdash B$ is defined by $\vdash A^{\perp}, B\ [A^{\perp} \leftrightarrow B]$. For instance $A \vdash A$.

Pure non-commutative cyclic LL [9, 35] ($\times$,+) and commutative LL ($\otimes$,$\wp$) are subsystems of ours. This logic enjoys cut-elimination and has also a presentation in terms of proof nets (and a correctness criterion for sequentialization).

Appendix: Proof of Lemma 3

By abuse of notation, we shall simply identify the agent A with its translation $A^{\dagger}$.

The *only if* part is a simple induction on $\equiv$ and $\Longrightarrow$.

For the *if* part, we first associate to a sequent $S = (\Gamma \vdash A\ [\mathrm{i}])$ the set $\widehat{S}$ of all the sequents $B \vdash A'\ [B \leftrightarrow A']$ which can be obtained from S by the rules $\otimes \vdash$ and $\vdash \rightsquigarrow$; more specifically, we define a relation $S \Rightarrow S'$ in the following way: if c is a constraint, $c \prec_{\mathrm{i}} A$ and $Gamma \neq \emptyset$, then $(\Gamma, c \vdash A\ [\mathrm{i}]) \Rightarrow (\Gamma \vdash c \rightsquigarrow A\ [\mathrm{i}\{c \rightsquigarrow A/c, A\}])$; moreover $(\Gamma, A, B \vdash C\ [\mathrm{i}]) \Rightarrow (\Gamma, A \otimes B \vdash C\ [\mathrm{i}\{A \otimes B/A, B\}])$ without any condition; now $(B \vdash A'\ [B \leftrightarrow A']) \in \widehat{S}$ iff $S = (\Gamma \vdash A\ [\mathrm{i}]) \Rightarrow^{*} (B \vdash A'\ [B \leftrightarrow A'])$.

We then prove, by induction on a (sequent calculus) proof of $S = (\Gamma \vdash_{INLL(\mathcal{C},\mathcal{D})} A\ [\mathrm{i}])$, that $(B \vdash A'\ [B \leftrightarrow A']) \in \widehat{S}$ implies $B \longrightarrow A'$.

First note that this induction makes sense because:
(1) if A and the formulas in Γ are translations of agents and $S = (\Gamma \vdash A\ [\mathrm{i}]) \Rightarrow^{*} (B \vdash A'\ [B \leftrightarrow A'])$, then B and A' are translations of agents as well;
(2) as a consequence of the cut-elimination theorem for INLL, the formulas eliminated by cut rules in an INLL($\mathcal{C}, \mathcal{D}$) proof are either constraints (cuts with some $c_1 \ldots c_n \vdash_{\mathcal{C}} c$ in $\mathcal{C}$) or agent names (cuts with some $p(\vec{x}) \vdash A$); so they are translations of agents as well, and therefore all formulas in the proof are translations of agents.

As we have already noticed (section 3), some deduction rules are read as transitions between `lcc` agents: the induction step is trivial for cuts, the rules for &, the left introduction of $\otimes$, and the right introductions of $\exists$ and $\rightsquigarrow$.

For axioms (logical $A \vdash A$ and $\vdash 1$, or non-logical: the axioms coming from $\mathcal{C}$ and $\mathcal{D}$), the point to quote is that $(1 \vdash c \rightsquigarrow c) \notin \widehat{c \vdash c}$, and this is justified because indeed $1 \not\vdash c \rightsquigarrow c$.

The only non-trivial cases are:

1. **the $\otimes$ right introduction:**

$$\frac{\Gamma \vdash A\ [\mathrm{i}] \qquad \Delta \vdash B\ [\mathrm{j}]}{\Gamma, \Delta \vdash A \otimes B\ [\mathrm{i} \otimes_{A,B} \mathrm{j}]}$$

When constructing $(\Gamma, \Delta \widehat{\vdash A} \otimes B)$, the formulas in Γ, Δ have to be "pasted" together according to the graph $\mathrm{i} \otimes_{A,B} \mathrm{j}$; by the definition of $\otimes_{A,B}$ and Lemma 1, the only way to get a formula $\prec A \otimes B$ is to paste the whole multiset Γ, Δ with $\otimes$'s. So $(\Gamma, \Delta \widehat{\vdash A} \otimes B) = (\Gamma, \Delta \vdash A \otimes B)$, and we conclude with the induction hypothesis and the monoidality of $\otimes$.

2. **the 1 left introduction:**

$$\frac{\Gamma \vdash C\ [\mathrm{i}]}{\Gamma, 1 \vdash C[\mathrm{i}\{(A \leftrightarrow 1 \leftrightarrow B \leftrightarrow A)/(A \leftrightarrow B)\}]}\ (A \leftrightarrow_{\mathrm{i}} B) \in (\Gamma, C)$$

In the construction of $(\Gamma, \widehat{1 \vdash} C)$, any formula, in particular A and B, has to be linked to the context via a $\otimes$ or an $\rightsquigarrow$. But the condition $A \leftrightarrow 1 \leftrightarrow B \leftrightarrow A$ imples that the only possibility is $\otimes$, and we get the result by $A \equiv A \otimes 1$.

3. **the $\rightsquigarrow$ left introduction:**

$$\frac{\Gamma \vdash A\ [\mathrm{i}] \qquad \Delta, B \vdash C\ [\mathrm{j}]}{\Delta, \Gamma, A \rightsquigarrow B \vdash C\ [\mathrm{i}\ \curlyvee_{A,B}\ \mathrm{j}]}$$

By examinating the possible cases in the construction of $(\Delta, \Gamma, A \widehat{\rightsquigarrow} B \vdash C)$, one can easily check the result, using the rules $(d \otimes c) \parallel (c \to A) \longrightarrow (d \parallel A)$ and $c \vdash_{\mathcal{C}} d \Rightarrow c \longrightarrow d$

4. **the $\exists$ left introduction:**

$$\frac{\Gamma, A \vdash B\ [\mathrm{i}]}{\Gamma, \exists x A \vdash B\ [\mathrm{i}\{\exists x A/A\}]}\ x \notin fv(\Gamma, B)$$

For an $S \in (\Gamma, \exists x \widehat{A} \vdash B)$ such that A occurs on the left ($S = A' \otimes \exists x A \vdash B'$) let us note that $x \notin fv(A', B')$ implies $A' \otimes \exists x A \equiv \exists x (A' \otimes A) \longrightarrow \exists x B'$ by induction hypothesis, $\equiv B'$, cqfd.

If A is a constraint and occurs on the right in some $S \in (\Gamma, \exists x \widehat{A} \vdash B)$, then we conclude by Lemma 2. ■