Concurrent Constraint Programming and Non-Commutative Logic

Paul Ruet*, François Fages

LIENS-CNRS, Ecole Normale Supérieure
45 rue d'Ulm, 75005 Paris, France
Phone : +33 1 44 32 20 83, Fax : +33 1 44 32 20 80
Email : {ruet,fages}@dmi.ens.fr

Abstract. This paper presents a connection between the intuitionistic fragment of a non-commutative version of linear logic introduced by the first author (NLI) and concurrent constraint programming (CC). We refine existing logical characterizations of operational aspects of CC, by providing a logical interpretation of finer observable properties of CC programs, namely stores, successes and suspensions.

1 Introduction

The class CC($\mathcal{X}$) of *Concurrent Constraint* programming languages introduced by Saraswat [29] in 1987 arises as a natural combination of constraint logic programming [11] and concurrent logic programming, with a synchronization mechanism based on constraint entailment [16]. CC programming is a model of concurrent computation, where 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 information c to the store (agent $tell(c)$), 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. The synchronization mechanism of CC languages gives an account for the co-routining facilities of implemented CLP systems, like the freeze predicate of Prolog, the delay mechanism of CLP($\mathbb{R}$) [11], or the constraint propagation schemes of CLP(FD). It also opens, to some extent, constraint programming to a new field of applications which are traditional in concurrent programming, like reactive systems and protocol specifications.

From the logic programming tradition however, the operational aspects of CC programming should also be closely connected to a logical semantics, via the *program as formula*, *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, 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, 16, 32, 6]. For instance, success constraints (i.e. final states of computations) can be observed logically :

* Current affiliation : McGill University. Post-doctoral fellowship from the INRIA.

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 $c_1 \dots c_n$, i.e. $\mathcal{C} \vdash \forall(c_1 \dots c_n \Rightarrow c)$. Such results make easier the design and understanding of programs, and provide useful tools for reasoning about them.

Maher in [16] was the first to suggest that the synchronization mechanism in concurrent logic programming could be given a logical interpretation. In [15] 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 $\Gamma \vdash A$ as "the agent Γ satisfies the test A". 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 $\Gamma \longrightarrow_{cc} (c_1 \wedge B_1) \vee ... \vee (c_n \wedge B_n)$ and for all $1 \leq i \leq n$ $\mathcal{C} \vdash \forall(c_i \rightarrow c)$.

However such a logical semantics does not capture 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 one succeeds, the second one loops and the third one 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 [9] 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 [3], 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 [20].

In [7, 26], a first-order non-monotonic variant, LCC (linear CC), is defined in which both stores and successes are characterized in ILL (intuitionistic linear logic). However suspensions cannot be characterized in ILL, because from

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

one cannot conclude that $A \parallel (c \rightarrow B)$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 $c \parallel c \rightarrow 1$ succeeds with 1, and does not suspend).

There is a lack of a "sequential" connective, that is a non-commutative one. The Pomset logic of Retoré [25] combines commutativity and non-commutativity, however it does not provide an immediate solution to our problem, since the non-commutative *before* connective $<$ enjoys the same 'porosity' property as linear implication : $A \otimes (c < B) \vdash c < (A \otimes B)$.

In this paper we show that the intuitionistic fragment of a mixed non-commutative version of linear logic (NLI) copes with this difficulty (Section 4) : this logic combines both commutative and non-commutative connectives, and is based on previous proposals of de Groote [5] and of the first author [27]. Its classical version, which extends commutative linear logic on one hand and the cyclic non-commutative linear logic of Girard and Yetter [10, 33] on the other hand, is presented in [28]. Here we just consider an intuitionistic fragment : NLI.

We show that the stores, the successes and the suspensions of an LCC computation can be characterized in NLI (Section 6) and that these results hold for usual monotonic CC as well.

2 Preliminaries on concurrent constraint programming

2.1 Monotonic CC

Definition 1 (Intuitionistic constraint system) *A* constraint system *is a pair* $(\mathcal{C}, \vdash_{\mathcal{C}})$*, where :*

- $\mathcal{C}$ *is a set of formulas (the* constraints*) built from a set* V *of variables, a set* Σ *of function and relation symbols, with logical operators :* 1 *(true), the conjunction* $\wedge$ *and the existential quantifier* $\exists$*;*
- $\Vdash$ *is a subset of* $\mathcal{C} \times \mathcal{C}$*, which defines the non-logical axioms of the constraint system. Instead of* $(c, d) \in \Vdash_{\mathcal{C}}$*, we write* $c \Vdash_{\mathcal{C}} d$*.*
- $\vdash_{\mathcal{C}}$ *is the least subset of* $\mathcal{C}^{\star} \times \mathcal{C}$ *containing* $\Vdash_{\mathcal{C}}$ *and closed by the rules of intuitionistic logic (IL) for* 1*,* $\wedge$ *and* $\exists$*.*

In the following, $c, d, e \ldots$ will denote constraints. Note that the intuitionistic logical framework (rather than the classical one) is not essential, it is simply sufficient, taking into account that the constraints are only built from conjunctions and existential quantifications.

Definition 2 (Syntax) *The syntax of CC* agents *is given by the following grammar:*

$$A ::= p(\boldsymbol{x}) \mid tell(c) \mid (A \parallel A) \mid A + A \mid \exists x A \mid c \rightarrow A$$

where 1 *stands for inaction,* $\parallel$ *for parallel composition,* $+$ *for non-deterministic choice,* $\exists$ *for variable hiding and* $\rightarrow$ *for blocking ask. The atomic agents* $p(\boldsymbol{x}) \ldots$ *are called* process names *or* procedure names.

Recursion is obtained by declarations:

$$D ::= \epsilon \mid p(\boldsymbol{x}) = A \mid D, D$$

We make the very natural hypothesis that in a declaration $p(\boldsymbol{x}) = A$, all the free variables occurring in A have a free occurrence in p, and that in a relation $c_1 \dots c_n \Vdash_C c$, all the free variables occurring in c have a free occurrence in $c_1 \dots c_n$. Notice that this is exactly the meaning associated with the Horn clauses in the logic programming languages, for instance : the variables which are free in the body but not in the head are considered (implicitly in the syntax, explicitly in the semantics) as existentially quantified in the body (because universally in the clause).

The operational semantics is defined on configurations (rather than agents) where the store is distinguished from agents:

Definition 3 (Configurations) *A* configuration *is a triple* $(\boldsymbol{x}; c; \Gamma)$, *where* $\boldsymbol{x}$ *is a set of variables,* c *is a constraint, and* Γ *a multi-set of agents.*

In a configuration $(\boldsymbol{x}; c; \Gamma)$, $\boldsymbol{x}$ denotes the set of *fresh* variables and c is the store. $\mathcal{A}$ (resp. $\mathcal{B}$) denotes the set of agents (resp. configurations).

The semantics is defined by a transition system which does not take into account specific evaluation strategies. The transition system is given in the style of the CHAM [2] (see also [24]). A congruence relation between configurations is thus distinguished from the very transition relation (between configurations). In this presentation the constraint system is implicit.

Definition 4 *The* structural congruence $\equiv$ *is the least congruence such that:*

Parallel composition	$(\boldsymbol{x}; c; A \parallel B, \Gamma) \equiv (\boldsymbol{x}; c; A, B, \Gamma)$	
Inaction	$(\boldsymbol{x}; c; 1, \Gamma) \equiv (\boldsymbol{x}; c; \Gamma)$	
α-Conversion	$(\boldsymbol{x}, y; c; \Gamma) \equiv (\boldsymbol{x}, z; c[z/y]; \Gamma[z/y])$	
Hiding	$\dfrac{y \notin fv(c, \Gamma) \cup \boldsymbol{x}}{(\boldsymbol{x}; c; \exists y A, \Gamma) \equiv (\boldsymbol{x}, y; c; A, \Gamma)}$	$\dfrac{y \notin fv(c, \Gamma)}{(\boldsymbol{x}, y; c; \Gamma) \equiv (\boldsymbol{x}; c; \Gamma)}$

The transition *relation* $\longrightarrow$ *is the least transitive relation such that:*

Tell	$(\boldsymbol{x}; c; tell(d), \Gamma) \longrightarrow (\boldsymbol{x}; c \wedge d; \Gamma)$
Ask	$\dfrac{c \vdash_{\mathcal{C}} d}{(\boldsymbol{x}; c; d \rightarrow A, \Gamma) \longrightarrow (\boldsymbol{x}; c; A, \Gamma)}$
Declarations	$\dfrac{(p(\boldsymbol{y}) = A) \in P}{(\boldsymbol{x}; c; p(\boldsymbol{y}), \Gamma) \longrightarrow (\boldsymbol{x}; c; A, \Gamma)}$
Congruence	$\dfrac{(\boldsymbol{x'}; c'; \Gamma') \equiv (\boldsymbol{x}; c; \Gamma) \longrightarrow (\boldsymbol{y}; d; \Delta) \equiv (\boldsymbol{y'}; d'; \Delta')}{(\boldsymbol{x'}; c'; \Gamma') \longrightarrow (\boldsymbol{y'}; d'; \Delta')}$
Non-determinism (blind choice)	$(\boldsymbol{x}; c; A + B, \Gamma) \longrightarrow (\boldsymbol{x}; c; A, \Gamma)$ $(\boldsymbol{x}; c; A + B, \Gamma) \longrightarrow (\boldsymbol{x}; c; B, \Gamma)$

As usual, the precise operational semantics depends on the choice of observables. We shall consider the stores, successes and suspensions:

Definition 5 (Observables) *The* store *of a configuration $(\boldsymbol{x}; c; \Gamma)$ is the constraint $\exists \boldsymbol{x} c$. We shall say that $\exists \boldsymbol{x} c$ is accessible from the agent A if there exists a multi-set of agent Γ such that $(\emptyset; 1; A) \longrightarrow (\boldsymbol{x}; c; \Gamma)$.*

A success *for an agent A is a constraint c such that $(\emptyset; 1; A) \longrightarrow (\emptyset; c; 1)$;*

A suspension *for A is a configuration $(\boldsymbol{x}; c; d_1 \rightarrow A_1, \dots, d_n \rightarrow A_n)$ such that $(\emptyset; 1; A) \longrightarrow (\boldsymbol{x}; c; d_1 \rightarrow A_1, \dots, d_n \rightarrow A_n)$ and for no i, $c \vdash_{\mathcal{C}} d_i$.*

Note that the non-deterministic agent $A + B$, can behave either like A or like B. The operator of non-deterministic choice $+$, called *blind choice*, is thus different from the *one-step guarded choice* defined by:

$$\frac{A \longrightarrow A'}{A + B \longrightarrow A'} \quad \text{and} \quad \frac{B \longrightarrow B'}{A + B \longrightarrow B'} \quad .$$

As remarked in [12, 8] the difference between (backtracking) non-determinism and (committed-choice) in-determinism arises in the way observations are defined, more than the way transitions are defined. However backtracking non-determinism generally refers to the blind choice rule, which is the only choice rule considered in this paper. (In-determinism definitely requires more material, namely proofs semantics, at least for the observation of suspensions, and this goes beyond the scope of the present paper.)

An essential property of this calculus is that the execution is *monotonic* : the informations contained in the store are not consumed by the communication rule (*ask*):

Proposition 6 (Monotonicity of the store [31])
– *If $(\boldsymbol{x}; c; \Gamma) \longrightarrow (\boldsymbol{y}; d; \Delta)$ then $\exists \boldsymbol{y} d \vdash_{\mathcal{C}} \exists \boldsymbol{x} c$.*

Proposition 7 (Monotonicity of the transitions [31])
– If $(\boldsymbol{x}; c; \Gamma) \longrightarrow (\boldsymbol{y}; d; \Delta)$, then for every multi-set of agents Σ and every constraint e, $(\boldsymbol{x}; c \wedge e; \Gamma, \Sigma) \longrightarrow (\boldsymbol{y}; d \wedge e; \Delta, \Sigma)$.

These properties provide CC with a denotational semantics where the agents are seen as closure operators on the semi-lattice of constraints [31, 12]. The property of monotonicity can however be dropped from CC programming, by considering linear constraint systems where constraints are formulas in linear logic [9].

2.2 Linear CC

Non-monotonic variants of CC have been introduced by Saraswat and Lincoln [30], then further studied in [3].

As for the monotonic CC, we define the constraint systems, the agents, the configurations and the transition system.

Definition 8 (Linear constraint system) *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, with logical operators : 1, the multiplicative conjunction $\otimes$, the existential quantifier $\exists$ and the exponential connective $!$;*
- *$\Vdash_{\mathcal{C}}$ is a subset of $\mathcal{C} \times \mathcal{C}$ which defines the non-logical axioms of the constraint system. 1 is the neutral element of $\otimes$.*
- *$\vdash_{\mathcal{C}}$ is the least subset of $\mathcal{C}^\star \times \mathcal{C}$ containing $\Vdash_{\mathcal{C}}$ and closed by the following rules ($fv(A)$ denotes the set of free variables occurring in A):*

$$c \vdash c \qquad \frac{\Gamma, c \vdash d \qquad \Delta \vdash c}{\Gamma, \Delta \vdash d} \qquad \vdash 1 \qquad \frac{\Gamma \vdash c}{\Gamma, 1 \vdash c}$$

$$\frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c} \qquad \frac{\Gamma \vdash c_1 \qquad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \qquad \frac{\Gamma \vdash c}{\Gamma \vdash \exists x c} \qquad \frac{\Gamma, c \vdash d}{\Gamma, \exists x c \vdash d}\ x \notin fv(\Gamma, d)$$

$$\frac{\Gamma, c \vdash d}{\Gamma, !c \vdash d} \qquad \frac{!\Gamma \vdash d}{!\Gamma \vdash !d} \qquad \frac{\Gamma \vdash d}{\Gamma, !c \vdash d} \qquad \frac{\Gamma, !c, !c \vdash d}{\Gamma, !c \vdash d}$$

These are the rules of intuitionistic linear logic (ILL) for 1, $\otimes$, $\exists$ and $!$ (see [9]).

The definition of LCC *agents*[1], *declarations* and *configurations* is the same as in the monotonic case (Section 2.1), and we assume again that in a declaration

[1] We have chosen to limitate the use of ! to constraints only, because the usual replication operator of process calculi (like the π-calculus [20], where it is also noted !) does not have the same behavior as the exponential connective : it allows replication $(!A \longrightarrow (!A \,||\, !A))$ but not erasing $(!A \not\longrightarrow 1)$.

$p(\boldsymbol{x}) = A$, all the free variables occurring in A have a free occurrence in p, and that in a relation $c_1 \ldots c_n \Vdash_{\mathcal{C}} c$, all the free variables occurring in c have a free occurrence in $c_1 \ldots c_n$.

Definition 9 *The* structural congruence $\equiv$ *is the same as for monotonic CC (Definition 4). The* transition *relation* $\longrightarrow$ *is defined by the same rules as for monotonic CC (Definition 4), except for* **Tell** *and* **Ask** *:*

Tell $\quad (\boldsymbol{x}; c; tell(d), \Gamma) \longrightarrow (\boldsymbol{x}; c \otimes d; \Gamma)$

Ask $\quad \dfrac{c \vdash_{\mathcal{C}} d \otimes e}{(\boldsymbol{x}; c; e \to A, \Gamma) \longrightarrow (\boldsymbol{x}; d; A, \Gamma)}$

The only differences compared to monotonic CC is that constraints are formulas of linear logic and that the communication rule *ask* consumes information. The calculus is thus intrinsically non-deterministic, even without the choice operator $+$, since several constraints can satisfy the condition of the rule. Of particular interest in this context (see also section 3) are the *synchronization constraints* which are linear atomic constraints without entailment.

Definition 10 (Synchronization constraints) *Given a linear constraint system* $(\mathcal{C}, \Vdash_{\mathcal{C}})$*, a set* $\mathcal{C}_s$ *of* synchronization constraints *is a set of atomic formulas of* $\mathcal{C}$ *not occurring in* $\Vdash_{\mathcal{C}}$*.*

A well-formed constraint *is a (tensor) product of synchronization constraints and of constraints not containing a synchronization constraint as a subformula.*

An agent or a configuration is said to be well-formed *if all the constraints in it are well-formed.*

We call these constraints synchronization constraints to indicate that it is this type of constraints which is considered in CC programs of protocol specification. They are the constraints that allow one to translate into CC process calculi : for instance the asynchronous version of Milner's π-calculus [20, 4] can be translated in a LCC language [30] with the atomic formulas $x : y$ as constraints, where x and y are variables and : is a predicate symbol ($x : y$ means : "the channel name x carries the channel name y), and these constraints are indeed synchronization constraints. The dining philosophers (cf. section 2.3) provide another example.

Another worthnoting property of synchronization constraints is that an *ask* with a synchronization constraint is deterministic.

Because constraints are linear formulas, we must slightly modify the definition of stores and suspensions :

Definition 11 (Observables) *The* successes *are defined as in the monotonic case.*

A store *accessible from* A *is a constraint* d *such that there exist a constraint* c *and a multiset* Γ *of agents such that* $(\emptyset; 1; A) \longrightarrow (\emptyset; c; \Gamma)$ *and* $c > d$*, where* *the relation between constraints "$c > d$" is the least relation containing $\vdash_{\mathcal{C}}$ and closed by the rule $\forall e \in \mathcal{C}(c > d \;\Rightarrow\; (c \otimes e) > d)$.*

A suspension *for A is a configuration $(\boldsymbol{x}; c; d_1 \to A_1, \dots, d_n \to A_n)$ such that $(\emptyset; 1; A) \longrightarrow (\boldsymbol{x}; c; d_1 \to A_1, \dots, d_n \to A_n)$ and for no i, $c > d_i$. An agent A* suspends with the store c on the constraints $d_1, ..., d_n$, *if there exist a suspension for A of the form $(\boldsymbol{x}; c; d_1 \to A_1, \dots, d_n \to A_n)$.*

2.3 Example

A classical benchmark of expressiveness for concurrent languages is the dining philosophers : n philosophers are sitting around a table and alternate thinking and eating. Each one of them has a fork on his right, and thus also on his left, and needs these two forks to eat (the chop-sticks version may be more realistic).

As suggested in [3, 22], this problem has an extremely simple solution in LCC. All constraints are atomic : either $\texttt{fork}_i$, or $\texttt{eat}_i$ $(1 \le i \le n)$, or `ticket`.

$$
\begin{array}{l}
\texttt{philosopher}_i \texttt{ = ticket} \to \texttt{fork}_i \to \texttt{fork}_{i+1 \bmod n} \to \\
\qquad\qquad\qquad (\texttt{tell(eat}_i\texttt{)} \parallel \\
\qquad\qquad\qquad \texttt{eat}_i \to (\texttt{tell(fork}_i\texttt{)} \parallel \texttt{tell(fork}_{i+1 \bmod n}\texttt{)} \parallel \\
\qquad\qquad\qquad\qquad \texttt{tell(ticket)} \parallel \texttt{philosopher}_i\texttt{)).}
\end{array}
$$

$$
\begin{array}{l}
\texttt{init = } \underbrace{\texttt{tell(ticket)} \parallel \cdots \parallel \texttt{tell(ticket)}}_{n-1\ \ times} \parallel \\
\qquad \texttt{tell(fork}_1\texttt{)} \parallel \cdots \parallel \texttt{tell(fork}_n\texttt{)} \parallel \\
\qquad \texttt{philosopher}_1 \parallel \cdots \parallel \texttt{philosopher}_n\texttt{.}
\end{array}
$$

This program enjoys safety and liveness properties : two adjacent philosophers cannot eat at the same time, at least one philosopher can eat, the program is suspension-free (absence of deadlock).

2.4 Translation from CC to LCC

The LCC languages are a refinement of usual monotonic CC. Indeed the monotonicity of CC can simply be restored with the exponential connective ! of linear logic, allowing replication of hypotheses and thus avoiding constraint consumption during an application of the *ask* rule :

Definition 12 *Let $(\mathcal{C}, \Vdash_{\mathcal{C}})$ be an intuitionistic constraint system. We define the* translation *of $(\mathcal{C}, \Vdash_{\mathcal{C}})$, which is the linear constraint system $(\mathcal{C}, \Vdash_{\mathcal{C}})^\circ$, as follows, at the same time as the translation of CC agents to LCC agents :*

$$
\begin{array}{rr}
\multicolumn{2}{c}{c^\circ = !c, \textit{ if } c \textit{ is an atomic constraint}} \\
(c \wedge d)^\circ = c^\circ \otimes d^\circ & (\exists x c)^\circ = \exists x c^\circ \\
\mathit{tell}(c)^\circ = \mathit{tell}(c^\circ) & p(\boldsymbol{x})^\circ = p(\boldsymbol{x}) \\
(A \parallel B)^\circ = A^\circ \parallel B^\circ & (A + B)^\circ = A^\circ + B^\circ \\
(c \to A)^\circ = c^\circ \to A^\circ & (\exists x A)^\circ = \exists x A^\circ
\end{array}
$$

The entailment relation $\Vdash_{\mathcal{C}}^{\circ}$ is defined by : $c \Vdash_{\mathcal{C}} d$ iff $c^{\circ} \Vdash_{\mathcal{C}}^{\circ} d^{\circ}$. The relation ${\vdash_{\mathcal{C}}}^{\circ}$ is obtained from ${\Vdash_{\mathcal{C}}}^{\circ}$ by the rules of linear logic for 1, $!$, $\otimes$ and $\exists$. The translation of a CC configuration $(\boldsymbol{x}; c; \Gamma)$ is the LCC configuration $(\boldsymbol{x}; c^{\circ}; \Gamma^{\circ})$. The transition relation $\longrightarrow^{\circ}$ is the one of LCC.

For constraints, the above translation is a well-known translation of intuitionistic logic into linear logic [9, p.81], hence :

Proposition 13 *Let $(\mathcal{C}, \Vdash_{\mathcal{C}})$ be an intuitionistic constraint system, and c, d be intuitionistic constraints : $c \vdash_{\mathcal{C}} d$ iff $c^{\circ} {\vdash_{\mathcal{C}}}^{\circ} d^{\circ}$.*

from which follows the soundness of the translation :

Proposition 14 *Let $(\boldsymbol{x}; c; \Gamma)$ and $(\boldsymbol{y}; d; \Delta)$ be CC configurations :*
(i) $(\boldsymbol{x}; c; \Gamma) \equiv (\boldsymbol{y}; d; \Delta)$ iff $(\boldsymbol{x}; c^{\circ}; \Gamma^{\circ}) \equiv^{\circ} (\boldsymbol{y}; d^{\circ}; \Delta^{\circ})$;
(ii) if $(\boldsymbol{x}; c; \Gamma) \longrightarrow (\boldsymbol{y}; d; \Delta)$ then $(\boldsymbol{x}; c^{\circ}; \Gamma^{\circ}) \longrightarrow^{\circ} (\boldsymbol{y}; d^{\circ}; \Delta^{\circ})$;
(iii) if $(\boldsymbol{x}; c^{\circ}; \Gamma^{\circ}) \longrightarrow^{\circ} (\boldsymbol{y}; d^{\circ}; \Delta^{\circ})$ then $(\boldsymbol{x}; c; \Gamma) \longrightarrow (\boldsymbol{y}; e; \Delta)$, with $e \vdash d$.

3 Observing suspensions : preliminary remarks

There are some obvious general obstacles to the logical observation of suspensions, namely :

$$(1) \quad d \otimes (c \multimap d) \otimes (d \multimap A) \vdash d \otimes (c \multimap A) \qquad (2) \quad \frac{c \vdash A}{1 \vdash c \multimap A}$$

and

$$(3) \quad \frac{c \vdash c'}{c \otimes (d \multimap A) \vdash c' \otimes (d \multimap A)} \qquad (4) \quad \frac{d' \vdash d}{c \otimes (d \multimap A) \vdash c \otimes (d' \multimap A)}$$

Obstacles (3) and (4) bear on the nature of the constraints considered, and suggest naturally to restrict ourselves to characterize the suspensions for which the blocking constraints are elementary informations : namely synchronization constraints (definition 10). Note that this does not prevent us from using more general constraints in the program : we will just not be interested in suspensions on those *non*-synchronization constraints.

Consider now the first implication : it constitutes an obstacle to the characterization of suspensions if one chooses to translate the agent $c \rightarrow A$ by $c \multimap A$: then indeed, if $d \nvdash c$, $d \otimes (c \multimap A)$ looks like a suspension, whereas the agent translated by $d \otimes (c \multimap d) \otimes (d \multimap A)$ can have a success (if A unblocks c) and not suspend. The simplest idea is thus to translate $c \rightarrow A$ by $c \multimap (\diamond \otimes A)$, where $\diamond$ is a new atomic formula (neither a constraint, nor a procedure name). Intuitively $\diamond$ ensures that the communication between agents happens through the store, and avoids this type of "composition of suspensions". At the same time this translation of ask prevents from obstacle (2).

The obstacle

$$A \otimes (c \multimap B) \vdash c \multimap (A \otimes B),$$

exposed in the introduction, is more serious and leads to non-commutative logic.

4 Intuitionistic non-commutative logic

We just present here an intuitionistic fragment of this logic, which we call NLI. Also not all the connectives are considered here. The complete presentation of this non-commutative logic, based on previous proposals [5, 27], is the topic of [28].

The set $\mathcal{F}$ of formulas is built from atoms $p, q, \ldots$, the *constant* 1, the *existential quantifier* $\exists$ and connectives : a multiplicative commutative conjunction *tensor* $\otimes$, a left non-commutative implication $\multimap\!\!\bullet$, the additive conjunction *with* &, and the exponential connective !.

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 $\multimap\!\!\bullet$). Sequents are of the form $\Gamma \vdash A$, where $A \in \mathcal{F}$ and $\Gamma \in \mathcal{H}$, and $\mathcal{H}$ (resp. $\mathcal{H}_0$), the sets of *contexts* (resp. *non-empty contexts*) are defined by the following grammar :

- $\mathcal{H} ::= () \mid \mathcal{H}_0$
- $\mathcal{H}_0 ::= \mathcal{F} \mid (\mathcal{H}_0 , \mathcal{H}_0) \mid (\mathcal{H}_0 ; \mathcal{H}_0)$

$\Gamma, \Delta \ldots$ will denote (possibly empty) contexts. We use the notation $\Gamma[\ ]$ to denote a context with a hole, and $\Gamma[\Delta]$ is the context obtained by "filling" the hole with Δ. We use the notation $!\Gamma$ for any context whose formulas are all under a !.

For sake of simplicity, we will assume the set of contexts quotiented by the *associativity* of "," and ";" $(\Gamma,(\Delta,\Sigma)) = ((\Gamma,\Delta),\Sigma)$ and $(\Gamma;(\Delta;\Sigma)) = ((\Gamma;\Delta);\Sigma)$, and the *commutativity* of "," : $(\Gamma,\Delta) = (\Delta,\Gamma)$[2].

The rules of the sequent calculus for NLI are :

Axiom - Cut

$$A \vdash A \qquad \vdash 1 \qquad \frac{\Gamma \vdash A \qquad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}$$

Entropy

$$\frac{\Gamma[\Delta;\Sigma] \vdash A}{\Gamma[\Delta,\Sigma] \vdash A}$$

Logical rules

$$\frac{\Gamma[\Delta] \vdash A}{\Gamma[\Delta;1] \vdash A} \qquad \frac{\Gamma[\Delta] \vdash A}{\Gamma[1;\Delta] \vdash A}$$

$$\frac{\Gamma[A,B] \vdash C}{\Gamma[A \otimes B] \vdash C} \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma,\Delta \vdash A \otimes B}$$

[2] The quotient of the set of sequents by associativity and commutativity is therefore the set of series-parallel orders labeled by formulas (see, e.g., [21] for a survey on series-parallel orders). Note that it is not the case in *classical* NL, where the quotient of sequents by the reversible structural rules is a set of *cyclic* orders labeled by formulas (see [28]).

$$\frac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[\Gamma ; A \mathrel{-\!\!\bullet} B] \vdash C} \qquad \frac{A ; \Gamma \vdash B}{\Gamma \vdash A \mathrel{-\!\!\bullet} B}$$

$$\frac{\Gamma[A] \vdash C}{\Gamma[A \& B] \vdash C} \qquad \frac{\Gamma[B] \vdash C}{\Gamma[A \& B] \vdash C} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \& B}$$

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

$$\frac{\Gamma[A] \vdash B}{\Gamma[!A] \vdash B} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A} \qquad \frac{\Gamma[!\Delta , !\Sigma] \vdash C}{\Gamma[!\Delta ; !\Sigma] \vdash C}$$

$$\frac{\Gamma[!A , !A] \vdash B}{\Gamma[!A] \vdash B} \qquad \frac{\Gamma[\Delta] \vdash B}{\Gamma[\Delta ; !A] \vdash B} \qquad \frac{\Gamma[\Delta] \vdash B}{\Gamma[!A ; \Delta] \vdash B}$$

The structure of sequents enables to express sequentiality constraints on formulas, specifically in the rules for $\otimes$ and $\mathrel{-\!\!\bullet}$.

In contrast to the purely commutative case however, one cannot prove $A \otimes (c \mathrel{-\!\!\bullet} B) \vdash c \mathrel{-\!\!\bullet} (A \otimes B)$ in general, and this is why we will be able to solve the problem of 'porosity' of suspensions exposed in the introduction.

The rules for & and $\exists$ are self-evident. The rules for ! express the fact that !-formulas should commute ($!A \odot !B \cong !(A \& B)$).

Theorem 15 *The sequent calculus for NLI enjoys cut-elimination.*

This can be proved as a consequence of the completeness of the phase semantics [28] : the cut rule is sound and the cut-free calculus is complete.

5 Translation of agents into formulas

One translates LCC agents and configurations into formulas of intuitionistic non-commutative logic, more precisely in the fragment of NLI with the constant 1, the connectives &, $\otimes$ and $\mathrel{-\!\!\bullet}$, and the quantifier $\exists$.

Fix a linear constraint system $(\mathcal{C}, \Vdash_{\mathcal{C}})$ and a set of declarations $\mathcal{D}$. Fix also a set $\mathcal{C}_s$ of synchronization constraints. One makes henceforth the hypothesis that the constraints of $\Vdash_{\mathcal{C}}$ and the agents in the declarations of $\mathcal{D}$ are all well-formed. (This hypothesis is in fact only used for the observation of suspensions.)

The fact of being a well-formed configuration is then preserved during the execution :

Lemma 16 *A subformula of a well-formed constraint is a well-formed constraint.*

A subagent of a well-formed agent is a well-formed agent.

If κ is a well-formed LCC configuration and $\kappa \longrightarrow \kappa'$, then κ' is well-formed.

As it is suggested in the previous paragraph, one introduces a new atomic formula $\diamond$, considered neither as a constraint, nor as a procedure name. $\vec{\diamond}$ denotes the product of n occurrences of $\diamond$ (n arbitrary ≥ 0) : $\diamond \otimes \cdots \otimes \diamond$.

Definition 17 *LCC agents are translated into formulas of (intuitionistic) non-commutative logic as follows :*

$$\begin{array}{rclrcl} & & c^* = c, \textit{ if } c \textit{ is a constraint} & & & \\ tell(c)^* & = & c & p(\boldsymbol{x})^* & = & p(\boldsymbol{x}) \\ (c \to A)^* & = & c \mathrel{-\!\!\bullet} (\diamond \otimes A^*) & (A \parallel B)^* & = & A^* \otimes B^* \\ (A + B)^* & = & A^* \& B^* & (\exists x A)^* & = & \exists x A^* \end{array}$$

If Γ is the multi-set $(A_1 \dots A_n)$ of agents, define $\Gamma^* = A_1^* \otimes \cdots \otimes A_n^*$. The translation $(\boldsymbol{x}; c; \Gamma)^*$ of a configuration $(\boldsymbol{x}; c; \Gamma)$ is the formula $\exists \boldsymbol{x}(c \otimes \Gamma^*)$.

NLI($\mathcal{C}$,$\mathcal{D}$) denotes the deduction system obtained by adding to NLI :

– the non-logical axiom $c \vdash d$ for every $c \Vdash_{\mathcal{C}} d$ in $\Vdash_{\mathcal{C}}$,
– the non-logical axiom$p(\boldsymbol{x}) \vdash A^*$ for every declaration $p(\boldsymbol{x}) = A$ in $\mathcal{D}$.

Theorem 18 (Soundness) *Let $(\boldsymbol{x}; c; \Gamma)$ and $(\boldsymbol{y}; d; \Delta)$ be LCC configurations.*
If $(\boldsymbol{x}; c; \Gamma) \equiv (\boldsymbol{y}; d; \Delta)$ then $(\boldsymbol{x}; c; \Gamma)^* \dashv\vdash_{NLI(\mathcal{C},\mathcal{D})} (\boldsymbol{y}; d; \Delta)^*$.
If $(\boldsymbol{x}; c; \Gamma) \longrightarrow (\boldsymbol{y}; d; \Delta)$ then $(\boldsymbol{x}; c; \Gamma)^* \vdash_{NLI(\mathcal{C},\mathcal{D})} (\boldsymbol{y}; d; \Delta)^* \otimes \vec{\diamond}$.

Proof. By induction on $\equiv$ and $\longrightarrow$.
– For *parallel composition*, the *α-conversion*, it is immediate.
– For *inaction*, remark that $A^\dagger \otimes 1 \dashv\vdash A^\dagger$.
– For *local variables*, $\exists x(A \otimes B) \dashv\vdash A \otimes \exists x B$ and $\exists x A \dashv\vdash A$ whence $x \notin fv(A)$.
– For *Tell*, $\equiv$ and the *declarations*, it is immediate.
– For *Ask*, we have $c \otimes (d \mathrel{-\!\!\bullet} (\diamond \otimes A)) \vdash e \otimes \diamond \otimes A$ if $c \Vdash_{\mathcal{C}} d \otimes e$:

$$\dfrac{\dfrac{c \vdash d \otimes e \qquad \dfrac{\dfrac{\dfrac{\dfrac{d \vdash d \qquad \dfrac{e \vdash e \qquad \diamond \otimes A \vdash \diamond \otimes A}{e, \diamond \otimes A \vdash e \otimes \diamond \otimes A}}{e, (d; d \mathrel{-\!\!\bullet} (\diamond \otimes A)) \vdash e \otimes \diamond \otimes A}}{e, d, d \mathrel{-\!\!\bullet} (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{d, e, d \mathrel{-\!\!\bullet} (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{d \otimes e, d \mathrel{-\!\!\bullet} (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{c, d \mathrel{-\!\!\bullet} (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{c \otimes (d \mathrel{-\!\!\bullet} (\diamond \otimes A)) \vdash e \otimes \diamond \otimes A}$$

■

6 Logical characterization of LCC stores, successes and suspensions

Definition 19 (Tests) *Let us call a* test *ϕ :*
– a procedure name,
– or an agent $c \otimes (d_1 \mathrel{-\!\!\bullet} A_1) \otimes \cdots \otimes (d_n \mathrel{-\!\!\bullet} A_n)$, $n \geq 0$, such that c is a constraint, and for every $i (0 \leq i \leq n)$ d_i is a synchronization constraint and for every $i (0 \leq i \leq n)$ $c \not> d_i$, where the relation $>$ between linear constraints is the

one defined in Definition 11. In the last case we shall say that the test ϕ is a suspension*, even if $n = 0$, in other terms one considers a success as a particular case of suspension.*

Let κ be an LCC configuration, and ϕ a test. We shall say that κ satisfies the test *ϕ (denoted by $\kappa \overset{\triangleright}{\longrightarrow} \phi$) under the following conditions :*
– if ϕ is a procedure name : "there exists a configuration $(\boldsymbol{y}; d; \phi)$, such that $vl(\phi) \cap \boldsymbol{y} = \emptyset$, $d \vdash_{\mathcal{C}} 1$ and $\kappa \longrightarrow (\boldsymbol{y}; d; \phi)$",
– if $\phi = c \otimes (d_1 \multimap A_1) \otimes \cdots \otimes (d_n \multimap A_n)$ is a suspension : "there exists a configuration

$$\kappa' = (\boldsymbol{y}; d; d_1 \to B_1, \ldots, d_n \to B_n),$$

such that $\kappa \longrightarrow \kappa'$, $d \vdash_{\mathcal{C}} c$ and for every $i (0 \leq i \leq n)$: $d \not\vdash d_i$".

Lemma 20 *Let κ and κ' be two LCC configurations such that $\kappa^* = \kappa'^*$, and ϕ a procedure name or a suspension.*
$\kappa \overset{\triangleright}{\longrightarrow} \phi$ iff $\kappa' \overset{\triangleright}{\longrightarrow} \phi$.

Lemma 21 *If c is a synchronization constraint, A a translation of agent and Γ and Δ two contexts whose formulas are translations of agents, then $c\mathbf{;}(\diamond\mathbf{,}\Delta)\mathbf{;}\Gamma \nvdash_{NLI(\mathcal{C},\mathcal{D})} \diamond \otimes A$.*

Proof. It is almost evident by absurdum : one shows by induction that if there is a proof of $c\mathbf{;}(\diamond\mathbf{,}\Delta)\mathbf{;}\Gamma \vdash_{NLI(\mathcal{C},\mathcal{D})} \diamond \otimes A$, then c is not a synchronization constraint.

The last applied rule is a left unary rule (1, $\&$, $\otimes$, $\exists$ or entropy) or a left binary rule ($\multimap$ or a cut), not a right rule because of the "**;**" to the left. One verifies then easily that in all cases the premisse(s) has (have) the required form.

The important point is that the $\diamond$ to the left does not disappear when one travels up into the proof, because the constraints d in the formulas $d \multimap D$ which are translations of agents contain no $\diamond$. ■

The following lemma is immediate :

Lemma 22 *If c and d are of synchronization constraints and $c \vdash_{\mathcal{C}} d$, then $c = d$. If c is a synchronization constraint, d a well-formed constraint, and $c \vdash_{\mathcal{C}} d$, then $c \dashv\vdash d$.*

We shall avoid the bureaucratic distinctions between two logically equivalent formulas, thus if $c \dashv\vdash d$ we shall consider that $c = d$.

Theorem 23 *Let $\kappa = (\boldsymbol{x}; c; \Gamma)$ be a well-formed LCC configuration, and ϕ be a procedure name or a suspension.*
If $\kappa^* \vdash_{NLI(\mathcal{C},\mathcal{D})} \vec{\diamond} \otimes \phi^*$, then $\kappa \overset{\triangleright}{\longrightarrow} \phi$.

Proof. In a proof in the sequent calculus $NLI(\mathcal{C}, \mathcal{D})$ of a sequent whose formulas are built with 1, $\otimes$, $\multimap$, $\exists$ and $\&$, one can assume that the cuts bear only on non-logical axioms coming from $\mathcal{C}$ and $\mathcal{D}$ (cut-elimination of NLI), so every formula in the proof contains the above mentioned connectives and quantifier.

Now in such a proof, the left members of sequents contain separators of two kinds : "," and ";". Assume that a ";" has been introduced, necessarily by the $\multimapdot\ \vdash$ rule

$$\frac{\Gamma \vdash c \qquad \Delta[A] \vdash B}{\Delta[\Gamma ; c \multimapdot A] \vdash B}$$

Travel down along this proof until this occurrence of ";" is eliminated. This has to happen with the entropy rule or $\vdash\ \multimapdot$.

The important point is that in case it is eliminated by $\vdash\ \multimapdot$, the part of the proof which is between the introduction and the elimination is very simple, because of the hypothesis on the nature of ϕ (suspensions bearing on synchronization constraints). Indeed when traveling down along the proof, the formulas on the left of the ";" considered are in particular built from formulas of Γ. The hypothesis on the nature of ϕ forces thus Γ to be either $\emptyset$ or just a synchronization constraint (atomic formula).

In the first case one would then have $\Delta[\,] = d;[\,];\Delta'$ for some synchronization constraint d, but as $A = \diamond \otimes A'$ one would then have a proof of $d;\diamond \otimes A';\Delta' \vdash B$, what is impossible according to lemma 21 as B is of the form $\diamond \otimes B'$.

Therefore $\Delta[\,] = [\,]$ and $\Gamma = d$, with d a synchronization constraint. As $d \vdash c$ and c is a well-formed constraint (by hypothesis), lemma 22 says that $d \dashv\vdash c$, so $d = c$ up to logical equivalence. Moreover, as $\Delta[\Gamma;c \multimapdot A] = c;c \multimapdot A$, the only possible next rule in the proof is $\vdash\ \multimapdot$. In the case of a ";" eliminated by $\vdash\ \multimapdot$, the proof has therefore the following necessary form :

$$\frac{\dfrac{c \vdash c \qquad A \vdash B}{c;c \multimapdot A \vdash B}}{c \multimapdot A \vdash c \multimapdot B} \qquad \text{what we shall abbreviate by} \qquad \frac{A \vdash B}{c \multimapdot A \vdash c \multimapdot B}$$

considering more simply that no sequentiality (";") has been actually introduced in this part of the proof.

Hence (with the above convention) every ";" introduced is eliminated by the entropy rule.

Now that the useful ";" have been isolated, one can concentrate on the main part of the proof : let us show that if $\Gamma \vdash_{NLI(\mathcal{C},\mathcal{D})} \overset{\rightarrow}{\diamond} \otimes \phi^*$, where Γ is an arbitrary context whose formulas are translations of agents $A_1, \ldots, A_n$, then $(\emptyset; 1; A_1, \ldots, A_n) \xrightarrow{\triangleright} \phi$. One proceeds by induction on a proof of $\Gamma \vdash \phi$ in the sequent calculus $NLI(\mathcal{C},\mathcal{D})$, with the associativity of "," and ";" and the commutativity of "," implicit, and with the above convention. This induction has a meaning because one can assume that cuts bear only on non-logical axioms (theorem 15), so that when traveling up through such a proof, the formula on the right of the sequent remains a constraint and the formulas on the left remain translations of agents.

Now each logical rule simulates an LCC transition rule.

– π is an axiom : one uses the reflexivity of $\longrightarrow$ in the case of a logical axiom, the rule *declarations* for an axiom $p \vdash q$; the case of an axiom $d \vdash_{\mathcal{C}} e$ is trivial.

– π ends with a cut. Consider for instance :

$$\frac{p \vdash \psi^* \qquad \Gamma^*[\psi^*] \vdash \phi^*}{\Gamma^*[p] \vdash \phi^*}$$

By induction hypothesis, $(\emptyset; 1; \Gamma, \psi^*) \overset{\triangleright}{\longrightarrow} \phi$, and $(p = \psi) \in \mathcal{P}$, thus by applying the rule *declaration*, one obtains $(\emptyset; 1; \Gamma, p) \overset{\triangleright}{\longrightarrow} \phi$, qed.

The other cases are similar.

– π ends with a left introduction of 1 : immediate by the rule *inaction*.

– π ends with a left or right introduction of $\otimes$: immediate.

– π ends with :

$$\frac{\Gamma^*[A^*] \vdash \phi^*}{\Gamma^*[A^* \& B^*] \vdash \phi^*}$$

By induction hypothesis, $(\emptyset; 1; A, \Gamma) \overset{\triangleright}{\longrightarrow} \phi$. Now $(\emptyset; 1; A + B, \Gamma) \longrightarrow (\emptyset; 1; A, \Gamma)$, thus $(\emptyset; 1; A + B, \Gamma) \overset{\triangleright}{\longrightarrow} \phi$.

– π ends with a right (case when ϕ is a constraint, $n = 0$) introduction of $\exists$: immediate.

– π ends with :

$$\frac{\Gamma^\dagger[A^\dagger] \vdash \phi}{\Gamma^\dagger[\exists x A^\dagger] \vdash \phi}\ x \notin fv(\Gamma, \phi)$$

By induction hypothesis, $(\emptyset; 1; A, \Gamma) \overset{\triangleright}{\longrightarrow} \phi$. As $x \notin fv(\Gamma)$, $(\emptyset; 1; \exists x A, \Gamma) \equiv (x; 1; A, \Gamma)$, and moreover $x \notin fv(\phi)$, so by lemma 20, $(\emptyset; 1; \exists x A, \Gamma) \overset{\triangleright}{\longrightarrow} \phi$, qed.

– π ends with :

$$\frac{\Delta^* \vdash c \qquad \Gamma^*[A^*] \vdash \diamond \otimes \phi^*}{\Gamma^*[\Delta^*; c \multimap A^*] \vdash \diamond \otimes \phi^*}$$

By induction hypothesis, $(\emptyset; 1; \Delta) \overset{\triangleright}{\longrightarrow} c$, i.e. there exists a configuration $(\boldsymbol{y}; d; 1)$, such that $d \vdash_{\mathcal{C}} c$ and $(\emptyset; 1; \Delta) \longrightarrow (\boldsymbol{y}; d; 1)$. Thus by applying the rule *ask*, one obtains $(\emptyset; 1; c \to A, \Delta) \longrightarrow (\boldsymbol{y}; d; c \to A) \longrightarrow (\boldsymbol{y}; 1; A)$. Therefore, $(\emptyset; 1; c \to A, \Delta, \Gamma) \longrightarrow (\boldsymbol{y}; 1; A, \Gamma)$. Moreover by induction hypothesis, $(\emptyset; 1; A, \Gamma) \overset{\triangleright}{\longrightarrow} \phi$, whence $(\emptyset; 1; c \to A, \Delta, \Gamma) \overset{\triangleright}{\longrightarrow} \phi$.

– π ends with :

$$\frac{A^* \vdash B^*}{c \multimap A^* \vdash c \multimap B^*}$$

(With our convention, this rule replaces the right introduction of $\multimap$.) It is clear, since $(\emptyset; 1; c \to A) \overset{\triangleright}{\longrightarrow} c \to B$.

– π ends with an entropy : immediate, we have already done the job of collecting the interesting "**;**" 's (i.e. those which are eliminated by the rule $\vdash \multimap$).

– π ends with a dereliction, a weakening or a contraction : immediate.

– π ends with a promotion : in this case all the formulas are necessarily constraints, therefore it is immediate. ■

Corollary 24 (Observation of suspensions) *Let A be an LCC agent and $\phi = c \otimes (d_1 \multimap\!\!\bullet A_1) \otimes \cdots \otimes (d_n \multimap\!\!\bullet A_n)$ be a suspension formula.*

If $A^* \vdash_{NLI(\mathcal{C},\mathcal{D})} \vec{\diamond} \otimes \phi$, then A suspends with a store $d > c$ on the constraints $d_1, ..., d_n$.

Proof. Evident, by definition of a suspension (11), applying the previous theorem to the configuration $(\emptyset; 1; A)$. ■

Corollary 25 (Observation of successes) *Let A be an LCC agent and c be a linear constraint. If $A^* \vdash_{NLI(\mathcal{C},\mathcal{D})} \vec{\diamond} \otimes c$, then c is a success for A, i.e. there exists a constraint d such that $d \vdash_{\mathcal{C}} c$ and $(\emptyset; 1; A) \longrightarrow (\emptyset; d; 1)$.*

Proof. It is a particular case of corollary 24, with ϕ a constraint ($n = 0$). ■

Recall that $\top$ is the additive true constant [9], which is neutral for &.

Corollary 26 (Observation of stores) *Let A be an LCC agent and c be a linear constraint. If $A^* \vdash_{NLI(\mathcal{C},\mathcal{D})} c \otimes \top$, then c is a store accessible from A, i.e. there exist a constraint d such that $d > c$ and a multiset Γ of agents such that $(\emptyset; 1; A) \longrightarrow (\emptyset; d; \Gamma)$.*

Proof. One shows by easy induction on a proof of $\Gamma \vdash_{NLI(\mathcal{C},\mathcal{D})} c \otimes \top$, where Γ is an arbitrary context whose formulas are translations of agents $A_1, \dots, A_n$, that c is accessible from $(\emptyset; 1; A_1, \dots, A_n)$. The proof uses corollary 25, for the right introduction of the tensor connective in $c \otimes \top$. ■

7 Conclusion and perspectives

The intuition behind CC computations has served to define a new non-commutative linear logic which combines both commutative and non-commutative connectives, and is further studied in [28]. That logic extends the intuitionistic version of de Groote [5] (and thus Lambek's calculus [14]) and is an intuitionistic fragment of the proposal of the first author in [27]. It differs from other proposals made by Retoré [25] to combine both kinds of connectives.

In this paper, we have considered monotonic and non-monotonic non-deterministic CC programs (with blind choice), and introduced the class of synchronization constraint systems. We have shown that non-commutative logic enables the characterization of stores, successes, and suspensions with synchronization constraints.

Besides the better understanding of the logical foundations of concurrent constraint programming, these results are directly relevant to CC program analysis methods. In [7] we study a (commutative) linear logic semantics of CC programs for the observation of both stores and successes, and show show how the phase semantics of linear logic can be used to give simple "semantical" proofs of safety properties of CC computations. The development of the phase semantics of NLI [28] should provide in turn more powerful tools for the static analysis of concurrent programs, and a new denotational semantics of CC programs capturing finer observable properties of CC computations than the ones currently available [31, 12, 29]. We believe that the phase semantics of NLI should be an interesting research direction from both viewpoints of proof theory and concurrency.

Besides the logical characterization of finer operational aspects of unrestricted CC computations (e.g. one-step committed choice, sequential composition, which have not been considered in this paper), the present work offers also new perspectives to the connection between concurrency and proof theory, in the paradigm of logic programming. Other examples of such correspondences have been proposed for Petri nets and a propositional fragment of linear logic [17], for LO [1] and the π-calculus [19] at the expense of extra-logical operators. It suggests also an investigation of the usual semantics of concurrency, specifically that of (bi)simulations in the light of proof theory as in [13, 18]. Another perspective is the study the execution of (L)CC agents in the syntax of proof nets (which is intrinsically parallel, thus naturally more suited for the representation of concurrent computations than are sequent calculi), as in [23].

References

1. J.M. Andreoli and R. Pareschi. Linear objects: logical processes with built-in inheritance. *New Generation Computing*, 9, 1991.
2. G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96, 1992.
3. E. Best, F.S. de Boer, and C. Palamidessi. Concurrent constraint programming with information removal. In *Proc. of Coordination'97*. Springer LNCS, 1997.
4. G. Boudol. Asynchrony and the π-calculus. Technical Report RR 1702, INRIA, 1992.
5. Ph. de Groote. Partially commutative linear logic: sequent calculus and phase semantics. In *Proofs and Linguistic Categories, Proceedings 1996 Roma Workshop*. In V. M. Abrusci and C. Casadio, eds. Cooperativa Libraria Universitaria Editrice Bologna, 1996.
6. F. Fages. Constructive negation by pruning. *J. of Logic Programming*, 32(2), 1997.
7. F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: operational and phase semantics. *Technical Report LIENS*, 1997.
8. 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.
9. J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987.
10. J.Y. Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92:69–108, 1989.
11. J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. 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, Department of Computer Science, University of Tokyo, 1993.

14. J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.
15. P. Lincoln and V.A. Saraswat. Proofs as concurrent processes. Parc Xerox Tech. Report, 1991.
16. M.J. Maher. Logic semantics for a class of committed-choice programs. In *Proceedings of ICLP'87, International Conference on Logic Programming*, 1987.
17. 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.
18. N. Mendler, P. Panangaden, P.J. Scott, and R.A.G. Seely. A logical view of concurrent constraint programming. *Nordic J. of Computing*, 2, 1995.
19. 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.
20. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1), 1992.
21. R. Mohring. Computationally tractable classes of ordered sets. *NATO ASI Series (I.Rival, ed)*, 255, 1989.
22. C. Palamidessi. Constraint programming for concurrent and distributed computing. In *Proc. of the JFPLC-UNIF '97, Invited talk*, pages 11–20, 1997.
23. G. Perrier. Concurrent programming as proof net construction. Technical Report CRIN 96-R-132, INRIA-Lorraine, 1996.
24. A. Podelski and G. Smolka. Operational semantics of constraint logic programming with coroutining. In *Proceedings of ICLP'95, International Conference on Logic Programming*, Tokyo, 1995.
25. Ch. Retoré. Pomset logic - A non-commutative extension of commutative linear logic. In *TLCA'97, Springer LNCS 1210*, 1997.
26. P. Ruet. Logical semantics of concurrent constraint programming. In *Proceedings of CP'96, 2nd 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*, Bulletin of Symbolic Logic 3-2:274–275, Jan. 1997.
28. P. Ruet. *Non-commutative logic and concurrent constraint programming.* PhD thesis, Université Denis Diderot, Paris 7, 1997.
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. Parc Xerox Technical Report, 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. D.N. Yetter. Quantales and (non-commutative) linear logic. *J. of Symbolic Logic*, 55(1), 1990.