Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 312 (2015) 179–195

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Proving Concurrent Constraint Programming Correct, Revisited

Carlos Olarte[1]

ECT, Universidade Federal do Rio Grande do Norte. Natal, Brazil.
DECC, Pontificia Universidad Javeriana Cali. Colombia.

Elaine Pimentel [2]

DMAT, Universidade Federal do Rio Grande do Norte. Natal, Brazil.

Abstract

Concurrent Constraint Programming (CCP) is a simple and powerful model of concurrency where processes interact by *telling* and *asking* constraints into a global store of partial information. Since its inception, CCP has been endowed with declarative semantics where processes are interpreted as formulas in a given logic. This allows for the use of logical machinery to reason about the behavior of programs and to prove properties in a declarative way. Nevertheless, the logical characterization of CCP programs exhibits normally a weak level of adequacy since proofs in the logical system may not correspond directly to traces of the program. In this paper, relying on a focusing discipline, we show that it is possible to give a logical characterization to different CCP-based languages with the highest level of adequacy. We shall also provide a neater way of interpreting procedure calls by adding fixed points to the logical structure.

Keywords: Linear Logic, Concurrent Constraint Programming, Proof Systems, Focusing, Fixed Points.

1 Introduction

Reasoning about concurrent programs is much harder than reasoning about sequential ones. Programmers often find themselves overwhelmed by the many subtle cases of thread interactions they must be aware of to decide whether a concurrent program is correct or not, and by the need of finding the right level of thread atomicity of concurrent programs, avoiding race conditions, coping with mutual exclusion requirements, and guaranteeing deadlock freeness to ensure program reliability.

Concurrent Constraint Programming (CCP) [22,21] is a simple and powerful model of concurrency where agents interact by telling constraints (i.e., formulas in

[1] Email: carlos.olarte@gmail.com

[2] Email: elaine.pimentel@gmail.com

http://dx.doi.org/10.1016/j.entcs.2015.04.011

logic) into a shared store of partial information and synchronize by asking if a given information can be deduced from the store.

The logical characterization of constraint systems in CCP has allowed the development of more powerful systems by simply replacing the underlying logical framework. For instance, in linear CCP (`lcc` [5]), constraints are formulas in Girard's intuitionistic linear logic (ILL) [8] and ask agents are allowed to consume tokens of information from the store. Moreover, in [16,17] it is shown that considering ILL with subexponentials [3] as constraint systems allows for the specification of concurrent systems where epistemic, spatial and temporal modalities are involved.

The logical foundations of CCP makes it an ideal language for the specification of concurrent systems: complex synchronization patterns can be expressed declaratively by means of constraint entailment. Moreover, the dual view of processes as computing agents and as formulas in logic allows for the use of techniques from both process calculi and logic to reason about the behavior of processes.

The connection between logic and CCP processes (and constraint systems) has been studied since its inception: in [22] a closure operator semantics is given to deterministic CCP programs that was later related to the logic of constraints in [19]. In [4] a calculus for proving properties of CCP programs is defined where properties are expressed in an enriched logic of the constraint system. The works in [20,5] relate operational steps of CCP and `lcc` with derivations in ILL. We can also mention the works in [13,4] that give logical semantics to timed CCP languages and provide calculi to verify temporal properties of programs. The reader may find a survey of all these developments in [18].

The relation of CCP programs and derivations in logic studied so far, unfortunately, exhibits a weak level of adequacy: proofs in the logical systems may not correspond to an operational derivation. This paper contributes to close this gap by showing that it is possible to exhibit a stronger level of adequacy for different flavors of CCP calculi. In order to do that, we interpret `lcc` agents as ILL formulas using a focusing discipline [1]. By using focusing, we can classify actions in `lcc` (and then in CCP) as *positive* or *negative*, depending on the polarity of the outermost connective obtained in their translation as formulas in ILL. The positive actions need to interact with the environment, either for choosing a path to follow, or for waiting for a guard to be available. Negative actions do not need any interaction with the context, and can be executed anytime and concurrently, not altering the final result of the computation. We prove that the translation from `lcc` to focused intuitionistic linear logic (ILLF) is adequate, in the sense that a focused phase in ILLF corresponds *exactly* to an operational step in `lcc`, and vice-versa. The results in this paper not only extend the ones in [5] (since we present a stronger adequacy result), but also the ones in [10], with a better understanding of operational derivability.

The idea of using focusing for ensuring a higher level of adequacy is not at all new. In fact, [14] shows how to use focusing, fixed points and delays in order to specify sequential programs: this can be achieved only by using subexponentials. In this work, we deal with concurrent instead of sequential programs and, differently from [14] and our previous work in [16], we do not make use of subexponentials. We interpret `lcc` processes using pure linear logic. Hence, the encoding is more natural and direct, and we can use all the rich and already stablished meta-theory developed for linear logic to help in drawing conclusions about CCP systems. Moreover, we study different notions of observables not considered in [16] (see Definition 4.6). Particularly, we show that there are `lcc` computations that cannot be mimicked by the standard encoding of processes as ILLF formulas. Then, by introducing delays in the encoding, we recover the one-to-one correspondence between ILLF derivations and `lcc` computations. We also study the behavior of non-deterministic processes with blind and guarded choices not present in [16].

Finally, we give to procedure calls a more modern presentation using fixed points. Although this idea is already present in [20], here we exploit better the use of intuitionistic linear logic with fixed points (μILL) for adding the greatest fixed point operator to ILL. On doing that, we open the possibility of using co-induction to study in CCP more interesting properties of concurrent system such as bisimilarity, liveness properties, or to reason about non-terminating computations.

The rest of this paper is organized as follows. We start in Section 2 by presenting the base CCP language, determinate-CCP with tell, ask, parallel, locality and recursion operators. We then introduce in Section 3 a focused system for intuitionistic linear logic (ILLF). We show in Section 4 that ILLF allows us to capture precisely the behavior of CCP ask processes: they can be triggered if and only if its guard can be inferred only by the present store and the non-logical axioms. Next, we introduce the indeterminate-CCP language with two kinds of choice operators: blind choice (aka internal choice) and one-step guarded choice. We show that a simple adjust in our encoding suffices to capture such behaviors keeping the level of adequacy. We also show how to interpret procedure calls using fixed points. Section 5 concludes the paper.

2 CCP calculi

Concurrent Constraint Programming (CCP) [22,21] (see a survey at [18]) is a model of concurrency that combines the traditional operational view of process calculi with a declarative view based on logic. This allows CCP to benefit from the large set of reasoning techniques of both process calculi and logic.

Processes in CCP interact with each other by telling and asking constraints (pieces of information) in a common store of partial information. The type of constraints processes may act on is not fixed but parametric in a constraint system. Such systems can be formalized as a Scott information system [23] as in [22], or they can be built upon a suitable fragment of logic [24,5,13].

The store in CCP grows monotonically, this means that agents are only allowed to add new information but it is not possible to delete constraints from the store. In order to have a better resource control, the Linear CCP (`lcc`) language was proposed in [5], where constraints are seen as formulas in a fragment of intuitionistic linear logic (ILL) [8]. More precisely, the linear constraint system is redefined as follows.

Definition 2.1 [Linear Constraint Systems [5]] A linear constraint system is a pair

$(\mathcal{C}, \vdash_\Delta)$ where $\mathcal{C}$ is a set of formulas (linear constraints) built from a first-order signature and the grammar

$$G := 1 \mid A \mid !\,G \mid G \otimes G \mid \exists \overline{x}.G$$

where A is an atomic formula, $\otimes$ is the multiplicative conjunction, 1 is its neutral element, $\exists$ is the existential quantifier and ! is the bang exponential of ILL. We shall use c, c', d, d', etc, to denote elements of $\mathcal{C}$. Moreover, let Δ be a set of non-logical axioms of the form $\forall \overline{x}[c \multimap c']$ where $\multimap$ is the linear implication and all free variables in c and c' are in $\overline{x}$. We say that d *entails* d', written as $d \vdash_\Delta d'$, iff the sequent $!\,\Delta, d \longrightarrow d'$ is provable in ILL.

As it was shown in [5], it is trivial to recover the monotonic behavior of CCP from `lcc` by requiring that every constraint in the constraint system is marked with "!", i.e., all constraints are seen as unbounded resources. For this reason, we shall keep our discussion on `lcc` having in mind that it extends straightforwardly to CCP.

The syntax of `lcc` processes (without the choice operator for the moment) is given next.

Definition 2.2 [Syntax of `lcc`] Processes are built from constraints in the underlying constraint system as follows:

$$P, Q ::= \textbf{tell}(c) \mid \textbf{ask}\ c\ \textbf{then}\ P \mid P \parallel Q \mid (\textbf{local}\, x)\, P \mid p(\overline{x})$$

The process $\textbf{tell}(c)$ adds c to the current store d producing the new store $d \otimes c$. The process $\textbf{ask}\ c\ \textbf{then}\ P$ evolves into P if the current store entails c. In this case, c is consumed from the store. In other case, the process remains blocked until enough information is added to the store. This provides a powerful synchronization mechanism based on constraint entailment. Following [5], the free variables occurring in c, denoted as $fv(c)$, are assumed to be universally quantified. To keep simpler the syntax, we shall omit the "$\forall x$" in $\forall x.(\textbf{ask}\ c\ \textbf{then}\ P)$ if $x \in fv(c)$.

The process $P \parallel Q$ represents the parallel (interleaved) execution of P and Q. The process $(\textbf{local}\, x)\, P$ behaves as P and binds the variable x to be local to it.

Given a process definition $p(\overline{y}) \stackrel{\Delta}{=} P$ where all free variables of P are in the set of pairwise distinct variables $\overline{y}$, the process $p(\overline{x})$ evolves into $P[\overline{x}/\overline{y}]$.

`lcc` Programs are of the form $\mathcal{D}.P$ where $\mathcal{D}$ is a set of process definitions and P a process. It is assumed that every process name $p(\cdot)$ has a unique definition in $\mathcal{D}$.

The structural operational semantics (SOS) of `lcc` is given by the transition relation $\gamma \longrightarrow \gamma'$ satisfying the rules in Figure 1. These rules are straightforward realizing the operational intuitions given above. Here we follow the semantics presented in [5] where the local variables created by the program appear explicitly in the transition system. More precisely, a *configuration* γ is a triple of the form $(X; \Gamma; c)$, where c is a constraint (a logical formula specifying the store), Γ is a multiset of processes, and X is a set of hidden (local) variables of c and Γ. The multiset $\Gamma = P_1, P_2, \ldots, P_n$ represents the process $P_1 \parallel P_2 \ldots \parallel P_n$. We shall indistinguishably use both notations to denote parallel composition of processes.

$$\frac{(X;\Gamma;c) \cong (X';\Gamma';c') \longrightarrow (Y';\Delta';d') \cong (Y;\Delta;d)}{(X;\Gamma;c) \to (Y;\Delta;d)}\ \mathrm{R}_{\mathrm{EQUIV}}$$

$$\frac{}{(X;\mathbf{tell}(c),\Gamma;d) \longrightarrow (X;\Gamma;c\otimes d)}\ \mathrm{R}_{\mathrm{T}} \qquad \frac{d \vdash_\Delta c[\overline{y}/\overline{x}]\otimes e, \qquad \overline{x} = fv(c)}{(X;\mathbf{ask}\ c\ \mathbf{then}\ P,\Gamma;d) \longrightarrow (X;P,\Gamma;e)}\ \mathrm{R}_{\mathrm{A}}$$

$$\frac{x \notin X \cup fv(d) \cup fv(\Gamma)}{(X;(\mathbf{local}\,x)\,P,\Gamma;d) \longrightarrow (X\cup\{x\};P,\Gamma;d)}\ \mathrm{R}_{\mathrm{L}} \qquad \frac{p(\overline{x}) \overset{\Delta}{=} P}{(X;p(\overline{y}),\Gamma;d) \longrightarrow (X;P[\overline{y}/\overline{x}],\Gamma;d)}\ \mathrm{R}_{\mathrm{C}}$$

Fig. 1. Operational semantics of `lcc`. In rule R_{A}, e is the most general constraint to avoid weakening the store (see [9])

Processes are quotiented by a structural congruence relation $\cong$ satisfying:

(i) $(\mathbf{local}\,\overline{x})\,P \cong (\mathbf{local}\,\overline{y})\,P[\overline{y}/\overline{x}]$ if $\overline{y}\cap fv(P)=\emptyset$; – alpha conversion

(ii) $P \parallel Q \cong Q \parallel P$;

(iii) $P \parallel (Q \parallel R) \cong (P \parallel Q) \parallel R$.

Furthermore, $\Gamma = \{P_1,...,P_n\} \cong \{P'_1,...,P'_n\} = \Gamma'$ iff $P_i \cong P'_i$ for all $1 \leq i \leq n$. Finally, $(X;\Gamma;c) \cong (X';\Gamma';c')$ iff $X = X'$, $\Gamma \cong \Gamma'$ and $c \equiv_\Delta c'$ (*i.e.*, $c \vdash_\Delta c'$ and $c' \vdash_\Delta c$).

We conclude here with the notion of observables that will play a central role in the adequacy theorems in Section 4.

Definition 2.3 [Observables] Let $\longrightarrow^*$ be the reflexive and transitive closure of $\longrightarrow$. If $(X;\Gamma;d) \longrightarrow^* (X';\Gamma';d')$ and $\exists X'.d' \vdash_\Delta c$ we write $(X;\Gamma;d) \Downarrow_c$. If $X=\emptyset$ and $d=1$ we simply write $\Gamma \Downarrow_c$.

Intuitively, if P is a process then $P \Downarrow_c$ says that P outputs c under input 1.

3 ILLF: a focused system for intuitionistic linear logic

Focusing is a discipline on proofs that was first proposed for Linear Logic in [1], in the context of logic programming, aiming at reducing the non-determinism during proof search. Focused proofs can be interpreted as the normal form proofs for proof search.

The focused intuitionistic linear logic system (ILLF) is depicted in Figure 2. This system is based on the system LJF proposed in [11].

ILL connectives are separated into two classes, the *negative* ones: $\multimap, \&, \top, \forall$ and the *positive* ones: $\otimes, \oplus, \exists, !, 1$. The polarity of non-atomic formulas is inherited from its outermost connective and *positive* bias is assigned to atomic formulas.[3] ILLF contains four types of sequents:

[3] Observe that the system ILLF, as presented in Figure 2 induces a *positive* polarity to atoms. Although the bias assigned to atoms does not interfere with provability [12], it changes *considerably* the shape of proofs. In the present work, it is extremely important, for the sake of guaranteeing the high level of adequacy, that atoms have a positive behavior.

Negative Phase

$$\frac{}{[\Upsilon : \Gamma], \Theta \longrightarrow \top}\ \top_R \qquad \frac{[\Upsilon : \Gamma], \Theta \longrightarrow \mathcal{R}}{[\Upsilon : \Gamma], \Theta, 1 \longrightarrow \mathcal{R}}\ 1_L \qquad \frac{[\Upsilon : \Gamma], \Theta, F \longrightarrow G}{[\Upsilon : \Gamma], \Theta \longrightarrow F \multimap G}\ \multimap_R$$

$$\frac{[\Upsilon : \Gamma], \Theta, F, G \longrightarrow \mathcal{R}}{[\Upsilon : \Gamma], \Theta, F \otimes G \longrightarrow \mathcal{R}}\ \otimes_L \qquad \frac{[\Upsilon : \Gamma], \Theta \longrightarrow F \quad [\Upsilon : \Gamma], \Theta \longrightarrow G}{[\Upsilon : \Gamma], \Theta \longrightarrow F \,\&\, G}\ \&_R$$

$$\frac{[\Upsilon : \Gamma], \Theta \longrightarrow G[y/x]}{[\Upsilon : \Gamma], \Theta \longrightarrow \forall x.G}\ \forall_R \qquad \frac{[\Upsilon : \Gamma], \Theta, G[c/x] \longrightarrow \mathcal{R}}{[\Upsilon : \Gamma], \Theta, \exists x.G \longrightarrow \mathcal{R}}\ \exists_L$$

$$\frac{[\Upsilon : \Gamma], \Theta, F \longrightarrow \mathcal{R} \quad [\Upsilon : \Gamma], \Theta, H \longrightarrow \mathcal{R}}{[\Upsilon : \Gamma], \Theta, F \oplus H \longrightarrow \mathcal{R}}\ \oplus_L \qquad \frac{[\Upsilon, F : \Gamma], \Theta \longrightarrow \mathcal{R}}{[\Upsilon : \Gamma], \Theta, !F \longrightarrow \mathcal{R}}\ !_L$$

Positive Phase

$$\frac{[\Upsilon : \Gamma_1] -_{H}\!\rightarrow \quad [\Upsilon : \Gamma_2] -_{G}\!\rightarrow}{[\Upsilon : \Gamma_1, \Gamma_2] -_{H \otimes G}\!\rightarrow}\ \otimes_R \qquad \frac{[\Upsilon : \Gamma_1] -_{F}\!\rightarrow \quad [\Upsilon : \Gamma_2] \xrightarrow{H} [G]}{[\Upsilon : \Gamma_1, \Gamma_2] \xrightarrow{F \multimap H} [G]}\ \multimap_L$$

$$\frac{[\Upsilon : \Gamma] -_{G[t/x]}\!\rightarrow}{[\Upsilon : \Gamma] -_{\exists x.G}\!\rightarrow}\ \exists_R \qquad \frac{[\Upsilon : \Gamma] \xrightarrow{F[t/x]} [G]}{[\Upsilon : \Gamma] \xrightarrow{\forall x.F} [G]}\ \forall_L$$

$$\frac{[\Upsilon : \Gamma] \xrightarrow{F_i} [G]}{[\Upsilon : \Gamma] \xrightarrow{F_1 \& F_2} [G]}\ \&_{L_i} \qquad \frac{[\Upsilon : \Gamma] -_{G_i}\!\rightarrow}{[\Upsilon : \Gamma] -_{G_1 \oplus G_2}\!\rightarrow}\ \oplus_{R_i} \qquad \frac{[\Upsilon : \cdot] \longrightarrow G}{[\Upsilon : \cdot] -_{!G}\!\rightarrow}\ !_R$$

$$\frac{}{[\Upsilon : \cdot] -_{1}\!\rightarrow}\ 1_R \qquad \frac{}{[\Upsilon : \Gamma] -_{A}\!\rightarrow}\ I_R \text{ given } A \in (\Gamma \cup \Upsilon) \text{ and } (\Gamma \subseteq \{A\})$$

Structural Rules

$$\frac{[\Upsilon : \Gamma, N_a], \Theta \longrightarrow \mathcal{R}}{[\Upsilon : \Gamma], \Theta, N_a \longrightarrow \mathcal{R}}\ []_L \qquad \frac{[\Upsilon : \Gamma], \Theta \longrightarrow [P]}{[\Upsilon : \Gamma], \Theta \longrightarrow P}\ []_R$$

$$\frac{[\Upsilon, F : \Gamma] \xrightarrow{F} [G]}{[\Upsilon, F : \Gamma] \longrightarrow [G]}\ D_L \qquad \frac{[\Upsilon : \Gamma] \xrightarrow{N} [G]}{[\Upsilon : N, \Gamma] \longrightarrow [G]}\ D_L \qquad \frac{[\Upsilon : \Gamma] -_{G}\!\rightarrow}{[\Upsilon : \Gamma] \longrightarrow [G]}\ D_R$$

$$\frac{[\Upsilon : \Gamma], P_a \longrightarrow [F]}{[\Upsilon : \Gamma] \xrightarrow{P_a} [F]}\ R_L \qquad \frac{[\Upsilon : \Gamma] \longrightarrow N}{[\Upsilon : \Gamma] -_{N}\!\rightarrow}\ R_R$$

Fig. 2. Focused Proof System for ILLF. $\mathcal{R}$ stands for either a bracketed formula, $[F]$, or an unbracketed formula. A is an atomic formula; P is a positive formula; P_a is a positive or atomic formula; N is a negative formula; and N_a is a negative or atomic formula. Variable y in $[\forall]$ rule does not occur elsewhere.

i. $[\Upsilon : \Gamma], \Theta \longrightarrow \mathcal{R}$ is an unfocused sequent, where $\mathcal{R}$ is either a bracketed formula $[F]$ or an unbracketed one. Here Γ contains only atomic or negative formulas, while Υ is the classical context.

ii. $[\Upsilon : \Gamma] \longrightarrow [F]$ is a sequent representing the end of a negative phase.

$$
\begin{array}{llll}
\mathcal{L}[\![\mathbf{tell}(c)]\!] & = c & \mathcal{L}[\![P \parallel Q]\!] & = \mathcal{L}[\![P]\!] \otimes \mathcal{L}[\![Q]\!] \\
\mathcal{L}[\![\mathbf{ask}\ c\ \mathbf{then}\ P]\!] & = \forall \overline{x}(c \multimap \mathcal{L}[\![P]\!]) & \mathcal{L}[\![\exists x(P)]\!] & = \exists x.(\mathcal{L}[\![P]\!]) \\
\mathcal{L}[\![p(\overline{x}) \stackrel{\Delta}{=} P]\!] & = \forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!] & \mathcal{L}[\![p(\overline{y})]\!] & = p(\overline{y})
\end{array}
$$

Fig. 3. Interpretation of `lcc` processes and processes definitions as ILL formulas.

iii. $[\Upsilon : \Gamma] -_F\!\!\rightarrow$ is a sequent focused on the right.

iv. $[\Upsilon : \Gamma] \xrightarrow{F} G$ is a sequent focused on the left.

Observe, in Figure 2, that the negative connectives have invertible *right* rules, while the positive connectives have invertible *left* rules. This separation induces a two phase proof construction: a *negative*, where no *backtracking* on the selection of inference rules is necessary, and a *positive*, where choices within inference rules can lead to failures for which one may need to backtrack.

In the negative phase, sequents have the shape (i) above and all the negative non-atomic formulas on the right and all the positive non-atomic formulas on the left are introduced. Also, atomic and negative formulas on the left and positive formulas on the right are stored in the respective contexts using the bracket rules $[]_L$ and $[]_R$. When this phase ends, sequents have the form (ii).

The positive phase begins by choosing via one of the decide rules ($[D_L]$ or $[D_R]$) a formula on which to focus, enabling sequents of the forms (iii) or (iv). Rules are then applied on the focused formula until either an axiom is reached (in which case the proof ends), the right promotion rule $!_R$ is applied (and focusing will be lost) or a negative subformula on the right or a positive subformula on the left is derived (and the proof switches to the negative phase again).

This means that focused proofs can be seen (bottom-up) as a sequence of alternations between negative and positive phases.

4 From `lcc` processes to ILLF formulas

In CCP-based calculi, processes are not only agents that evolve according to the rules of the underlying operational semantics: they also can be seen as formulas in intuitionistic linear logic. The logical interpretation of `lcc` is defined with the aid of a function $\mathcal{L}[\![\cdot]\!]$ defined in Figure 3 [5]. We will call p the *head* of the process definition $\forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]$ while c will be the *guard* of the ask formula $\forall \overline{x}(c \multimap \mathcal{L}[\![P]\!])$.

The following result states that the interpretation $\mathcal{L}[\![\cdot]\!]$ is *faithful*.

Theorem 4.1 (Adequacy – ILL [5]) *Let P be a process, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, for any constraint c, $P \Downarrow_c$ iff there is a proof of the sequent $!\,\mathcal{L}[\![\Psi]\!], !\,\Delta, \mathcal{L}[\![P]\!] \longrightarrow c \otimes \top$ in ILL.* [4]

The adequacy of this interpretation is *on the level of proofs* [15], meaning that

[4] The top ($\top$) erases the formulas corresponding to blocked processes. We will denote by $!\,\mathcal{L}[\![\Psi]\!]$ the mapping of $!\,\mathcal{L}[\![\cdot]\!]$ to elements in Ψ.

there may be logical steps not corresponding to any operational step and vice versa. For instance, consider the case where the last rule applied on a proof of an ILL sequent is the implication left

$$\dfrac{\overset{\pi_1}{\Gamma_1, b \longrightarrow c} \quad \overset{\pi_2}{\Gamma_2 \longrightarrow a}}{\Gamma_1, \Gamma_2, a \multimap b \longrightarrow c} \multimap_L$$

In this case, π_2 could contain sub-derivations that have nothing to do with the proof of a. For instance, processes definitions could be unfolded or other processes could be executed. This would correspond, operationally, to the act of triggering an ask **ask** a **then** P with no guarantee that its guard a will be derivable only from the set of non-logical axioms Δ and the store d: it may be the case that a will be later produced by a process P' such that $\mathcal{L}[\![P']\!] \in \Gamma_2$, for example. Observe that this *is not* allowed by the operational semantics of `lcc` (see rule $\mathrm{R_A}$ in Figure 1).

A simple inspection on the interpretation in Figure 3 shows that the fragment of ILL needed for encoding `lcc` processes and processes definitions is given by the following grammar for guards/goals G, processes P and processes definitions PD.

$$\begin{array}{ll} G & := 1 \mid A \mid\, !\,G \mid G \otimes G \mid \exists x.G \\ P & := G \mid P \otimes P \mid P \,\&\, P \mid \forall \overline{x}.G \multimap P \mid \exists x.P \mid p(\overline{t}) \\ PD & := \forall \overline{x}.p(\overline{x}) \multimap P. \end{array}$$

where A is an atomic formula in $\mathcal{C}$ (see Definition 2.1) while p is also atomic but $p \notin \mathcal{C}$. The process corresponding to the formula $P \,\&\, P$ is the non-deterministic choice that we later introduce in Section 4.3.

Note that, due to this syntax, we will use only a small fragment of ILLF. In fact, we will only use the negative rules $1_L, \otimes_L, \exists_L, !_L$ and the positive rules $\otimes_R, \multimap_L$ $, \exists_R, !_R, \&_L$. The formulas also have special forms and behavior, as described bellow.

- *Formulas on the right (guards/goals G, heads p).* The correspondent logical fragment of `lcc` in ILLF have *strictly positive* formulas on the right. There are three cases to consider.
  - The formula on the right is the head p of a process definition. Thus it has to be focused (since it will come from a focused implication formula on the left), it is positive and atomic. Hence the proof must end immediately with the initial axiom I_R.
  - Goals have no occurrences of !: hence, focusing *cannot* be lost on the right, and a focused formula will be decomposed *entirely* and *at once* into its atomic subformulas. At this point, since the only possible action on focused atomic formulas on the right is to apply the axiom I_R, the derivation must end.
  - Focusing on a goal of the form $!\,G$ is only possible if the linear context is empty (see rule $!_R$ in Figure 2). In this case, focus will be lost but only classical constraints, non-logical axioms and procedure calls can be in the context. See Lemma 4.2 for a detailed behavior in this case.

- *Formulas on the left (programs P).* On the other hand, it is possible to have positive or negative formulas on the left.
  - *Positive formulas* on the left (that cannot be focused) come from the interpretation of one of the actions: *tell*, *parallel composition* and *locality* that do not need any interaction with the context. We call these actions *negative*. As an example, the parallel composition $P \parallel Q$ is translated in ILLF as $\mathcal{L}[\![Q]\!] \otimes \mathcal{L}[\![P]\!]$. Notice that $\otimes$ is a positive connective, decomposing it on the left side of a sequent will be done in a negative phase.
  - *Negative formulas* on the left (that can be chosen for focusing) come from one of the actions: *ask*, *non-deterministic choice* (to be presented later – see Definition 4.11) and *procedure calls*. They *do need* to interact with the environment, either for choosing a path to follow (in non-deterministic choices), or for waiting for a guard to be available (in asks or procedure calls). We call these actions *positive*. Consider, for example, an *ask* process $\forall \overline{x}(c \multimap \mathcal{L}[\![P]\!])$. Note that both $\forall$ and $\multimap$ are negative, hence decomposing them on the left side of a sequent will be done in a positive phase.

The next lemma clarifies better the above cases when formulas on the right are focused. In particular, we show how is the shape of the derivations in a proof involving banged guards and goals: such formulas are derivable by other guards and non-logical axioms only. Actually, we may state a stronger result: there is *no proof* of banged guards and goals if a process definition is chosen to be focused on.

Lemma 4.2 *Let G be a guard, $\mathcal{G}$ be a set of guards, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, the sequent $[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] -_{!G}\!\!\rightarrow$ is provable if and only if $[\mathcal{G}, \Delta : \cdot] -_{!G}\!\!\rightarrow$ is provable. Moreover, if $\forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!] \in \mathcal{L}[\![\Psi]\!]$ then the sequent*

$$[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] \xrightarrow{\forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]} [G]$$

is not provable.

Proof. Note that all sequents in a derivation of $[\mathcal{L}[\![\Psi]\!], \Delta : \cdot], \mathcal{L}[\![P]\!] \longrightarrow c$ will have the shape $[\mathcal{L}[\![\Psi]\!], \Delta, \mathcal{G} : \Gamma], \Theta \longrightarrow G'$, where $\mathcal{G}$ is a set of classical constraints, Γ has only atomic or negative formulas and Θ has only encoded processes. Note also that proving $[\mathcal{L}[\![\Psi]\!], \Delta, \mathcal{G} : \Gamma], \Theta \longrightarrow !G$ by focusing on the banged formula on the right is possible only if $\Gamma = \Theta = \emptyset$. Finally, observe that a proof of $[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] -_{!G}\!\!\rightarrow$ has necessarily the shape:

$$\dfrac{\overset{\pi}{[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] \longrightarrow G}}{[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] -_{!G}\!\!\rightarrow}\ !_R$$

Now, π either continues by focusing on G or on some formula in $\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta$. Assume that $\forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!] \in \Psi$ is chosen for focusing. Since $\forall$ and $\multimap$ are negative, focus will not be lost and π must have the shape

$$\cfrac{\cfrac{[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] \xrightarrow{\mathcal{L}[\![P]\!]} [G] \quad [\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] -_{p(t)}\!\!\rightarrow}{[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] \xrightarrow{\forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]} [G]} \ \forall_L, \multimap_L}{[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] \longrightarrow [G]} \ []_R, D_L$$

But the sequent $[\mathcal{L}[\![\Psi]\!], \mathcal{G}, \Delta : \cdot] -_{p(t)}\!\!\rightarrow$ is not provable since p is atomic, positive and not a constraint, hence focus cannot be lost and the proof cannot finish with the initial axiom since p is not in $\mathcal{L}[\![\Psi]\!] \cup \mathcal{G} \cup \Delta$. □

We are now able to show how focusing in ILL allows us to prove that every logical step in ILLF corresponds exactly to an operational step in `lcc`.

Theorem 4.3 (Adequacy – ILLF) *Let P be a process, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, for any constraint c,*

$$P \Downarrow_c \text{ iff there is a proof of the sequent } [\mathcal{L}[\![\Psi]\!], \Delta : \cdot], \mathcal{L}[\![P]\!] \longrightarrow c \otimes \top$$

in ILLF. Moreover, the adequacy from `lcc` to ILLF is at the level of proofs*, while from ILLF to `lcc` the adequacy is at the level of* derivations*, that is, one focused logical phase corresponds exactly to one operational step.*

Proof. Since the focused system ILLF is complete w.r.t. ILL, the first part of the proof is an immediate corollary of Theorem 4.1. The proof of the adequacy level from ILLF to `lcc` is by straightforward case analysis. We will illustrate it by showing the case for the ask process. Suppose that $P = \textbf{ask } a \textbf{ then } P'$. Hence focusing on $\mathcal{L}[\![P]\!]$ would produce the derivation

$$\cfrac{\cfrac{\overset{\pi_1}{[\Upsilon : \Gamma_1] \xrightarrow{\mathcal{L}[\![P']\!]} [d]} \quad \overset{\pi_2}{[\Upsilon : \Gamma_2] -_a\!\!\rightarrow}}{[\Upsilon : \Gamma_1, \Gamma_2] \xrightarrow{\forall \overline{x}.a \multimap \mathcal{L}[\![P']\!]} [d]} \ \forall_L, \multimap_L}{[\Upsilon : \forall \overline{x}.a \multimap \mathcal{L}[\![P']\!], \Gamma_1, \Gamma_2] \longrightarrow [d]} \ D_L$$

Observe that a is guard. If it does not have banged subformulas, it cannot be unfocused on the right, and it will be decomposed entirely into its atomic subformulas, and the subproofs should finish with the initial axiom on those formulas. This means that we can only focus on $\mathcal{L}[\![P]\!]$ if all atomic subformulas needed to prove a are already in the context. On the other hand, if a have banged subformulas, Lemma 4.2 states that a will be proved by using constraints and non-logical axioms only, matching *exactly* the semantics given by rule $\mathrm{R_A}$: the guard a can be inferred by the context, possibly using non-logical axioms in Δ. □

We note that the result above is not at the *full* level of derivations, though, since there are operational steps that do not have any logical correspondent as shown in the next section.

4.1 Maximal derivations and interleaving in ask agents

In this section we study the cases when the operational steps cannot be followed by the derivations in the logical system. Then, we show how to obtain the highest level of adequacy in such cases. Let us start by presenting some examples illustrating these situations.

Example 4.4 [Traces, proofs and focusing] Consider the `lcc` process

$$P = \mathbf{tell}(a\otimes b) \parallel \mathbf{ask}\ a\ \mathbf{then}\ \mathbf{ask}\ b\ \mathbf{then}\ \mathbf{tell}(a\otimes b) \parallel \mathbf{ask}\ b\ \mathbf{then}\ \mathbf{ask}\ a\ \mathbf{then}\ \mathbf{tell}(\mathtt{ok}).$$

We denote the second and the third ask agents in P as A_1 and A_2 respectively. The SOS of `lcc` dictates that there are two possible transitions leading to the store `ok`. Both of such transitions start with the negative action $\mathbf{tell}(a \otimes b)$:

$$\texttt{Derivation 1: } \langle\emptyset; P; 1\rangle \longrightarrow \langle\emptyset; A_1 \parallel A_2; a \otimes b\rangle \longrightarrow \langle\emptyset; \mathbf{ask}\ b\ \mathbf{then}\ \mathbf{tell}(a \otimes b) \parallel A_2; b\rangle$$
$$\longrightarrow \langle\emptyset; \mathbf{tell}(a \otimes b) \parallel A_2; 1\rangle \longrightarrow \langle\emptyset; A_2; a \otimes b\rangle \longrightarrow^* \langle\emptyset; \cdot; \mathtt{ok}\rangle \not\longrightarrow$$
$$\texttt{Derivation 2: } \langle\emptyset; P; 1\rangle \longrightarrow \langle\emptyset; A_1 \parallel A_2; a \otimes b\rangle \longrightarrow \langle\emptyset; A_1 \parallel \mathbf{ask}\ a\ \mathbf{then}\ \mathbf{tell}(\mathtt{ok}); a\rangle$$
$$\longrightarrow \langle\emptyset; A_1 \parallel \mathbf{tell}(\mathtt{ok}); 1\rangle \longrightarrow \langle\emptyset; A_1; \mathtt{ok}\rangle \not\longrightarrow$$

These transitions correspond exactly to a different focused proof of the sequent $\mathcal{L}[\![P]\!] \longrightarrow \mathtt{ok}$: one focusing first on $\mathcal{L}[\![A_1]\!]$ and the other focusing first on $\mathcal{L}[\![A_2]\!]$.

On the other hand, there is also an *interleaved* execution of A_1 and A_2 that does not lead to the final store `ok`:

$$\texttt{Derivation 3: } \langle\emptyset; P; 1\rangle \longrightarrow \langle\emptyset; A_1 \parallel A_2; a \otimes b\rangle \longrightarrow \langle\emptyset; \mathbf{ask}\ b\ \mathbf{then}\ \mathbf{tell}(a \otimes b) \parallel A_2; b\rangle$$
$$\longrightarrow \langle\emptyset; \mathbf{ask}\ b\ \mathbf{then}\ \mathbf{tell}(a \otimes b) \parallel \mathbf{ask}\ a\ \mathbf{then}\ \mathbf{tell}(\mathtt{ok}); 1\rangle \not\longrightarrow$$

The above trace does not have any correspondent derivation in the ILLF system. In fact, since $\multimap$ is a negative connective, focusing on $\mathcal{L}[\![A_1]\!]$ will decompose the formula $a \multimap b \multimap (a \otimes b)$, consuming a and producing the focused formula $b \multimap (a \otimes b)$, which is still negative. Hence focusing cannot be lost and the inner ask has to be triggered.

Remark 4.5 This last example shows something really interesting: although the formulas $A \otimes B \multimap C$ and $A \multimap B \multimap C$ are logically equivalent, they are operationally different when concurrent computations are considered.

Observe that the last derivation in Example 4.4 does not produce any observable store (see Definition 2.3). Hence a good question is whether it is possible to restrict the behavior of ask agents to avoid interleaved executions when we are interested in observing a given constraint, i.e., when the system exhibits an output. Fortunately, the answer is positive as shown below.

As we know, once the guard c in $\mathbf{ask}\ c\ \mathbf{then}\ P$ is entailed, the operational semantics dictates that P is enabled for execution (rule $\mathrm{R_A}$). The semantics, however, does not enforce the *immediate* execution of P. Next definition gives an alternative semantics to ask agents in order to force the execution of P avoiding *interleaved* derivations as the ones in Example 4.4.

Definition 4.6 [Standard and Maximal Derivations and Observables] We say that a derivation in `lcc` using the relation $\longrightarrow$ as in Figure 1 is *standard*. We define the *maximal*-ask transition relation $\rightsquigarrow$ on configuration similar to $\longrightarrow$ but replacing the rule $\mathrm{R_A}$ with the rule $\mathrm{R_{AM}}$ below:

$$\frac{d \vdash_{\Delta} c_1[\overline{y}_1/\overline{x}_1] \otimes ... \otimes c_n[\overline{y}_n/\overline{x}_n] \otimes e, \qquad \overline{x}_i = fv(c_i) \qquad \star}{(X; P, \Gamma; d) \rightsquigarrow (X; Q, \Gamma; e)}\ \mathrm{R_{AM}}$$

where the side condition $\star$ means that P is a process of the shape:

$$\textbf{ask } c_1 \textbf{ then ask } c_2 \textbf{ then } ... \textbf{ask } c_n \textbf{ then } Q \qquad (1)$$

and Q is not an ask agent. We define the observables of a process under the $\rightsquigarrow$ relation, notation $(X; \Gamma; d) \Downarrow\!\!\Downarrow c$, similarly as in Definition 2.3.

Intuitively, in a standard derivation of the shape $\gamma \longrightarrow \gamma'$, interleaved executions of ask agents are allowed as in `Derivation 3` above. On the contrary, in a maximal derivation $\gamma \rightsquigarrow \gamma'$, an ask as the one in Equation 1 has to wait until all the guards $c_1, ..., c_n$ can be entailed from the store and then executes Q in *one* step as in `Derivation 1` and `Derivation 2` above.

The next theorem shows that if there is an output using the semantics $\longrightarrow$, then such output can be also computed by using the semantics $\rightsquigarrow$.

Theorem 4.7 *Let P be a process. Then, for any constraint c,*

$$(X; P; e) \Downarrow_c \ \textit{if and only if}\ \ (X; P; e) \Downarrow\!\!\Downarrow c$$

Proof. The ($\Leftarrow$) part of the proof is immediate since $\rightsquigarrow$ is a particular scheduling for a $\longrightarrow$ derivation. As for the ($\Rightarrow$) part we proceed as follows. If the current store is able to entail d, then a process of the shape

$$P = \textbf{ask } d \textbf{ then } (\textbf{ask } d' \textbf{ then } P')$$

evolves into **ask** d' **then** P' not producing any constraint. Then either P will remain blocked and hence it will not be used for producing c, or d' will be produced by some other process R and P will reduce to P'. But in this last case, R does not depend on P, and it can be executed before. Hence the nested asks can be executed at once. □

We can now state a stronger adequacy result, restricted to maximal derivations in `lcc`.

Theorem 4.8 (Strong adequacy – maximal derivations) *Let P be a process, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, for any constraint c,*

$$P \Downarrow\!\!\Downarrow c \textit{ iff there is a proof of the sequent } [\mathcal{L}[\![\Psi]\!], \Delta : \cdot], \mathcal{L}[\![P]\!] \longrightarrow c \otimes \top$$

in ILLF. Moreover, the adequacy is at the level of derivations, *that is, one focused logical phase corresponds exactly to one operational step and vice-versa.*

Proof. Observe that, in the process **ask** c **then** P, the only possibility of $\mathcal{L}[\![P]\!]$ being a negative formula is if P is also an ask. Hence maximal derivations in `lcc` correspond exactly to focused derivations in ILLF. □

4.2 Interleaving and delays

Interleaving can be handled in a focused system by the use of *logical delays*. In fact, since the formula $c \multimap \mathcal{L}[\![P]\!]$ is negative, focusing on it on the left side of the sequent will produce a focused $\mathcal{L}[\![P]\!]$. If this formula is also negative, focusing will not be lost and this operationally means that P should be executed right away. However, for any formula A, A is logically equivalent to $A \otimes 1$, which is a positive formula. Hence, we can easily force the focusing phase to end by substituting $\mathcal{L}[\![P]\!]$ by $\mathcal{L}[\![P]\!] \otimes 1$. More precisely, we define the encoding $\mathcal{L}[\![\cdot]\!]_1$ as $\mathcal{L}[\![\cdot]\!]$ but replacing the cases for the ask agents and process definitions as follows:

$$\begin{aligned}
\mathcal{L}[\![\mathbf{ask}\ c\ \mathbf{then}\ P]\!]_1 &= \forall \overline{x}.c \multimap (\mathcal{L}[\![P]\!]_1 \otimes 1)\\
\mathcal{L}[\![p(\overline{x}) \stackrel{\Delta}{=} P]\!]_1 &= \forall \overline{x}.p(\overline{x}) \multimap (\mathcal{L}[\![P]\!]_1 \otimes 1)
\end{aligned}$$

In this case, we can have a stronger adequacy theorem for the whole `lcc` system.

Theorem 4.9 (Strong adequacy – standard derivations) *Let P be a process, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, for any constraint c,*

$$P \Downarrow_c \ \textit{iff there is a proof of the sequent } [\mathcal{L}[\![\Psi]\!]_1, \Delta : \cdot], \mathcal{L}[\![P]\!]_1 \longrightarrow c \otimes \top$$

in ILLF. The adequacy is at the level of derivations.

Remark 4.10 Observe that Theorem 4.8 gives a canonical trace to `lcc` successful computations via focusing. In this case, the guards of nested ask agents are evaluated at once to decide whether the process continues blocked or not. On the other hand, Theorem 4.9 shows that traces of a derivation in logic have a one-to-one correspondence with traces of a computation in a `lcc` program.

4.3 Indeterminate CCP languages

Non-determinism is introduced in CCP by means of the choice operator. Let us then extend the syntax and the operational semantics given in Section 2.[5]

Definition 4.11 [Indeterminate CCP] Indeterminate CCP processes are obtained from the syntax in Definition 2.2 extended with the constructor

$$P, Q ::= \sum_I P_i$$

[5] Note that `lcc`, without choices, is also non-deterministic since ask agents may *compete* for the same token.

$$\frac{}{\langle \overline{x}; \sum_I P_i, \Gamma; c\rangle \longrightarrow \langle \overline{x}; P_i, \Gamma; c\rangle}\ \mathrm{R_{BCH}}$$

(a) Blind Choice

$$\frac{\langle \overline{x}; P_i, \Gamma; c\rangle \longrightarrow \langle \overline{x}'; \Gamma'; c'\rangle}{\langle \overline{x}; \sum_I P_i, \Gamma; c\rangle \longrightarrow \langle \overline{x}'; \Gamma'; c'\rangle}\ \mathrm{R_{GCH}}$$

(b) One-step guarded choice

Fig. 4. Operational Rules for the Non-Deterministic Choice

Given a finite set of indexes I, the process $\sum_I P_i$ chooses one o the P_j for execution. The choice of one of the processes precludes the others from execution. When $|I| = 2$, we shall write $P_1 + P_2$ instead of $\sum_I P_i$.

We can give at least two possible interpretation of the non-deterministic choice as shown in Figure 4. The Rule $\mathrm{R_{BCH}}$ corresponds to the blind (or internal) choice (BC). This rules says that a process P_i is chosen for execution regardless whether the process P_i can evolve or not. In the Rule $\mathrm{R_{GCH}}$, the chosen process P_i must not block, i.e., it must exhibit at least one transition. Since the only blocking agent in Syntax 2.2 is the ask agent, this kind of non-deterministic choice is written as a summation of ask agents as in $\sum_i \textbf{ask}\ c_i\ \textbf{then}\ P_i$. Thus, rule for one-step guarded choice (GC) can be read as P_i is chosen for execution whenever its guard c_i can be entailed from the current store. We note that a BC can be seen also as a GC where all guard c_i is the constraint 1.

The logical clauses representing blind and guarded choice are, respectively:

$$\mathcal{L}[\![\sum_I P_i]\!]_{BC} = \&_I(\mathcal{L}[\![P_i]\!] \otimes 1) \qquad \mathcal{L}[\![\sum_I P_i]\!]_{GC} = \&_I \mathcal{L}[\![P_i]\!]$$

Observe that, this way, we capture well the behavior of choosing one process from the choices we have. At the same time, forcing formulas to be positive in the BC case implies that the chosen process will not block on the positive phase. On the other hand, P_i being a negative formula in the case GC, assures that the choice will be triggered *only if* the guard is already in the context. Hence, we continue having a neat logical control corresponding to the operational semantics and Theorem 4.9 is also valid for indeterminate CCP.

4.4 Procedure calls as fixed points

We conclude this section by explaining how procedure calls can be seen as (greatest) fixed points. It turns out that we can give meaning to procedure calls as formulas in μILL (ILL with fixed points [6]). Assuming that $p(\overline{y}) \stackrel{\Delta}{=} P$, the encoding $\mathcal{L}[\![\cdot]\!]_\mu$ is the same as $\mathcal{L}[\![\cdot]\!]_1$ (see Section 4.2) but

$$\mathcal{L}[\![p(\overline{t})]\!]_\mu = \nu(\mathcal{L}[\![P]\!]_\mu[\overline{t}/\overline{y}] \otimes 1)$$

[6] The system presented here is an adaptation, for the intuitionistic case, of the system μMALL presented in [2]. We note that, although μMALL do not have exponentials, it is possible to encode exponentials using fixed points. In fact, $[!\,P]$ can be encoded as $\nu(\lambda p.1 \,\&\, (p \otimes p) \,\&\, [P])$.

where ν is the greatest fixed point operator with rules

$$\frac{[\Upsilon : \Gamma] \xrightarrow{B(\nu B\bar{t})} [G]}{[\Upsilon : \Gamma] \xrightarrow{\nu B\bar{t}} [G]}\ \nu_L \qquad \frac{[\Upsilon : \Gamma], \Theta \longrightarrow S\bar{t} \quad [\Upsilon : \Gamma], \Theta, S\overline{x} \longrightarrow B(S\overline{x})}{[\Upsilon : \Gamma], \Theta \longrightarrow \nu B\bar{t}}\ \nu_R$$

In the above rules, $\overline{x}$ are fresh variables, $\bar{t}$ are terms, S is a closed formula of the same type as B (called the co-invariant), and B has the form $\lambda p\lambda\overline{x}.Pp\overline{x}$ with p a predicate constant. Observe that ν is defined here as a negative operator and that B can be unfolded indefinitely, since it is applied to νB in the rule ν_L. For having completeness of the focusing system, it would be necessary to add *frozen formulas* and restrict the system to *quasi-infinite* proofs [2], but all this machinery will not be needed here, since we use only a small fragment of μMALL.

First of all, we do not use co-induction, since fixed points do not appear in the right side of sequents. In fact, our base grammar for guards/goals G and processes P now is the following

$$G := 1 \mid A \mid !G \mid G \otimes G \mid \exists x.G$$

$$P := G \mid P \otimes P \mid P \,\&\, P \mid \forall\overline{x}.G \multimap P \mid \exists x.P \mid \nu P[\bar{t}/\overline{x}]$$

Secondly, higher-order procedure calls are not present in our encoding. Hence the application of the rule ν_L matches *exactly* the behavior of focusing on $\forall\overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]_1 \otimes 1$, where $p(\overline{x})$ is substituted by $P[\bar{t}/\overline{x}]$:

$$\frac{[\Upsilon : \Gamma] \xrightarrow{\mathcal{L}[\![P]\!]_1[\bar{t}/\overline{x}]\otimes 1} [G] \quad \dfrac{}{[\Upsilon : p(\bar{t})] -_{p(\bar{t})}\!\!\rightarrow}\ I_R}{[\Upsilon : \Gamma, p(\bar{t})] \xrightarrow{\forall\overline{x}.p(\overline{x})\multimap(\mathcal{L}[\![P]\!]_1\otimes 1)} [G]}\ \forall_L, \multimap_L \qquad \rightsquigarrow \qquad \frac{\dfrac{[\Upsilon : \Gamma] \xrightarrow{\mathcal{L}[\![P]\!]_\mu[\bar{t}/\overline{x}]\otimes 1} [G]}{[\Upsilon : \Gamma] \xrightarrow{\nu(\mathcal{L}[\![P]\!]_\mu[\bar{t}/\overline{x}]\otimes 1)} [G]}\ \nu_L}{[\Upsilon : \Gamma, \nu(\mathcal{L}[\![P]\!]_\mu[\bar{t}/\overline{x}] \otimes 1)] \longrightarrow [G]}\ D_L$$

Observe that focus will be lost due to the delay.

Finally, we observe that we could have followed [14] and simply *unfold* fixed points. Then, in order to verify properties of the systems, one can adopt a two level approach as in [7] and use a meta-level tool, like Abella [6], in order to prove properties by co-induction.

Theorem 4.12 (Adequacy – fixed points) *Let P be a process and Δ be a set of non-logical axioms. Then, for any constraint c,*

$$P \Downarrow_c \textit{ iff } [\Delta : \cdot], \mathcal{L}[\![P]\!]_\mu \longrightarrow c \otimes \top \textit{ is provable in } \mu ILL.$$

The discussion of the levels of adequacy is the same as done in the precedent sections.

5 Related work and concluding remarks

We presented a new translation of CCP systems into linear logic. We showed that, by using a focusing discipline, one is able to have a complete control of concurrent processes via logic, closing for good the connection between proof theory and constraint systems.

The chosen translation is simpler than the one presented in [16]. In fact, here we do not make use of subexponentials [3], and we show that focusing is responsible, alone, for the strongest possible level of adequacy. However, it is not possible to deal with different CCP-flavors (e.g., epistemic, spatial and temporal extensions of CCP) using only focusing: for that the subexponentials are needed. Also, differently from [16], here we have explored different aspects of computation. In particular, we dealt with non-determinism and we showed how to control the traces due to the interleaving of processes. We also studied in a deeper detail how the synchronization of agents can be better controlled, as well as how to deal with procedure calls via fixed points.

It is worthy noticing that in [14] the authors used subexponentials for reasoning about sequential programs, while in [16] we used subexpontentials for handling modalities in CCP systems. Here we do not use subexponentials, keeping the translation simpler and more natural.

We plan to use the logical semantics presented here to derive optimization procedures for CCP interpreters. In particular, our characterization of positive and negative actions in `lcc` may allow us to "*sequentialize*" part of the code. This is useful to reduce the number of suspended threads in an execution of a CCP-program. We also plan to extend the encodings presented here for *timed* extensions of CCP [13]. Finally we want to make the full use of fixed points, adding verification of properties that can be proved using co-induction. Since μMALL is implemented in the system Bedwyr (`http://slimmer.gforge.inria.fr/bedwyr/`), we can use all its machinery in order to specify and verify properties of `lcc` systems.

Another research direction would be to consider higher-order processes as those in [20]. This can be handled by fixpoints characterization of procedure calls as done in Section 4.4. For completeness of the resulting focusing system, however, we need to restrict the system to *quasi-finite derivations*: derivations having a finite height and using a finite number of different co-invariants, as done in [2]. This shall allow us to deal with properties like bisimulation and liveness in `lcc`.

References

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[2] David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Log.*, 13(1):2, 2012.

[3] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In *Kurt Gödel Colloquium*, pages 159–171, 1993.

[4] Frank S. de Boer, Maurizio Gabbrielli, Elena Marchiori, and Catuscia Palamidessi. Proving concurrent constraint programs correct. *ACM Trans. Program. Lang. Syst.*, 19(5):685–725, 1997.

[5] François Fages, Paul Ruet, and Sylvain Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Inf. Comput.*, 165(1):14–41, 2001.

[6] Andrew Gacek. The Abella interactive theorem prover (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of IJCAR 2008*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 154–161. Springer, August 2008.

[7] Andrew Gacek, Dale Miller, and Gopalan Nadathur. A two-level logic approach to reasoning about computations. *J. Autom. Reasoning*, 49(2):241–273, 2012.

[8] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[9] Rémy Haemmerlé. Observational equivalences for linear logic concurrent constraint languages. *TPLP*, 11(4-5):469–485, 2011.

[10] Radha Jagadeesan, Gopalan Nadathur, and Vijay A. Saraswat. Testing concurrent systems: An interpretation of intuitionistic logic. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 517–528. Springer, 2005.

[11] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theor. Comput. Sci.*, 410(46):4747–4768, 2009.

[12] Dale Miller and Alexis Saurin. From proofs to focused proofs: A modular proof of focalization in linear logic. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 405–419. Springer, 2007.

[13] M. Nielsen, C. Palamidessi, and F. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Computing*, 9(1):145–188, 2002.

[14] Vivek Nigam and Dale Miller. Algorithmic specifications in linear logic with subexponentials. In António Porto and Francisco Javier López-Fraguas, editors, *PPDP*, pages 129–140. ACM, 2009.

[15] Vivek Nigam and Dale Miller. A framework for proof systems. *J. of Automated Reasoning*, 45(2):157–188, 2010.

[16] Vivek Nigam, Carlos Olarte, and Elaine Pimentel. A general proof system for modalities in concurrent constraint programming. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR*, volume 8052 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.

[17] Carlos Olarte, Vivek Nigam, and Elaine Pimentel. Dynamic spaces in concurrent constraint programming. *Electr. Notes Theor. Comput. Sci.*, 305:103–121, 2014.

[18] Carlos Olarte, Camilo Rueda, and Frank D. Valencia. Models and emerging trends of concurrent constraint programming. *Constraints*, 18(4):535–578, 2013.

[19] Prakash Panangaden, Vijay A. Saraswat, Philip J. Scott, and R. A. G. Seely. A hyperdoctrinal view of concurrent constraint programming. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 666 of *Lecture Notes in Computer Science*, pages 457–476. Springer, 1992.

[20] Vijay Saraswat and Patrick Lincoln. Higher-order Linear Concurrent Constraint Programming. Technical report, 1992.

[21] Vijay A. Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.

[22] Vijay A. Saraswat, Martin C. Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In David S. Wise, editor, *POPL*, pages 333–352. ACM Press, 1991.

[23] Dana S. Scott. Domains for denotational semantics. In Mogens Nielsen and Erik Meineche Schmidt, editors, *ICALP*, volume 140 of *LNCS*, pages 577–613. Springer, 1982.

[24] Gert Smolka. A foundation for higher-order concurrent constraint programming. In J.-P. Jouannaud, editor, *Proceedings of Constraints in Computational Logics*, volume 845 of *LNCS*, pages 50–72. Springer-Verlag, 1994.