Theoretical Computer Science 685 (2017) 46–64

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On concurrent behaviors and focusing in linear logic

Carlos Olarte [a,*], Elaine Pimentel [b,*]

[a] Escola de Ciências e Tecnologia, Universidade Federal do Rio Grande do Norte, Natal, Brazil
[b] Departamento de Matemática, Universidade Federal do Rio Grande do Norte, Natal, Brazil

ARTICLE INFO

Article history:
Received 12 July 2015
Received in revised form 17 August 2016
Accepted 23 August 2016
Available online 20 September 2016

Keywords:
Linear logic
Concurrent constraint programming
Proof systems
Focusing
Multi-focusing
Fixed points

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 of them. 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, we study different encodings from CCP into intuitionistic linear logic (ILL) and we compare the level of adequacy attained in each. By relying on a focusing discipline, we show that it is possible to give a logical characterization to CCP with the highest level of adequacy. Moreover, we show how to characterize maximal-parallelism semantics for CCP by relying on a multi-focusing discipline for ILL. These results, besides giving proof techniques for CCP, entail (safe) optimizations for the execution of CCP programs. Finally, we show how to interpret CCP procedure calls as fixed points in ILL, thus opening the possibility of reasoning by induction about properties of CCP programs.

1. Introduction

Concurrent Constraint Programming (CCP) [44,45] is a simple and powerful model of concurrency based upon the shared-variables communication model. In this model, agents interact by telling constraints (i.e., formulas in logic) into a shared store of partial information and synchronize by asking if a given information can be deduced from the store. Hence, processes can be seen as information transducers.

The connection between logic and CCP processes (and constraint systems) has been studied since its inception: in [45] a losure operator semantics is given to deterministic CCP programs that was later related to the logic of constraints in [38]. In [13] 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 [42,15] relate operational steps of CCP and its linear variant `lcc` with derivations in intuitionistic linear logic (ILL) [18]. We can also mention the works in [28,13] 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 [36].

The logical foundations of CCP make 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

* Corresponding authors.
E-mail addresses: carlos.olarte@gmail.com (C. Olarte), elaine.pimentel@gmail.com (E. Pimentel).

http://dx.doi.org/10.1016/j.tcs.2016.08.026
0304-3975/

computing agents and as formulas in logic allows for the use of techniques from both process calculi and logic for reasoning about the behavior of processes. These features have been extensively used for the specification and verification of systems in different applications domains such as biochemical, multimedia interaction, physical/mechanical and mobile systems (see e.g. [36]).

Unfortunately, the relation of CCP programs and derivations in logic studied so far exhibits a weak level of adequacy: proofs in the logical system may not correspond to an operational derivation. Moreover, different notions of observables in CCP cannot be directly (and accurately) traced in the logical system. This paper closes this gap by studying different encodings of CCP into ILL and by stating precisely the level of adequacy attained in each. Moreover, we show how different forms of focusing in ILL give rise to different forms of concurrent behaviors in CCP. We then strive at establishing the foundations to build better proof procedures for the verification of CCP programs and to guide the design of interpreters for CCP languages.

We rely on a focusing discipline for ILL [2] (ILLF) and classify actions 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 (i.e., a constraint) 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 it is possible to give an ILL interpretation to CCP with the highest level of adequacy, where a focused phase in ILLF corresponds *exactly* to an operational step in CCP, and vice versa. The results in this paper not only extend the ones in [15] (since we present a stronger adequacy result), but also the ones in [20], 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, [29] shows how to use focusing, fixed points and delays in order to specify *sequential* programs: this can be achieved only by using subexponentials [12] in linear logic. In this work, we deal with concurrent instead of sequential programs and, differently from [29] and our previous work in [31], we do not make use of subexponentials: we interpret CCP processes using pure linear logic. Hence, the encodings are more natural and direct, and we can use all the rich and already established meta-theory of ILL to help in drawing conclusions about CCP systems. Moreover, we study different notions of observables not considered in [31] (see e.g. Definition 4.1). In particular, we show that there are CCP computations that cannot be mimicked by the standard encoding of processes as ILLF formulas. We recover the one-to-one correspondence between ILLF derivations and CCP computations by introducing logical delays. We also study the behavior of non-deterministic processes with blind and guarded choices not present in [31].

For representing CCP systems with maximal parallelism semantics, we propose mILL, a multi-focusing proof system for ILL. We show that, relying on a multi-focusing discipline, it is possible to accurately represent CCP systems where all the enabled actions must be executed at once in a single step. We show also that we can give an ILL characterization to `tccp` [14], a timed extension of CCP. In this language, the notion of time is identified with the time needed to tell/ask constraints and a maximal parallelism semantics is assumed.

Previous encodings from CCP into ILL [15,31] treat procedure definitions as formulas of the shape $\forall\vec{x}.p(\vec{x}) \multimap B$ where B is the formula representing the body defining the procedure $p(\cdot)$. Such technique allows for mimicking the operational behavior of procedure calls (i.e., unfolding the definition) but it is not strong enough to verify properties of infinite computations. For that, as standardly done, it is needed to include fixpoint operators into the system. For instance, [13] enhances the language of properties with a least fixpoint operator, and Scott induction can be used to derive (semantic) properties of recursive CCP programs. We hence conclude this paper by encoding procedure calls as fixed point formulas, showing the possibility of using induction to prove more interesting properties for CCP programs. Although this idea is already present in [42], here we exploit better the use of intuitionistic linear logic with fixed points (μILL) [3].

Our contributions are: (a) we propose alternative semantics to CCP systems, based *only* on different logical strategies. This allows for code optimization procedures based on strong logical grounds. That is logic guiding computation; (b) we put to work some already developed proof theory for ILL, giving it a strong computational meaning. That is computation giving meaning to logic; and (c) we propose a new logical system (mILL), capable of capturing maximal parallelism and timed executions.

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 present in Section 3 a focused system for intuitionistic linear logic (ILLF). In Section 4 we study different encodings of CCP into ILLF and identify, in each case, the level of adequacy attained w.r.t. to a certain kind of observables (outputs). Next, we introduce the indeterminate-CCP language with two kinds of choice operators: blind choice (a.k.a. internal choice) and one-step guarded choice. We show that a simple adjustment in our encoding suffices to capture such behaviors keeping the highest level of adequacy. The multifocus system mILL is introduced in Section 5 as well the adequacy results for the timed CCP language proposed in [14]. In Section 6, we show how to interpret procedure calls using fixed points. We also present some examples of properties of CCP programs that require the induction principle derived from μILL. Finally, Section 7 concludes the paper.

A preliminary short version of this paper was published in [34]. In this paper we give many more examples and explanations. We also refine several technical details and present full proofs. The new contributions with respect to [34] are: (1) we give a better classification of each encoding w.r.t. the level of adequacy attained; (2) we introduce a multifocus system that is able to give precise meaning to maximal-parallelism semantics in CCP; (3) we use the multifocus system to encode the

timed language in [14]; and, finally, (4) we not only introduce fixpoints to give meaning to procedure calls as in [34] but we also show how this encoding can be used to inductively reason about computations in CCP.

2. CCP calculi

Concurrent Constraint Programming (CCP) [44,45] (see [36] for a survey) 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. Intuitively, a constraint system provides a signature from which constraints can be built from basic tokens (e.g. predicate symbols) and variables, and two basic operations: conjunction ($\wedge$) and variable hiding ($\exists$). The constraint system defines also an *entailment* relation ($\vdash_\Delta$) specifying inter-dependencies between constraints: $c \vdash_\Delta d$ means that the information d can be deduced from the information represented by c. Such systems can be formalized as a Scott information system [46] as in [45], or they can be built upon a suitable fragment of logic e.g. as in [47,15,28]. Here we shall follow the second approach and constraints are seen as formulas in intuitionistic logic [17].

Definition 2.1 *(Constraint system).* A constraint system is a tuple $(\mathcal{C}, \vdash)$ where $\mathcal{C}$ is a set of formulas (constraints) built from a first-order signature and the grammar

$$F := \mathtt{true} \mid A \mid F \wedge F \mid \exists \overline{x}.F$$

where A is an atomic formula. 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 $\forall \overline{x}.[c \supset c']$ where all free variables in c and c' are in $\overline{x}$. We say that d *entails* c, written as $d \vdash c$, iff the sequent $\Delta, d \longrightarrow c$ is probable in LJ [17].

CCP processes. In the spirit of process calculi (see e.g. [6,27]), the language of processes in CCP is given by a small number of primitive operators or combinators. Usually, a CCP language features the following operators: a **tell** operator to add new information (constraints) to the store; an **ask** operator querying if a constraint can be deduced from the store; **parallel composition** combining processes concurrently; a **local** (also known as hiding or restriction) introducing local variables, thus restricting the interface a process can use to interact with others; finally, infinite computations are obtained by means of **recursion**.

Some forms of non-determinism are also allowed in CCP by adding a **choice** operator. For the moment, we introduce the syntax for the deterministic calculus and later we deal with non-deterministic behavior.

Definition 2.2 *(Syntax of deterministic CCP).* Processes are built from constraints in the underlying constraint system as follows:

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

The process $\mathbf{tell}(c)$ adds c to the current store d producing the new store $d \wedge c$. The process $\mathbf{ask}\ c\ \mathbf{then}\ P$ evolves into P if the current store entails c. Otherwise, the process remains blocked until enough information is added to the store. This provides a powerful synchronization mechanism based on entailment of constraints.

The process $P \parallel Q$ represents the parallel (interleaved) execution of P and Q, i.e., P and Q running in parallel and possibly communicating via the shared store.

The process $(\mathbf{local}\, x)\ P$ behaves as P and binds the variable x to be local to it.

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

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

The structural operational semantics (SOS) of CCP is given by the transition relation $\gamma \longrightarrow \gamma'$ satisfying the rules in Fig. 1. Here we follow the semantics presented in [15] 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.

Processes are quotiented by a structural congruence relation $\cong$ satisfying: (1) $P \cong Q$ if they differ only by a renaming of bound variables (alpha-conversion); (2) $P \parallel Q \cong Q \parallel P$; and (3) $P \parallel (Q \parallel R) \cong (P \parallel Q) \parallel R$. Furthermore, $\Gamma = \{P_1, \ldots, P_n\} \cong \{P'_1, \ldots, P'_n\} = \Gamma'$ 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 $c' \vdash_\Delta c$).

Rules R_T and R_C are self-explanatory. Rule R_{EQUIV} says that structurally congruent processes have the same derivations. Rule R_L adds the variable x to the set of variables X when the free-variable condition is satisfied. In other case, Rule R_{EQUIV} can be used to apply alpha conversion. Finally, rule R_A says that the process $Q = \mathbf{ask}\ c\ \mathbf{then}\ P$ evolves into P if the current store d entails c.

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

$$\frac{}{(X;\mathbf{tell}(c),\Gamma;d)\longrightarrow(X;\Gamma;c\wedge d)}\ \mathrm{R}_{\mathrm{T}}\qquad\frac{d\vdash_\Delta c}{\langle X;\mathbf{ask}\ c\ \mathbf{then}\ P,\Gamma;d\rangle\longrightarrow\langle X;P,\Gamma;d\rangle}\ \mathrm{R}_{\mathrm{A}}$$

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

Fig. 1. Operational semantics of CCP. $fv(\Gamma,d)$ means $fv(\Gamma)\cup fv(d)$. $fv(X,\Gamma,d)$ means $fv(\Gamma,d)\cup X$.

Negative Phase

$$\frac{}{\Upsilon:\Gamma;\Theta\Rightarrow\top}\ \top_R\qquad\frac{\Upsilon:\Gamma;\Theta\Rightarrow G}{\Upsilon:\Gamma;\Theta,1\Rightarrow G}\ 1_L\qquad\frac{\Upsilon:\Gamma;\Theta,F\Rightarrow G}{\Upsilon:\Gamma;\Theta\Rightarrow F\multimap G}\ \multimap_R$$

$$\frac{\Upsilon:\Gamma;\Theta,F,H\Rightarrow G}{\Upsilon:\Gamma;\Theta,F\otimes H\Rightarrow G}\ \otimes_L\qquad\frac{\Upsilon:\Gamma;\Theta\Rightarrow F\quad\Upsilon:\Gamma;\Theta\Rightarrow G}{\Upsilon:\Gamma;\Theta\Rightarrow F\,\&\,G}\ \&_R$$

$$\frac{\Upsilon:\Gamma;\Theta\Rightarrow G\{y/x\}}{\Upsilon:\Gamma;\Theta\Rightarrow\forall x.G}\ \forall_R\qquad\frac{\Upsilon:\Gamma;\Theta,F\{y/x\}\Rightarrow G}{\Upsilon:\Gamma;\Theta,\exists x.F\Rightarrow G}\ \exists_L$$

$$\frac{\Upsilon:\Gamma;\Theta,F\Rightarrow G\quad\Upsilon:\Gamma;\Theta,H\Rightarrow G}{\Upsilon:\Gamma;\Theta,F\oplus H\Rightarrow G}\ \oplus_L\qquad\frac{\Upsilon,F:\Gamma;\Theta\Rightarrow G}{\Upsilon:\Gamma;\Theta,!F\Rightarrow G}\ !_L$$

Positive Phase

$$\frac{\Upsilon:\Gamma_1;\cdot\rightarrow[H]\quad\Upsilon:\Gamma_1;\cdot\rightarrow[G]}{\Upsilon:\Gamma_1,\Gamma_2;\cdot\rightarrow[H\otimes G]}\ \otimes_R\qquad\frac{\Upsilon:\Gamma_1;\cdot\rightarrow[F]\quad\Upsilon:\Gamma_1,[H];\cdot\rightarrow G}{\Upsilon:\Gamma_1,\Gamma_2,[F\multimap H];\cdot\rightarrow G}\ \multimap_L$$

$$\frac{\Upsilon:\Gamma;\cdot\rightarrow[G\{t/x\}]}{\Upsilon:\Gamma;\cdot\rightarrow[\exists x.G]}\ \exists_R\qquad\frac{\Upsilon:\Gamma,[F\{t/x\}];\cdot\rightarrow G}{\Upsilon:\Gamma,[\forall x.F];\cdot\rightarrow G}\ \forall_L$$

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

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

Structural Rules

$$\frac{\Upsilon:\Gamma,N_a;\Theta\Rightarrow G}{\Upsilon:\Gamma;\Theta,N_a\Rightarrow G}\ \mathtt{store}\qquad\frac{\Upsilon:\Gamma;\cdot\rightarrow[P]}{\Upsilon:\Gamma;\cdot\Rightarrow P}\ D_R\qquad\frac{\Upsilon,F:\Gamma,[F];\cdot\rightarrow G}{\Upsilon,F:\Gamma;\cdot\Rightarrow G}\ D_{LC}$$

$$\frac{\Upsilon:\Gamma,[N];\cdot\rightarrow G}{\Upsilon:\Gamma,N;\cdot\Rightarrow G}\ D_{LL}\qquad\frac{\Upsilon:\Gamma;P\Rightarrow F}{\Upsilon:\Gamma,[P];\cdot\rightarrow F}\ R_L\qquad\frac{\Upsilon:\Gamma;\cdot\Rightarrow N}{\Upsilon:\Gamma;\cdot\rightarrow[N]}\ R_R$$

Fig. 2. Focused proof system for ILLF. A is an atomic formula; P is a positive formula; N is a negative formula; and N_a is a negative or atomic formula. Variable y in $\forall_R$ and $\exists_L$ rules does not occur elsewhere.

We conclude 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 $d=\mathtt{true}$ we simply write $\Gamma\Downarrow_c$.

Intuitively, if P is a process then $P\Downarrow_c$ says that P outputs c under input $\mathtt{true}$.
The next section introduces the logical framework we shall use for giving a declarative meaning to CCP processes.

3. ILLF: a focused system for intuitionistic linear logic

Focusing [2] is a discipline on proofs aiming at reducing the non-determinism during proof search. Focused proofs can be interpreted as the normal form proofs. The focused intuitionistic linear logic system (ILLF) is depicted in Fig. 2.

ILL connectives are separated into two classes, the *negative*: $\multimap,\&,\top,\forall$ and the *positive*: $\otimes,\oplus,\exists,!,1$. The polarity of non-atomic formulas is inherited from its outermost connective and *positive* bias is assigned to atomic formulas. It is worth noticing that, although the bias assigned to atoms does not interfere with provability [24], it changes *considerably* the shape of proofs (see [40]). In the present work, it is extremely important, for the sake of guaranteeing the high level of adequacy, that atoms have a positive behavior.

Constraints		Processes	
		$\mathcal{L}[\![\mathbf{tell}(c)]\!]$	$= \triangledown c$
$\triangledown \mathtt{true}$	$= 1$	$\mathcal{L}[\![P \parallel Q]\!]$	$= \mathcal{L}[\![P]\!] \otimes \mathcal{L}[\![Q]\!]$
$\triangledown A$	$= !A$	$\mathcal{L}[\![\mathbf{ask}\ c\ \mathbf{then}\ P]\!]$	$= \triangledown c \multimap \mathcal{L}[\![P]\!]$
$\triangledown F_1 \wedge F_2$	$= \triangledown F_1 \otimes \triangledown F_1$	$\mathcal{L}[\![(\mathbf{local}\, x)\ P]\!]$	$= \exists x.\mathcal{L}[\![P]\!]$
$\triangledown \exists x.F$	$= \exists x.\triangledown F$	$\mathcal{L}[\![p(\overline{y})]\!]$	$= p(\overline{y})$
$\triangledown \forall \overline{x}.[c \supset c']$	$= \forall \overline{x}.[\triangledown c \multimap \triangledown c']$	$\mathcal{L}[\![p(\overline{x}) \triangleq P]\!]$	$= \forall \overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]$

Fig. 3. Interpretation of constraints, non-logical axioms (of the constraint system), CCP processes and process definitions as ILL formulas. A is an atomic formula.

Observe, in Fig. 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.

We separate the left context of sequents in ILLF in three: the set Υ will always denote the unbounded context, containing only banged formulas; Γ is a linear context containing only negative or atomic formulas; and Θ is a general linear context. We will differentiate focused and unfocused sequents by using different arrow symbols: "$\Rightarrow$" for unfocused and "$\rightarrow$" for focused. In this way, ILLF contains four types of sequents:

i. $\Upsilon : \Gamma; \Theta \Rightarrow G$ is an unfocused sequent.
ii. $\Upsilon : \Gamma; \cdot \Rightarrow G$ is an unfocused sequent representing the end of a negative phase.
iii. $\Upsilon : \Gamma; \cdot \rightarrow [F]$ is a sequent focused on the right.
iv. $\Upsilon : \Gamma, [F]; \cdot \rightarrow G$ is a sequent focused on the left.

In the negative phase, sequents have the shape (i) above and all the negative formulas on the right and all the positive non-atomic formulas on the left are introduced. Also, atomic and negative formulas on the left are moved to the left linear context Γ using the `store` rule. When this phase ends, sequents have the form (ii).

The positive phase begins by choosing, via one of the decide rules D_{LL}, D_{LC} 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. We will call a *focused phase* a positive phase followed by a negative one.

Rules for intuitionistic linear logic (ILL) are the same as in ILLF, but not considering focusing, and the structural rules being substituted by the usual bang left rules (dereliction, contraction and weakening). Sequents in ILL will be denoted by $\Theta \vdash C$.

4. From CCP processes to ILLF formulas

It is well known [15,31] that processes in CCP can be interpreted as formulas in ILL. It turns out, though, that there are several ways of proposing such an interpretation. And, quite interestingly, these possible different approaches can lead to different logical and computational behaviors, as we shall see in this section.

Any interpretation of a system into another must be *adequate*, in the sense that there must be a 1–1 relation between the sets of interpreted objects with the set of their interpretations. The level of adequacy can then determine how tight are those systems. Following [30], we will classify our interpretations of CCP processes as ILL formulas into two levels of adequacy:

- **FCP** (*full completeness of proofs*) claims that processes outputting an observable are in 1–1 correspondence with the corresponding completed proofs.
- **FCD** (*full completeness of derivations*) claims that one step of computation (in CCP) should correspond to one step of logical reasoning.

While *one step of computation* is rigidly determined by the operation semantics of the CCP system considered, *one step of logical reasoning* depends strongly on the logical framework chosen. In ILL, one step of logical reasoning means one application of a logical rule; in ILLF it means one focused phase; and in mILL (see Section 5) it means one multi-focused phase.

The logical interpretation of CCP is defined with the aid of a function $\mathcal{L}[\![\cdot]\!]$ defined in Fig. 3. Function $\triangledown$ maps constraints (Definition 2.1) into ILL formulas. Recall that the store in CCP is monotonic, i.e., constraints cannot be removed. Hence, atomic constraints (A in Fig. 3) are marked with a bang. In the case of processes, as expected, parallel composition is identified with multiplicative conjunction and ask processes correspond to linear implication. Non-logical axioms of the form $\forall \overline{x}.[c \supset c']$ will be translated as $\forall \overline{x}.[\triangledown c \multimap \triangledown c']$. Moreover, process definitions are (universally quantified) implications to allow the unfolding of its body.

In what follows we call p in $\forall\overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]$ the *head* of the formula while c in $\triangledown c \multimap \mathcal{L}[\![P]\!]$ the *guard* of the formula. Note that those formulas result from the encoding of process definitions and ask processes respectively.

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

Theorem 4.1 *(Adequacy – ILL [15]). 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 $!\mathcal{L}[\![\Psi]\!], !\triangledown\Delta, \mathcal{L}[\![P]\!] \vdash \triangledown c \otimes \top$ in ILL.*[1] *The level of adequacy is **FCP**.*

The low level of adequacy of this interpretation implies that 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 $\multimap_L$

$$\frac{\overset{\pi_1}{\Gamma_1, F \vdash d} \quad \overset{\pi_2}{\Gamma_2 \vdash \triangledown c}}{\Gamma_1, \Gamma_2, \triangledown c \multimap F \vdash d} \multimap_L$$

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

On what follows, we will show, step-by-step, how the encoding (and, later on, the logical system) can be enhanced for having a better level of adequacy w.r.t. concurrent computations. The importance of this discussion is not only proof-theoretical: we will be able to use all linear logic's rich theory in order to propose better computational strategies in CCP languages.

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

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

where A is an atomic formula (constraint) in $\mathcal{C}$ and p is also atomic but $p \notin \mathcal{C}$. The process corresponding to the formula $P \,\&\, P$ is the non-deterministic choice, that will be introduced in Section 4.4.

Note that, due to this syntax, we will only use the negative rules $1_L, \otimes_L, \exists_L, !_L, \top_R$ and the positive rules $\otimes_R, \multimap_L, \exists_R, !_R$, $\&_L, \forall_L$. 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 the encoding 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, i.e., it is positive and atomic.
  - The main connective in the goal is not !. Hence, the connective should be 1, $\otimes$ or $\exists$. In all theses cases focusing *cannot* be lost on the right, and the focused formula will be *entirely* decomposed into formulas of the shape 1 or $!A$.
  - Focusing on a goal of the form $!A$ is only possible if the linear context is empty (see rule $!_R$ in Fig. 2). This means that only the theory $\triangledown\Delta$, the encoding of constraints and procedure calls can be in the context, since they are classical, i.e., formulas proceeded by !. Lemma 4.1 bellow shows that the encoding of procedure calls cannot be used in the proof of A.
- *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 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. As another example, the formula $\exists x.!A_1 \otimes !A_2$, resulting from the encoding of **tell**$(\exists x.A_1 \wedge A_2)$, can be entirely decomposed in a negative phase using the rules $\otimes_L$, $\exists_L$ and $!_L$.
  - *Negative formulas* on the left (that can be chosen for focusing) come from one of the actions: *ask*, *non-deterministic choice* (see Definition 4.3) and *procedure calls*. They *do need* to interact with the environment, either for choosing a path to follow (in non-deterministic choices), or waiting for a guard to be available (in asks or procedure calls). We call these actions *positive*. Consider, for example, the formula $\triangledown c \multimap \mathcal{L}[\![P]\!]$ resulting from the encoding of **ask** c **then** P. Note that $\multimap$ is negative, hence decomposing this formula on the left side of a sequent will be done in a positive phase.

[1] The top ($\top$) erases the formulas corresponding to processes that were not executed. We will denote by $!\mathcal{L}[\![\Psi]\!]$ the mapping of $!\mathcal{L}[\![\cdot]\!]$ to elements in Ψ and by $!\triangledown\Delta$ the mapping of $!\triangledown$ to elements in Δ.

The following theorem is an immediate corollary of Theorem 4.1, since ILLF is complete w.r.t. ILL [2].

Theorem 4.2 *(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,*

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

*in ILLF. The level of adequacy is **FCP**.*

Although we did not yet enhance the level of adequacy, the use of focusing introduce some improvements when compared to ILL. More precisely, the next lemma shows the shape of derivations in a proof involving goals: such formulas are derivable from other guards (i.e., the encoding of constraints) and non-logical axioms in Δ only. Actually, the result is even stronger: there is *no proof* of a guard G if a process definition is chosen to be focused on.

Lemma 4.1. *Let G be a guard, $\mathcal{G}$ be a set of atomic guards, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then $\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \to [G]$ has a proof in ILLF if and only if $\mathcal{G}, \triangledown\Delta : \cdot; \cdot \to [G]$ is provable in ILLF. Moreover, if $\forall\overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!] \in \mathcal{L}[\![\Psi]\!]$ then the following sequent is not provable*

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

Proof. Consider a proof of $\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \to [G]$. Note that all the formulas in G are strictly positive and the connectives $\exists$ and $\otimes$ (on the right) do not lose focus. Hence, we end up either with a sequent $\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \to [1]$ (that finishes immediately) or with a derivation of the shape

$$\dfrac{\overset{\pi}{\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \Rightarrow A}}{\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \to [!\,A]}\ !_R$$

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

$$\dfrac{\dfrac{\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : [\mathcal{L}[\![P]\!]]; \cdot \to G \quad \mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \to [p(t)]}{\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : [\forall\overline{x}.p(\overline{x}) \multimap \mathcal{L}[\![P]\!]]; \cdot \to G}\ \forall_L, \multimap_L}{\mathcal{L}[\![\Psi]\!], \mathcal{G}, \triangledown\Delta : \cdot; \cdot \Rightarrow G}\ D_{LC}$$

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

Now let us explain why, even with focusing, we do not obtain an adequacy at the **FCD** level. Let $P = \textbf{ask}\ c\ \textbf{then}\ P'$. Hence focusing on $\mathcal{L}[\![P]\!]$ would produce the derivation

$$\dfrac{\dfrac{\overset{\pi_1}{\Upsilon : \Gamma, [\mathcal{L}[\![P']\!]]; \cdot \to G} \quad \overset{\pi_2}{\Upsilon : \cdot; \cdot \to [\triangledown c]}}{\Upsilon : \Gamma, [\triangledown c \multimap \mathcal{L}[\![P']\!]]; \cdot \to G}\ \forall_L, \multimap_L}{\Upsilon : \Gamma; \cdot \Rightarrow G}\ D_L$$

Observe that c is a guard, hence $\triangledown c$ will be decomposed entirely into subformulas of the shape 1 and $!\,A$. From Lemma 4.1 we know that the proof of $!\,A$ must proceed by using constraints and non-logical axioms only, matching *exactly* the semantics given by rule $\mathrm{R_A}$. However, note that $\mathcal{L}[\![P']\!]$ is also focused on the left. Operationally, this means that, if P' is a positive action (i.e., an ask), P' has to be executed immediately, which is not enforced by CCP's operational semantics. This will be analyzed in the next section.

Another important aspect to consider is the use of non-logical axioms. Although this is not an operational step in CCP, focusing on such axioms counts as a focused step in ILLF. However, observe that focusing over a non-logical axiom necessarily produces a derivation of the following type

$$\dfrac{\dfrac{\overset{\pi_1}{\Upsilon : \cdot; \cdot \to [\triangledown c]} \quad \dfrac{\overset{\pi_2}{\dfrac{\Upsilon, \mathcal{A}_d : \Gamma; \cdot \Rightarrow G}{\Upsilon : \Gamma; \triangledown d \Rightarrow G}}}{\Upsilon : \Gamma, [\triangledown d]; \cdot \to G}\ R_l}{\Upsilon : \Gamma, [\forall\overline{x}.\ \triangledown c \multimap \triangledown d]; \cdot \to G}\ \forall_L, \multimap_L$$

where π_1 is the proof of $\triangledown c$ from the classical formulas in Υ, and $\mathcal{A}_d$ is the multi-set of atomic subformulas of $\triangledown d$. If G is not atomic and the last rule applied in π_2 is D_R, then Γ must be empty, G will be completely decomposed into its atomic subformulas and the derivation above can be transformed into

$$\dfrac{\dots \quad \dfrac{\dfrac{\pi_1}{\Upsilon : \cdot; \cdot \to [\triangledown c]} \quad \dfrac{\dfrac{\dfrac{\pi_2}{\Upsilon, \mathcal{A}_d : \cdot; \cdot \Rightarrow A}}{\Upsilon : \cdot; \triangledown d \Rightarrow A}\, R_l}{\Upsilon : [\triangledown d]; \cdot \to A}}{\Upsilon : [\triangledown c \multimap \triangledown d]; \cdot \to A}\, \multimap_L}{\Upsilon : \cdot; \cdot \Rightarrow A}\, D_{LC} \quad \dots}{\Upsilon : \cdot; \cdot \to [G]}\, D_R$$

$$\Upsilon : \cdot; \cdot \Rightarrow G$$

where A is an atomic subformula of G. That is, non-logical axioms permute up, and it is always possible to apply them at the top of proofs. The same happens if any left rule is applied in π_2. Hence we can mimic, in logic, the same behavior of using non-logical axioms only for entailing constraints.

From now on, we shall assume that non-logical axioms are applied just before the leafs of proofs and we will not count this step in our adequacy theorems.

4.1. Maximal derivations and interleaving in ask agents

In this section we show how to obtain the highest level of adequacy when considering standard derivations in the operational semantics of CCP.

Example 4.1 *(Traces, proofs and focusing).* Consider the CCP process

$$P \;=\; \mathbf{tell}(a \wedge b) \parallel \mathbf{ask}\ a\ \mathbf{then\ ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel \mathbf{ask}\ b\ \mathbf{then\ ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}')$$

We denote the two external ask agents in P as A_1 and A_2 respectively. The operational semantics dictates that there are three possible transitions leading to the final store $d = a \wedge b \wedge \mathtt{ok} \wedge \mathtt{ok}'$. All such transitions start with the negative action $\mathbf{tell}(a \wedge b)$:

$$\begin{aligned}
\mathtt{Trace\ 1}: \langle \emptyset; P; \mathtt{true}\rangle &\longrightarrow \langle \emptyset; A_1 \parallel A_2; a \wedge b\rangle \longrightarrow \langle \emptyset; \mathbf{ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel A_2; a \wedge b\rangle \\
&\longrightarrow \langle \emptyset; \mathbf{tell}(\mathtt{ok}) \parallel A_2; a \wedge b\rangle \longrightarrow \langle \emptyset; A_2; a \wedge b \wedge \mathtt{ok}\rangle \longrightarrow^* \langle \emptyset; \cdot; d\rangle \not\longrightarrow \\
\mathtt{Trace\ 2}: \langle \emptyset; P; \mathtt{true}\rangle &\longrightarrow \langle \emptyset; A_1 \parallel A_2; a \wedge b\rangle \longrightarrow \langle \emptyset; A_1 \parallel \mathbf{ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}'); a \wedge b\rangle \\
&\longrightarrow \langle \emptyset; A_1 \parallel \mathbf{tell}(\mathtt{ok}'); a \wedge b\rangle \longrightarrow \langle \emptyset; A_1; a \wedge b \wedge \mathtt{ok}'\rangle \longrightarrow^* \langle \emptyset; \cdot; d\rangle \not\longrightarrow \\
\mathtt{Trace\ 3}: \langle \emptyset; P; \mathtt{true}\rangle &\longrightarrow \langle \emptyset; A_1 \parallel A_2; a \wedge b\rangle \longrightarrow \langle \emptyset; \mathbf{ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel A_2; a \wedge b\rangle \\
&\longrightarrow \langle \emptyset; \mathbf{ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel \mathbf{ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}'); a \wedge b\rangle \\
&\longrightarrow \langle \emptyset; \mathbf{tell}(\mathtt{ok}) \parallel \mathbf{ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}'); a \wedge b\rangle \longrightarrow \langle \emptyset; \mathbf{tell}(\mathtt{ok}) \parallel \mathbf{tell}(\mathtt{ok}'); a \wedge b\rangle \\
&\longrightarrow^* \langle \emptyset; \cdot; d\rangle
\end{aligned}$$

`Trace 1` and `Trace 2` correspond exactly to a different focused proof of the sequent $\mathcal{L}[\![P]\!] \longrightarrow \mathcal{L}[\![d]\!]$: one focusing first on $\mathcal{L}[\![A_1]\!]$ and the other focusing first on $\mathcal{L}[\![A_2]\!]$. On the other hand, `Trace 3` corresponds to an *interleaved* execution of A_1 and A_2. We note that such a 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 $\triangledown a \multimap \triangledown b \multimap \triangledown \mathtt{ok}$ producing the focused formula $\triangledown b \multimap \triangledown \mathtt{ok}$, which is still negative. Hence focusing cannot be lost and the inner ask has to be triggered.

Remark 4.1. 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. In fact, if we allow processes to consume constraints [15], an interleaving execution as the one in `Trace 3` may not output the constraint `ok`, since agents may compete for the same resources.

On looking for a 1–1 correspondence between proofs in ILLF and operational steps, one has two options: (1) to force the operational semantics to execute at once nested ask agents, thus ruling out behaviors as the one in `Trace 3` above; or (2) to introduce the so called logical delays in the encoding of $\mathcal{L}[\![\cdot]\!]$, allowing ILLF derivations to mimic `Trace 3`. In the following we explore option (1) and in Section 4.2 we explore (2).

Next definition introduces an operational rule that allows executing at once nested ask processes. The use of this rule gives rise to what we call *maximal derivations*. Since the language we are dealing with is deterministic, it is not difficult to show that the new semantics coincides with the one in Section 2.3 (Theorem 4.3).

Definition 4.1 *(Standard and maximal derivations and observables).* A derivation in CCP using the relation $\longrightarrow$ as in Fig. 1 is called *standard*. The *maximal* transition relation $\rightsquigarrow$ is a standard derivation obtained by replacing the rules R_A and R_C with the rules R_{AM} and R_{CM}, respectively

$$\frac{d \vdash_\Delta c_1 \wedge \ldots \wedge c_n \quad \star}{(X;\, P, \Gamma;\, d) \rightsquigarrow (X;\, Q\,, \Gamma;\, d)}\ \mathrm{R_{AM}} \qquad \frac{p(\overline{x}) \triangleq P \quad d \vdash_\Delta (c_1 \wedge \ldots \wedge c_n)[\overline{y}/\overline{x}] \quad \star}{(X;\, p(\overline{y}), \Gamma;\, d) \longrightarrow (X;\, Q\,[\overline{y}/\overline{x}], \Gamma;\, d)}\ \mathrm{R_{CM}}$$

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

$$\textbf{ask}\ c_1\ \textbf{then ask}\ c_2\ \textbf{then}\ \ldots \textbf{ask}\ c_n\ \textbf{then}\ Q \tag{1}$$

and Q is not an ask agent. In $\mathrm{R_{CM}}$, if P is not an ask agent then $n = 0$ and the side condition $d \vdash_\Delta (c_1 \wedge \ldots \wedge c_n)[\overline{y}/\overline{x}]$ become the trivial assertion $d \vdash_\Delta \mathtt{true}$. We define the observables of a process under the $\rightsquigarrow$ relation, denoted by $(X; \Gamma; d) \Downarrow\!\!\Downarrow_c$, 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 `Trace 3` of Example 4.1. On the contrary, in a maximal derivation $\gamma \rightsquigarrow \gamma'$, an ask having the form of Equation (1) above has to wait until all the guards $c_1, \ldots, c_n$ can be entailed from the store. Then, it executes Q in *one* step (as in `Trace 1` and `Trace 2` of Example 4.1).

The next theorem relates the outputs obtained by the standard and the maximal semantics.

Theorem 4.3. *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, assume that $(X_1; \Gamma_1; e_1) \Downarrow_c$, i.e., there is a derivation of the shape $(X_1; \Gamma_1; e_1) \longrightarrow \cdots \longrightarrow (X_m; \Gamma_m; e_m)$ such that $\exists X_m. e_m \vdash_\Delta c$. We note that the store evolves monotonically, i.e., $e_i \vdash_\Delta e_j$ for all $j \leq i$. Moreover, ask agents query the store without affecting it (Rule $\mathrm{R_A}$). Consider a process $R = \textbf{ask}\ c_1\ \textbf{then ask}\ c_2\ \textbf{then}\ \ldots \textbf{ask}\ c_n\ \textbf{then}\ Q$ in a given configuration in the derivation above. We know that R can add information to the store iff Q is executed. This happens iff there is a configuration γ_k with store e_k entailing the last guard of R (i.e., $e_k \vdash_\Delta c_n$). By monotonicity and Rule $\mathrm{R_A}$, it must be the case that $e_k \vdash_\Delta (c_1 \wedge \ldots \wedge c_n)$. We conclude by noticing that there is a (maximal) derivation using $\mathrm{R_{AM}}$ ending in the store e_m. Such derivation differs from the standard derivation in that it delays the execution of R until the store e_k is produced. The same argumentation follows for procedure calls and the rule $\mathrm{R_{CM}}$. □

Remark 4.2. Interestingly enough, the theorem above reflects a well-known proof theoretical result: negative rules permute down with positive rules, meaning that all negative actions can be done before the positive ones.

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

Theorem 4.4 *(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,*

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

*in ILLF. Moreover, the level of adequacy is **FCD**.*

Proof. The result follows immediately from Remark 4.2, since positive and negative actions in CCP with maximal derivations correspond *exactly* to positive and negative phases in the logical system. In particular, observe that, in the process **ask** c **then** P, the only possibility of $\mathcal{L}[\![P]\!]$ being a negative formula is when P is also an ask. The same with procedure calls. □

4.2. Interleaving and delays

Interleaving executions as the one in `Trace 3` can be handled in a focused system by means of the so called *logical delays* [29].

Definition 4.2. The positive and negative delay operators $\delta^+(\cdot), \delta^-(\cdot)$ are defined as $\delta^+(F) = F \otimes 1$ and $\delta^-(F) = 1 \multimap F$ respectively.

Observe that $\delta^+(F) \equiv \delta^-(F) \equiv F$, hence delays can be used in order to replace a formula with a provably equivalent formula of a given polarity. We define the encoding $\mathcal{L}[\![\cdot]\!]_+$ as $\mathcal{L}[\![\cdot]\!]$ but replacing the following cases:

$$\begin{aligned} \mathcal{L}[\![\textbf{ask}\ c\ \textbf{then}\ P]\!]_+ &= \triangledown c \multimap \delta^+(\mathcal{L}[\![P]\!]_+) \\ \mathcal{L}[\![p(\overline{x}) \triangleq P]\!]_+ &= \forall \overline{x}. p(\overline{x}) \multimap \delta^+(\mathcal{L}[\![P]\!]_+) \end{aligned}$$

The use of the encoding above forces the focusing phase to end. In this case, we can have a stronger adequacy theorem for the whole CCP system.

Theorem 4.5 *(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,*

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

*in ILLF. The adequacy level is **FCD**.*

Proof. By straightforward case analysis. To illustrate, consider the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{L}[\![\Psi]\!], \triangledown\Delta : \cdot; \mathcal{L}[\![P]\!] \Rightarrow G}{\mathcal{L}[\![\Psi]\!], \triangledown\Delta : \cdot; \delta^+(\mathcal{L}[\![P]\!]) \Rightarrow G}\otimes_L, 1_L}{\mathcal{L}[\![\Psi]\!], \triangledown\Delta : [\delta^+(\mathcal{L}[\![P]\!])]; \cdot \rightarrow G}R_L \qquad \mathcal{L}[\![\Psi]\!], \triangledown\Delta : \cdot; \cdot \rightarrow [\triangledown c]}{\mathcal{L}[\![\Psi]\!], \triangledown\Delta : [\triangledown c \multimap \delta^+(\mathcal{L}[\![P]\!]_+)]; \cdot \rightarrow G}\forall_L, \multimap_L}{\mathcal{L}[\![\Psi]\!], \triangledown\Delta : \cdot; \cdot \Rightarrow G}D_{LC}$$

Observe that the positive formula $\delta^+(\mathcal{L}[\![P]\!]_+)$ forces the end of the positive phase on the left premise. □

Remark 4.3. Observe that Theorem 4.4 gives a canonical trace to CCP 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 4.5 shows that derivation in logic have a one-to-one correspondence with traces of a computation in a CCP program.

4.3. A closer view to negative actions

So far we have shown that the focus discipline and the use of delays allow us to have a one-to-one correspondence between CCP positive actions (i.e., ask agents and procedure calls) and ILLF positive phases. In this section we show in detail the behavior of negative actions as well.

Recall that negative actions are those that do not need to interact with the environment, i.e., parallel composition, local and tell processes in CCP. From the logical point of view, the formulas resulting in ILL from the encoding of those processes are all introduced (in any order) in a negative phase since the rules governing them are all invertible ($\otimes_L, \exists_L, !_L$).

We thus have two sources of *don't care* non-determinism that may separate derivations in CCP and the corresponding derivations in ILLF:

1. Let $P = \mathbf{tell}(c) \parallel \mathbf{ask}\ c\ \mathbf{then}\ Q \parallel \mathbf{tell}(d)$. P may proceed by adding c to the store, then reducing the ask agent to Q and finally adding d to the store. The ILLF derivations for $\mathcal{L}[\![P]\!]$ starts with a negative phase decomposing $\triangledown c$ and $\triangledown d$. Then it focuses on the encoding of the ask agent to produce Q. In other words, the focusing discipline forces to postpone the execution of the ask agent after introducing both c and d.
2. The operational semantics stores c in $\mathbf{tell}(c)$ to the current store d producing $c \wedge d$ in one step. The encoding $\triangledown c$ may use several application of the negative rules $\otimes_L, \exists_L, !_L$ until the atomic subformulas of c are stored into the context.

There are at least two options for having also a match between negative actions in CCP and derivations in ILLF.

(i) Propose a new semantics that eagerly executes all the negative actions (in a parallel composition) and then works on positive actions. It is not difficult then proving an adequacy result similar to Theorem 4.3 for such semantics. For that, note that if $\mathbf{ask}\ c\ \mathbf{then}\ P$ can exhibit a transition on a store d then it can do the same on a stronger store d' (i.e., $d' \vdash_\Delta d$).

(ii) Introduce negative delays in the encoding of tells and local processes. Hence they must be introduced in a positive (focused) phase and the only negative actions will be the steps needed to store the constraints in the context. In this way, we have a perfect match between polarity changes in ILLF and operational steps.

We shall explore in detail the alternative (ii) in Section 5 where we analyze a timed extension of CCP. In that language, the logical adequacy cannot be established without forcing all the actions to be positive.

4.4. Indeterminate CCP languages

Non-determinism is introduced in CCP by means of the choice operator.

Definition 4.3 *(Indeterminate CCP).* Indeterminate CCP processes are obtained by adding $\sum_I P_i$ to the syntax in Definition 2.2.

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

(a) Blind Choice (b) One-step guarded choice

Fig. 4. Operational rules for the non-deterministic choice.

The process $\sum_I P_i$ chooses one of 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 interpretations of the non-deterministic choice as shown in Fig. 4. The Rule $\mathrm{R_{BCH}}$ corresponds to the blind (or internal) choice (BC). This rule says that a process P_i is chosen for execution regardless whether the process P_i can evolve or not. The local choice fragment of indeterminate CCP enjoys interesting properties as confluence, i.e., the output of a process is independent of the scheduling policy of the parallel operator [16].

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 usually written as a summation of ask agents as in $\sum_i$ **ask** c_i **then** P_i. Thus, the 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 the guard $c_i = \mathtt{true}$ for all i.

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

$$\mathcal{L}[\![\sum_I P_i]\!]_{BC} \;=\; \&_I(\delta^+(\mathcal{L}[\![P_i]\!])) \qquad \mathcal{L}[\![\sum_I P_i]\!]_{GC} \;=\; \&_I\mathcal{L}[\![P_i]\!]$$

Observe that 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 in 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 can be inferred from the context (store) and non-logical axioms. Hence, we continue having a neat logical control corresponding to the operational semantics, and results in Theorems 4.4 and 4.5 remain valid for indeterminate CCP.

Theorem 4.6 *(Adequacy for indeterminate CCP). Let P be a process from Definition 4.3, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, for any constraint c:*

- $P \downarrow\downarrow_c$ *iff there is a proof of the sequent* $\mathcal{L}[\![\Psi]\!], \triangledown\Delta : \cdot; \mathcal{L}[\![P]\!] \Rightarrow \triangledown c \otimes \top$
- $P \Downarrow_c$ *iff there is a proof of the sequent* $\mathcal{L}[\![\Psi]\!]_+, \triangledown\Delta : \cdot; \mathcal{L}[\![P]\!]_+ \Rightarrow \triangledown c \otimes \top$

Moreover, the level of adequacy in both cases is ***FCD****.*

Negative actions. The discussion on Section 4.3 takes a new perspective in the case of guarded choices. Note that GC processes are affected by the scheduling policy of the parallel operator. This means that, in general, GC processes are not confluent in the sense of [16]. Take for instance the process $P =$ **tell**$(c) \parallel Q$ where $Q =$ **ask** $\mathtt{true}$ **then** $Q_1 +$ **ask** c **then** Q_2. If Q is first chosen for execution, the only alternative for Q is to execute Q_1. On the contrary, if **tell**(c) is first executed, Q may chose both Q_1 or Q_2.

Alternatives (i) and (ii) proposed in Section 4.3 (in order to exhibit an adequacy result between ILLF and both negative and positive actions) are also applicable in the context of indeterminate CCP. The semantic approach in (i) is valid since augmenting the store enables more choices in a GC process (i.e., an alternative cannot be discarded by considering a stronger store); on the other hand, by adding negative delays as in (ii), we can recover a one-to-one correspondence between both positive and negative actions in indeterminate CCP and ILLF.

5. Maximal parallelism and multi-focusing

The reduction relations $\longrightarrow$ (Definition 2.3) and $\rightsquigarrow$ (Definition 4.1) define interleaving semantics for the parallel operator. This means that, in a configuration of the shape $\gamma = \langle X; P_1 \parallel \cdots \parallel P_n; d\rangle$, if $\gamma \longrightarrow \gamma'$, then only one of the P_i exhibits a transition (and the other processes remain the same).

In this section, we show how to capture *maximal parallelism* in CCP using the so called maximal multi-focusing discipline in ILL. By maximal parallelism we mean that *all* the *enabled* processes are executed at the same time. As we shall see, this kind of semantics is needed for characterizing timed CCP processes [14]. Finally, we establish adequate ILL characterizations to the systems mentioned above.

5.1. Multi-focused ILL

We start by presenting mILL, a multi-focused system for ILL, which is an extension of the system mLJF introduced in [40] or the restriction to the positive intuitionistic case of the system presented in [5].

Positive Phase

$$\frac{\Upsilon : \Gamma_1; \cdot \to [P] \quad \Upsilon : \Gamma_2; \cdot \to [Q]}{\Upsilon : \Gamma_1, \Gamma_2; \cdot \to [P \otimes Q]} \otimes_r \qquad \frac{\Upsilon : \Gamma, \Psi, [N_i]; \cdot \to P}{\Upsilon : \Gamma, \Psi, [N_1 \,\&\, N_2]; \cdot \to P} \,\&_{Li}$$

$$\frac{\Upsilon : \Gamma_1; \cdot \to [P] \quad \Upsilon : \Gamma_2, \Psi, [N]; \cdot \to Q}{\Upsilon : \Gamma_1, \Gamma_2, \Psi, [P \multimap N]; \cdot \to Q} \multimap_l \qquad \frac{\Upsilon : \Gamma; \cdot \to [P_i]}{\Upsilon : \Gamma; \cdot \to [P_1 \oplus P_2]} \otimes_{Ri}$$

$$\frac{\Upsilon : \Gamma; \cdot \to [P\{t/x\}]}{\Upsilon : \Gamma; \cdot \to [\exists x P]} \exists_r \qquad \frac{\Upsilon : \Gamma, \Psi, [N\{t/x\}]; \cdot \to P}{\Upsilon : \Gamma, \Psi, [\forall x N]; \cdot \to P} \forall_l$$

$$\frac{\Upsilon : \cdot; \cdot \Rightarrow N}{\Upsilon : \cdot; \cdot \to [!N]} \,!_R \qquad \frac{}{\Upsilon : \cdot; \cdot \to [1]} \,1_R$$

$$\frac{}{\Upsilon : \Gamma; \cdot \to [A]} \; I_R \text{ given } A \in (\Gamma \cup \Upsilon) \text{ and } (\Gamma \subseteq \{A\})$$

Structural Rules

$$\frac{\Upsilon : \Gamma, [\Upsilon^*, \Phi]; \cdot \to P}{\Upsilon : \Gamma, \Phi; \cdot \Rightarrow P} \; mD_L \qquad \frac{\Upsilon : \Gamma; \cdot \to [P]}{\Upsilon : \Gamma; \cdot \Rightarrow P_u} \; mD_R$$

$$\frac{\Upsilon : \Gamma; \Theta \Rightarrow P}{\Upsilon : \Gamma, [\uparrow \Theta]; \cdot \to P} \; mR_L \qquad \frac{\Upsilon : \Gamma; \cdot \Rightarrow N}{\Upsilon : \Gamma; \cdot \to [\downarrow N]} \; mR_R$$

$$\frac{\Upsilon : \Gamma, \Phi; \Theta \Rightarrow R}{\Upsilon : \Gamma; \Theta, \downarrow \Phi \Rightarrow R} \; \texttt{store}_N \qquad \frac{\Upsilon : \Gamma, A; \Theta \Rightarrow R}{\Upsilon : \Gamma; \Theta, A_a \Rightarrow R} \; \texttt{store}_A$$

Fig. 5. mILL system. Here A is atomic, P, Q positive, N negative, P_u represents either a negative formula of the kind $\uparrow P$ or a positive formula and A_a is either atomic or a formula of the kind $\uparrow A$. Υ^* represents a multiset of arbitrary numbers of copies of formulas in Υ. In mD_L, $\Upsilon^* \cup \Phi$ is non-empty.

The system mILL has two kinds of formulas:

$$P, Q \;:=\; A \mid 1 \mid P \otimes Q \mid P \oplus Q \mid \exists x.P(x) \mid \,!N \mid \downarrow N$$
$$M, N \;:=\; \top \mid M \,\&\, N \mid P \multimap N \mid \forall x.N(x) \mid \uparrow P$$

where P, Q are positive while M, N are negative formulas. The symbols $\uparrow$ and $\downarrow$ mark the changing of polarities. The syntax for contexts is the following

$$\Phi := \Phi, N \qquad \Upsilon := \cdot \mid \Phi \qquad \Gamma := \Upsilon, A \qquad \Psi := [\Phi] \qquad \Theta := \Upsilon, P$$

The intuition is the following: Φ, Υ, Γ are sets that contain negative formulas, where Φ is non-empty and Γ may also have atomic formulas; Ψ is a non-empty set of focused negative formulas, and Θ is any set of formulas.

There are three kind of sequents in mILL:

- the sequent $\Upsilon : \Gamma; \Theta \Rightarrow R$ is unfocused;
- the sequent $\Upsilon : \Gamma, [\Phi]; \cdot \to R$ is multi-focused on the left;
- the sequent $\Upsilon : \Gamma; \cdot \to [R]$ is focused on the right.

The negative phase in mILL is the same as in ILL. The rest of the rules for mILL are similar to the ones presented in Fig. 2, now considering possibly multi-focused contexts (Fig. 5). The only difference is the way of triggering the release rules: the positive phase ends only when all the focused formulas are marked with arrows. This means that the focusing persists up to the point that every focused formula reaches a negative formula, and the focusing is released at once to all of them.

Theorem 5.1. *mILL is sound and complete with respect to ILL.*

Proof. The proof is standard: just note that erasing the $\uparrow$ and $\downarrow$ arrows and the context Ψ and restricting Φ to a singleton in mD_L, mILL collapses to ILLF. □

5.2. Maximal multi-focusing

The following are adapted versions of definitions in [10,9].

Definition 5.1. The proofs Ξ_1 and Ξ_2 of the same mILL sequent are *locally permutable equivalent*, written $\Xi_1 \sim \Xi_2$, if each can be rewritten to the other using intra-phase permutations, that is, permutations inside a phase step. Ξ_1 and Ξ_2 are *permutable equivalent*, written $\Xi_1 \approx \Xi_2$, if they are locally permutable equivalent and each can be rewritten to the other using inter-phase permutations, *i.e.*, permutations of rules in different phase steps.

Observe that, since all negative rules are invertible, they permute over any other rule. This means that the whole negative phase collapse to one step, modulo local permutations. Non-locally permutable equivalent proofs, on the other hand, require considering permutations of entire phases.

Example 5.1. Let A, B, C, D (positive) atomic formulas and suppose that $\Upsilon : \Gamma_1, \Gamma_2, A \multimap (\uparrow B), C \multimap (\uparrow D); \cdot \Rightarrow R$ is provable with derivation Ξ_1

$$\dfrac{\dfrac{}{\Upsilon : \Gamma_1; \cdot \to [A]}\, I_R \qquad \dfrac{\dfrac{\dfrac{}{\Upsilon : \Gamma_2'; \cdot \to [C]}\, I_R \quad \dfrac{\overset{\pi}{\Upsilon : \Gamma_2'', B, D; \cdot \Rightarrow R}}{\Upsilon : \Gamma_2'', B, [\uparrow D]; \cdot \to R}\, mR_L}{\Upsilon : \Gamma_2, B, [C \multimap (\uparrow D)]; \cdot \to R}\, \multimap_L}{\Upsilon : \Gamma_2, C \multimap (\uparrow D), B; \cdot \Rightarrow R}\, mD_L \quad \dfrac{}{\Upsilon : \Gamma_2, C \multimap (\uparrow D), [\uparrow B]; \cdot \to R}\, mR_L}{\Upsilon : \Gamma_1, \Gamma_2, C \multimap (\uparrow D), [A \multimap (\uparrow B)]; \cdot \to R}\, \multimap_L$$
$$\dfrac{\Upsilon : \Gamma_1, \Gamma_2, C \multimap (\uparrow D), [A \multimap (\uparrow B)]; \cdot \to R}{\Upsilon : \Gamma_1, \Gamma_2, A \multimap (\uparrow B), C \multimap (\uparrow D); \cdot \Rightarrow R}\, mD_L$$

The derivation Ξ_2 below is also a proof of $\Upsilon : \Gamma_1, \Gamma_2, A \multimap (\uparrow B), C \multimap (\uparrow D); \cdot \Rightarrow R$ and $\Xi_1 \approx \Xi_2$:

$$\dfrac{\dfrac{}{\Upsilon : \Gamma_1; \cdot \to [A]}\, I_R \quad \dfrac{\dfrac{\overset{\pi}{\Upsilon : \Gamma_2'', D, B; \cdot \Rightarrow R}}{\Upsilon : \Gamma_2'', D, [\uparrow B]; \cdot \to R}\, mR_L}{\Upsilon : \Gamma_1, \Gamma_2'', D, [A \multimap (\uparrow B)]; \cdot \to R}\, \multimap_L}{\Upsilon : \Gamma_1, \Gamma_2'', A \multimap (\uparrow B), D; \cdot \Rightarrow R}\, mD_L \qquad \dfrac{}{\Upsilon : \Gamma_2'; \cdot \to [C]}\, I_R \quad \dfrac{}{\Upsilon : \Gamma_1, , \Gamma_2'', A \multimap (\uparrow B), [\uparrow D]; \cdot \to R}\, mR_L$$
$$\dfrac{\Upsilon : \Gamma_1, \Gamma_2, A \multimap (\uparrow B), [C \multimap (\uparrow D)]; \cdot \to R}{\Upsilon : \Gamma_1, \Gamma_2, A \multimap (\uparrow B), C \multimap (\uparrow D); \cdot \Rightarrow R}\, mD_L \qquad \multimap_L$$

Observe that we have exchanged the order of application of the implication left rule, hence performing *inter-phase* steps.

Definition 5.2. If a proof Ξ in mILL ends with an instance of mD_L, let $\texttt{foci}(\Xi)$ be defined as the multiset of focused formulas in the premise of that instance. We say that this instance of mD_L is *maximal* if and only if, for every $\Xi' \approx \Xi$, $\texttt{foci}(\Xi') \subseteq \texttt{foci}(\Xi)$. A proof in mILL is *maximal* if and only if every instance of mD_L in it is maximal.

In Example 5.1, Ξ_1 and Ξ_2 are permutable equivalent to the following maximal multi-focused proof

$$\dfrac{\dfrac{}{\Upsilon : \Gamma_1; \cdot \to [A]}\, I_R \quad \dfrac{\dfrac{}{\Upsilon : \Gamma_2'; \cdot \to [C]}\, I_R \quad \dfrac{\overset{\pi}{\Upsilon : \Gamma_2'', B, D; \cdot \Rightarrow R}}{\Upsilon : \Gamma_2'', [\uparrow B, \uparrow D]; \cdot \to R}\, mR_L}{\Upsilon : \Gamma_2, [C \multimap (\uparrow D), \uparrow B]; \cdot \to R}\, \multimap_L}{\Upsilon : \Gamma_1, \Gamma_2, [C \multimap (\uparrow D), A \multimap (\uparrow B)]; \cdot \to R}\, \multimap_L$$
$$\dfrac{\Upsilon : \Gamma_1, \Gamma_2, [C \multimap (\uparrow D), A \multimap (\uparrow B)]; \cdot \to R}{\Upsilon : \Gamma_1, \Gamma_2, C \multimap (\uparrow D), A \multimap (\uparrow B); \cdot \Rightarrow R}\, mD_L$$

Note that Ξ_1 and Ξ_2 are also permutable equivalent to the maximal multi-focused proof where $C \multimap (\uparrow D)$ is reduced first. But these maximal proofs are locally permutable equivalent: the inter-phase permutations are now intra-phase permutations.

As in [2], we call a neighboring pair of phases, with the bottom phase positive and the top phase negative, a *bipole*.

Theorem 5.2. *Every provable sequent in mILL has a maximal proof. Moreover, restricted to the* `CCP Grammar` *(see Section 4) any two maximal multi-focused proofs of the same sequent are locally permutable equivalent.*

Proof. Consider two neighboring bipoles in mILL. If the positive phase of the top bipole permutes with the negative phase of the bottom bipole, then, in an unfocused form, we can perform the permutation and merge the two bipoles. This is done by uniting their positive and negative phases, thus obtaining another (multi-)focused proof. This operation obviously terminates, giving rise to a maximal multi-focused proof. For the unicity result, if we restrict the formulas so that they fall into the `CCP Grammar`, then the proof is the same as done for MALL in [10]. While the unicity should also hold for the whole mILL system, proving this result is out of the scope of this paper. □

$$\frac{(X;P;c) \cong (X';P';c') \longrightarrow (Y';Q';d') \cong (Y;Q;d)}{(X;P;c) \longrightarrow (Y;Q;d)}\ \mathrm{R}_{\mathrm{EQUIV}}$$

$$\frac{}{(X;\mathbf{tell}(c);d) \longrightarrow (X;\mathbf{stop};c \wedge d)}\ \mathrm{R}_{\mathrm{T}} \qquad \frac{d \vdash_\Delta c_i}{\langle X;\sum_{i\in I}\mathbf{ask}\ c_i\ \mathbf{then}\ P_i;d\rangle \longrightarrow \langle X;P_i;d\rangle}\ \mathrm{R}_{\mathrm{A}}$$

$$\frac{p(\vec{x}) \triangleq P}{(X;p(\vec{y});d) \longrightarrow (X;P[\vec{y}/\vec{x}];d)}\ \mathrm{R}_{\mathrm{C}}$$

$$\frac{(X;P;d) \longrightarrow (X \cup \vec{x_1};P';d \wedge c_1) \quad (X;Q;d) \longrightarrow (X \cup \vec{x_2};Q';d \wedge c_2) \qquad \vec{x_1} \cap \vec{x_2} = \emptyset}{(X;P \parallel Q;d) \longrightarrow (X \cup \vec{x_1} \cup \vec{x_2};P' \parallel Q';d \wedge c_1 \wedge c_2)}\ \mathrm{R}_{\mathrm{P}}1$$

$$\frac{(X;P;d) \longrightarrow (X \cup \vec{x_1};P';d \wedge c_1) \qquad (X;Q;d) \not\longrightarrow}{(X;P \parallel Q;d) \longrightarrow (X \cup \vec{x_1};P' \parallel Q;d \wedge c_1)}\ \mathrm{R}_{\mathrm{P}}2$$

Fig. 6. Operational semantics of tccp.

5.3. Timed CCP processes

Reactive systems (see e.g., [4]) are those that react continuously with their environment at a rate controlled by the environment. For example, a controller or a signal-processing system, receives a stimulus (input) from the environment, computes an output and then waits for the next interaction with the environment. Temporal extensions of CCP has been proposed [43,28,4,37] in order to specify and verify reactive systems combining the elegant computation model of CCP with ideas from the paradigm of Synchronous Languages [4].

The tccp process calculus [14] is an orthogonal timed non-deterministic extension of CCP. In this language, time is identified with the time needed to ask and tell information to the store.

Definition 5.3 *(tccp processes).* Processes are built from constraints in the underlying constraint system as follows[2]:

$$P, Q ::= \mathbf{stop} \mid \mathbf{tell}(c) \mid \sum_{i\in I}\mathbf{ask}\ c_i\ \mathbf{then}\ P_i \mid P \parallel Q \mid (\mathbf{local}\, x)\ P \mid p(\vec{x})$$

The operational semantics is defined in Fig. 6. Similar to Section 2, processes are quotiented by a structural congruence relation $\cong$ satisfying alpha conversion and commutativity and associativity on parallel composition.

We define the congruence relation on configurations as we did in Section 2 (C1) but with an extra ingredient (C2) that will be clarified soon:

C1 $(X;P;c) \cong (X';P';c')$ iff $X = X'$, $P \cong P'$ and $c \equiv_\Delta c'$.
C2 $(X;(\mathbf{local}\, x)\ P;c) \cong (X \cup \{x\};P;c)$ if $x \notin fv(X,c)$

In Fig. 6, a transition of the form $(X;P;c) \longrightarrow (Y;Q;d)$ must be understood as "the process P evolves in *one time-unit* to Q and produces the new store d". Processes in tccp, as in CCP, are monotonic, i.e., in the above derivation it must be the case that $d \vdash_\Delta c$ and $X \subseteq Y$.

The process **stop** represents inaction and there is no a transition from it. Rules R_{T} and R_{A} are similar to those in Section 2, but the transition must be also understood as a temporal evolution. For instance, R_{T} says that the process $\mathbf{tell}(c)$ makes available c to the store only in the next time-unit. Rule R_{C} can be explained similarly.

The novelty with respect to the system in Fig. 1 comes from the rules for local processes and parallel composition.

Recall that in CCP, we identified the multiset of processes $\Gamma = \{P_1, ..., P_n\}$ with the parallel composition $P_1 \parallel \cdots \parallel P_n$. In tccp, all the enabled processes must be executed *at the same time* during a time-unit. Hence, rules $\mathrm{R}_{\mathrm{P}}1$ and $\mathrm{R}_{\mathrm{P}}2$ model the parallel composition operator in terms of *maximal parallelism*: the process $P \parallel Q$ executes, in one time-unit, all the enabled actions in P and Q.

Regarding the rule for the local operator, it was defined in [14] as follows:

$$\frac{(P;c \wedge \exists x.d) \longrightarrow (P';c' \wedge \exists x.d)}{((\mathbf{local}\, x;c)\ P;d) \longrightarrow ((\mathbf{local}\, x;c')\ P';d \wedge \exists x.c')} \tag{2}$$

Here d is the global store and c is the local store containing the information that P accumulates on x. Since the local process cannot observe the information about x in d, the premise of this rule hides this information by using existential quantification. Moreover, since the other processes cannot observe the information about x produced by P (i.e., c'), the rule

[2] We do not consider the process now c then P else Q that executes P if c can be entailed and Q otherwise. The reason is that such operator lacks of a proper proof theoretic semantics: the reduction to Q amounts to showing that there is no proof of c.

hides it by existentially quantifying x in c' before adding it to the global store d. Observe also that this rule says that the process $(\mathbf{local}\, x; c)\ P$ takes one time-unit to perform an action if P also takes one time-unit.

Let $Q = (\mathbf{local}\, x)\ P$ and consider the rule $\mathrm{R_L}$ in Fig. 1. Such rule follows the approach in [15] where local processes simply create a fresh variables (as $\exists_L$ does in logic). If we were to use the rule $\mathrm{R_L}$ (in Fig. 1) for `tccp`, then it would "consume" a time-unit in order to create the fresh variable. This does not correspond to the behavior of local processes in `tccp` as dictated by rule in Equation (2). For fixing that, we embed in the structural rule C2 the action of creating the local variable as a "silent" action (i.e., an action that does not consume a time-unit). Therefore, P is the only responsible for the time passing in $(\mathbf{local}\, x)\ P$.

We define the `tccp` observables as we did for CCP.

Definition 5.4. If $(X_0; P_0; d_0) \longrightarrow (X_1; P_1; d_1) \longrightarrow \cdots \longrightarrow (X_n; P_n; d_n)$ we write $(X_0; P_0; d_0) \longrightarrow^n (X_n; P_n; d_n)$. If there exists $n \geq 0$ such that $(X; \Gamma; d) \longrightarrow^n (X'; \Gamma'; d')$ and $\exists X'.d' \vdash_\Delta c$ we write $(X; \Gamma; d) \Downarrow_c^n$. If $X = \emptyset$ and $d = \mathtt{true}$ we simply write $\Gamma \Downarrow_c^n$.

5.4. Maximal parallelism in `tccp` and maximal multi-focusing

In order to accurately mimic the maximal parallelism semantics in `tccp`, we need all the actions to be positive, as alternative (ii) in Section 4.3 suggests. The only negative action should be the introduction of the eingnevariables for local processes and the action of storing the atoms in $\triangledown c$ to the context. More precisely, we define $\triangledown$ as in Fig. 3 and $\mathcal{T}[\![\cdot]\!]$ as follows:

$$\begin{aligned}
\mathcal{T}[\![\mathbf{tell}(c)]\!] &= \uparrow \triangledown c \\
\mathcal{T}[\![P \parallel Q]\!] &= \downarrow^* (\mathcal{T}[\![P]\!]) \otimes \downarrow^* (\mathcal{T}[\![Q]\!]) \\
\mathcal{T}[\![\textstyle\sum_{i\in I} \mathbf{ask}\, c_i\ \mathbf{then}\ P_i]\!] &= \&_I (\triangledown c_i \multimap \uparrow (\downarrow^* \mathcal{T}[\![P_i]\!]) \\
\mathcal{T}[\![(\mathbf{local}\, x)\ P]\!] &= \exists x.(\downarrow^* \mathcal{T}[\![P]\!]) \\
\mathcal{T}[\![p(\overline{x}) \stackrel{\Delta}{=} P]\!] &= \forall \overline{x}.p(\overline{x}) \multimap \uparrow (\downarrow^* \mathcal{T}[\![P]\!]) \\
\mathcal{T}[\![p(\overline{y})]\!] &= \uparrow p(\overline{y})
\end{aligned}$$

where $\downarrow^*$ denotes a possible presence of $\downarrow$, depending on the polarity of the subformulas, that is, if F is positive then $\downarrow^* F = F$, otherwise $\downarrow^* F = \downarrow F$. For example, $P = (\mathbf{local}\, x)\, \mathbf{tell}(c) \parallel \mathbf{ask}\ d\ \mathbf{then}\ \mathbf{tell}(a)$ is encoded as

$$(\exists x.\downarrow(\uparrow \triangledown c)) \otimes \downarrow(\triangledown d \multimap (\uparrow \triangledown a))$$

Observe that the negative phase will apply eagerly the existential and the tensor rule, as well as store the negative formulas $\uparrow \triangledown c$ and $(\triangledown d \multimap (\uparrow \triangledown a))$ in the negative linear context. Those formulas can be chosen later in a multi-focusing step.

The assignment of arrows intuitively says that immediate subformulas of a formula with positive main connective should be positive. Similarly for negative connectives. The arrows then mark the change of polarity. Although the syntax of formulas seems a little bit more complicated, the rules of mILL are simple enough: one only loses focusing if all focused formulas are marked with an arrow. This is the trigger for changing from positive to negative polarities.

The following theorem shows that passing from a negative to a positive phase in a maximal multi-focused proof in mILL corresponds exactly to the change of a time-unit in the operational semantics.

Theorem 5.3 *(Adequacy for `tccp`). Let P be a `tccp` process, Ψ be a set of process definitions and Δ be a set of non-logical axioms. Then, for any constraint c, $P \Downarrow_c^n$ iff there is a maximal multi-focused proof of the sequent $\mathcal{T}[\![\Psi]\!], \triangledown\Delta : \cdot; \mathcal{T}[\![P]\!] \Rightarrow \triangledown c \otimes \top$ in mILL. Moreover, the level of adequacy is **FCD**, and hence a time unit execution corresponds to a maximal multi-focusing step.*

Proof. Suppose $P = P_1 \parallel \cdots \parallel P_n$. Hence $\mathcal{T}[\![P]\!] = \downarrow^* \mathcal{T}[\![P_1]\!] \otimes \cdots \otimes \downarrow^* \mathcal{T}[\![P_n]\!]$ will be decomposed into $\downarrow^* \mathcal{T}[\![P_1]\!], \cdots, \downarrow^* \mathcal{T}[\![P_n]\!]$ in the negative phase. If $\mathcal{T}[\![P_i]\!]$ is marked with $\downarrow$ for some i, then $\mathcal{T}[\![P_i]\!]$ is negative and it will be stored using the $\mathtt{store}_N$ rule. Otherwise, $\mathcal{T}[\![P_i]\!]$ is positive and it will continue to be decomposed until a formula marked with $\downarrow$ is reached. Note this always eventually happens for any $\mathcal{T}[\![P_i]\!]$. At the end of this process, the linear general context will be empty and a positive phase should start. Since the proof is maximally multi-focused, it should follow by choosing the maximal set of formulas to be focused on the left. This will correspond exactly to choosing a maximal set of processes that can be executed in parallel in one time unit. □

6. Fixed points: procedure calls and a richer language of properties

In this section we explain how procedure calls can be seen as fixed points as in [13], but giving it a new insight under the focusing discipline. We will call μILL the logic ILL with a fixed point operator μ and the unfolding rule

$$\frac{\Upsilon : \Gamma, [B(\mu B)\bar{t}]; \cdot \to G}{\Upsilon : \Gamma, [\mu B \bar{t}]; \cdot \to G}\ \mathtt{unfold}$$

In [3] there is a deep discussion about fixed points, focusing, termination and the admissibility of the rule above. Since here we will use fixed points in a very simple way (e.g., as in [39,29]), we will avoid all these technicalities. Assuming that $p(\overline{x}) \triangleq P$, the encoding $\mathcal{L}[\![\cdot]\!]_\mu$ is the same as $\mathcal{L}[\![\cdot]\!]_+$ (see Section 4.2) but

$$\mathcal{L}[\![p(\overline{t})]\!]_\mu = \mu(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu))\overline{t}$$

The base grammar for guards/goals G and processes P now is the following

$$G := 1 \mid A \mid !\,G \mid G \otimes G \mid \exists x.G$$
$$P := G \mid P \otimes P \mid P \,\&\, P \mid \forall\overline{x}.G \multimap P \mid \exists x.P \mid \mu B\overline{t}$$

Observe that the application of the rule `unfold` matches exactly the behavior of focusing on $\forall\overline{x}.p(\overline{x}) \multimap \delta^+(\mathcal{L}[\![P]\!]_+)$ in our previous encodings. More precisely, the derivation

$$\dfrac{\dfrac{\Upsilon : \Gamma; \delta^+(\mathcal{L}[\![P]\!]_+\{\overline{t}/\overline{x}\}) \Rightarrow G}{\Upsilon : \Gamma, [\delta^+(\mathcal{L}[\![P]\!]_+\{\overline{t}/\overline{x}\})]; \cdot \rightarrow G}\, R_l \quad \dfrac{}{\Upsilon : p(\overline{t}); \cdot \rightarrow [p(\overline{t})]}\, I_R}{\Upsilon : \Gamma, p(\overline{t}), [\forall\overline{x}.p(\overline{x}) \multimap \delta^+(\mathcal{L}[\![P]\!]_+)]; \cdot \rightarrow G}\, \forall_L, \multimap_L$$

now becomes

$$\dfrac{\dfrac{\Upsilon : \Gamma; \delta^+(\mathcal{L}[\![P]\!]_\mu\{\mu(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu))/p\}\{\overline{t}/\overline{x}\}) \Rightarrow G}{\Upsilon : \Gamma, [\delta^+(\mathcal{L}[\![P]\!]_\mu\{\mu(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu))/p\}\{\overline{t}/\overline{x}\})]; \cdot \rightarrow G}\, R_L}{\Upsilon : \Gamma, [\mu(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu))\overline{t}]; \cdot \rightarrow G}\, \texttt{unfold}$$

since

$$(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu))(\mu(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu)))\overline{t} =_\beta$$
$$\delta^+(\mathcal{L}[\![P]\!]_\mu\{\mu(\lambda p.\lambda\overline{x}.\delta^+(\mathcal{L}[\![P]\!]_\mu))/p\}\{\overline{t}/\overline{x}\})$$

The following adequacy theorem is then straightforward.

Theorem 6.1 *(Adequacy – fixed points). Let P be a process and Δ be a set of non-logical axioms. Then, for any constraint c,*

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

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

Verification of (infinite) processes. It is easy to extend the simple unfolding setting presented here by adding the rules for equality and least and/or greatest fixed points. The resulting system would be just an adaptation, for the intuitionistic case and with the exponential !, of the system μMALL presented in [3]. This allows for the verification of properties inside the logical system. We will illustrate this by adding the least fixed point in order to prove some simple properties by induction.

The rules for the least fixed point μ are

$$\dfrac{\Upsilon : \Gamma; \Theta, S\overline{t} \Rightarrow G \quad \cdot : \cdot; S\overline{x} \Rightarrow BS\overline{x}}{\Upsilon : \Gamma; \Theta, \mu B\overline{t} \Rightarrow G}\, \mu_L \qquad \dfrac{\Upsilon : \Gamma; \cdot \rightarrow [B(\mu B)\overline{t}]}{\Upsilon : \Gamma; \cdot \rightarrow [\mu B\overline{t}]}\, \mu_R$$

In the above rules, $\overline{x}$ are fresh variables, $\overline{t}$ are terms and S is a closed formula of the same type as B, called the *invariant*.

Consider a Herbrand constraint system [45], where constraints are equalities on terms. Let 0 (zero) be a constant, `suc` be a function (successor) and consider the following process definitions:

$$\begin{aligned}
\texttt{nat}(x) &\stackrel{\text{def}}{=} \mathbf{ask}\, x = 0\, \mathbf{then}\, \mathbf{tell}(\texttt{true}) + \\
&\quad\ \mathbf{ask}\, \exists x'.x = \texttt{suc}(x')\, \mathbf{then}\, (\mathbf{local}\, x')\, \mathbf{tell}(x = \texttt{suc}(x')) \parallel \texttt{nat}(x') \\
\texttt{plus}(x, y, z) &\stackrel{\text{def}}{=} \mathbf{ask}\, x = 0\, \mathbf{then}\, \mathbf{tell}(y = z) + \\
&\quad\ \mathbf{ask}\, \exists x'.x = \texttt{suc}(x')\, \mathbf{then}\, (\mathbf{local}\, x', z') \\
&\qquad \mathbf{tell}(x = \texttt{suc}(x')) \parallel \texttt{plus}(x', y, z') \parallel \mathbf{tell}(z = \texttt{suc}(z'))
\end{aligned}$$

As expected, by unfolding, we can prove the sequent below:

$$\Upsilon : \cdot \Rightarrow \forall y, z.(\mathcal{L}[\![\texttt{nat}(y)]\!]_\mu \multimap \mathcal{L}[\![\texttt{nat}(z)]\!]_\mu \multimap \mathcal{L}[\![\texttt{plus}(0, y, z)]\!]_\mu \multimap y = z)$$

However, proving the sequent

$$\Upsilon : \cdot \Rightarrow \forall x, z.(\mathcal{L}[\![\texttt{nat}(x)]\!]_\mu \multimap \mathcal{L}[\![\texttt{nat}(z)]\!]_\mu \multimap \mathcal{L}[\![\texttt{plus}(x, 0, z)]\!]_\mu \multimap x = z)$$

requires induction. A simple inspection shows that the invariant S, needed in rule μ_L is

$$\lambda p.\lambda x, y, z.\forall z'.(\texttt{plus}(x, 0, z') \multimap x = z')$$

Note that the right hand side of the sequents above does not correspond to a guard (G) as defined in the previous sections. In fact, it does not correspond to the encoding of any process, since the rule μ_L clearly does not have any operational counterpart: it is used only for verification purposes.

7. Concluding remarks

In this work, we have analyzed different encodings from CCP into intuitionistic linear logic, determining the level of adequacy in each case. We showed that, by using a focusing discipline, we have a complete control of concurrent processes via logic, closing for good the connection between proof theory and CCP calculi.

The encodings proposed here are simpler than the ones presented in [31]. In fact, here we do not make use of subexponentials [12], and we show that focusing is responsible, alone, for the strongest possible level of adequacy. However, it is not possible to deal with other CCP languages such as epistemic and spatial extensions [21] using only focusing, for that, the subexponentials are needed. Also, differently from [31], 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 greater detail how the synchronization of agents can be better controlled, as well as how to deal with procedure calls via fixed points. Our encodings thus open the possibility of using induction for the verification of CCP programs.

Linear CCP (`lcc`) [15] is a CCP language where constraints are build from the fragment $!, \exists, \otimes, 1$ of ILL. Ask agents (interpreted as linear implications) can consume information when querying the store: if the current store is d, the linear ask agent **ask** c **then** P executes P on the store e if $d \vdash_\Delta c[\vec{t}/\vec{x}] \otimes e$ (in `lcc`, the free variables $\vec{x}$ in c are implicitly universally quantified). Note that even without the choice operator, `lcc` is non-deterministic since there may be several constraints that satisfy the condition $d \vdash_\Delta c[\vec{t}/\vec{x}] \otimes e$ (see also discussion on the *most general choice* in [19]).

The results presented here extend straightforwardly to `lcc` when *synchronization constraints*, i.e., linear atomic constraints without non-logical axioms [41], are considered. The problem of handling `lcc` with non-logical axioms in our encoding is the following. Consider an atomic linear constraint c and the formula $F = c \multimap \mathcal{L}[\![P]\!]$ corresponding to the encoding of the linear ask process **ask** c **then** P. If we decide to focus on F, the atom c must be already in the context and the proof must end immediately. This means that the non-logical axioms of the constraint system cannot be used.[3] Of course, one could prove the possibility of applying such axioms before the focusing phase, but this would then break the adequacy results. The simplest way we know for adequately specifying, with the highest level of adequacy, linear constraints systems with non-logical axioms is by using subexponentials to mark processes, constraints, and processes definitions, as done in [31].

Since the constraint system in CCP behaves classically, one may wonder whether the same results in this paper may be achieved by using (focused) intuitionistic logic (LJ). In the case of indeterminate CCP, it is clear that one source of linearity is needed to avoid, due to contraction, the possibility of observing both P and Q in the non-deterministic choice $P + Q$. In the case of the deterministic language, a FCP (full completeness of proofs) result can be stated. However, focusing in LJ is not enough to show a FCD (full completeness of derivations) result. Firstly, in our encodings, the linear context stores the processes (not yet executed) and the classical context stores the constraints already added (i.e., it encodes the CCP store). Such distinction is not possible in LJ. Secondly, in a LJ derivation, the same implication may appear twice as the main formula. This would correspond to execute twice the same ask agent. Finally, it would be impossible to prove that the use of non-logical axioms permutes up in a derivation (see discussion after Lemma 4.1).

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 CCP 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 generate specifications for the system Bedwyr (http://slimmer.gforge.inria.fr/bedwyr/) based on the fixpoint interpretation of procedure calls in Section 6. This may allow to verify systems modeled in CCP. Another research direction would be to consider higher-order processes as those in [42]. This can be handled by fixpoints characterization of procedure calls as done in Section 6.

Finally, it is worth noticing that linear logic has been shown to be an adequate logical framework to reason about other process algebras such as CCS [25] and the π-calculus [27] (see a survey in [8]). Two lines of research can be identified in these developments. On one side, the *processes-as-terms* interpretation [1] identifies operational reductions with cut-elimination as in the Curry–Howard isomorphism. For a more recent track on this direction, the reader may refer to [7] (resp. [48]) where formulas in intuitionistic (resp. classical) linear logic are given a computational interpretation as session types. On the other side, we have the *processes-as-formulas* interpretation [23] where constructs in the language are interpreted as connectives in the logic as we did in our encodings. Adequacy results relate proof steps with operational steps.

Extensions of linear logic have also played an important role in the specification of different concurrent behaviors. For instance, linear logic with subexponentials has been used in the specification of bigraphs [26] (a general model for concur-

[3] Note that this does not happen in the case of CCP since all the constraints behave classically and $\triangledown c = !c$ – see Fig. 3. Hence, the focusing is lost in $!c$ and we can deduce c from the set of constraints and the non-logical axioms in the theory $\triangledown\Delta$ – see discussion after Lemma 4.1.

rency that subsumes both CCS and the π-calculus) in [11], spatial and epistemic behaviors in [31,33,35] and biochemical systems in [32]. We can also mention [22] that uses Hybrid Linear Logic for the specification of biological systems.

Acknowledgements

We thank the anonymous reviewers for their detailed comments that helped us to improve the paper. The work of Olarte and Pimentel have been supported by CAPES, CNPq and by the Marie Curie project GetFun (PIRSES-GA-2012-318986 funded by EU-FP7).

References

[1] Samson Abramsky, Proofs as processes, Theoret. Comput. Sci. 135 (1) (1994) 5–9.
[2] Jean-Marc Andreoli, Logic programming with focusing proofs in linear logic, J. Logic Comput. 2 (3) (1992) 297–347.
[3] David Baelde, Least and greatest fixed points in linear logic, ACM Trans. Comput. Log. 13 (1) (2012) 2.
[4] Gérard Berry, Georges Gonthier, The esterel synchronous programming language: design, semantics, implementation, Sci. Comput. Program. 19 (2) (1992) 87–152.
[5] Taus Brock-Nannestad, Nicolas Guenot, Cut elimination in multifocused linear logic, in: Proceedings Third International Workshop on Linearity, LINEARITY, 2014, Vienna, Austria, 13th July, 2014, 2015, pp. 24–33.
[6] Stephen D. Brookes, C.A.R. Hoare, A.W. Roscoe, A theory of communicating sequential processes, J. ACM 31 (3) (1984) 560–599.
[7] Luís Caires, Frank Pfenning, Bernardo Toninho, Linear logic propositions as session types, Math. Structures Comput. Sci. 26 (3) (2016) 367–423.
[8] Iliano Cervesato, Andre Scedrov, Relating state-based and process-based concurrency through linear logic (full-version), Inform. and Comput. 207 (10) (2009) 1044–1077.
[9] Kaustuv Chaudhuri, Stefan Hetzl, Dale Miller, A systematic approach to canonicity in the classical sequent calculus, in: Patrick Cégielski, Arnaud Durand (Eds.), CSL'12, in: LIPIcs, vol. 16, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 183–197.
[10] Kaustuv Chaudhuri, Dale Miller, Alexis Saurin, Canonical sequent proofs via multi-focusing, in: 5th Int. Conf. in TCS, in: IFIP, vol. 273, 2008, pp. 383–396.
[11] Kaustuv Chaudhuri, Giselle Reis, An adequate compositional encoding of bigraph structure in linear logic with subexponentials, in: Martin Davis, Ansgar Fehnker, Annabelle McIver, Andrei Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning – 20th International Conference, LPAR-20, 2015, Suva, Fiji, 24–28 November, 2015, in: Lecture Notes in Computer Science, vol. 9450, Springer, 2015, pp. 146–161.
[12] Vincent Danos, Jean-Baptiste Joinet, Harold Schellinx, The structure of exponentials: uncovering the dynamics of linear logic proofs, in: Kurt Gödel Colloquium, 1993, pp. 159–171.
[13] Frank S. de Boer, Maurizio Gabbrielli, Elena Marchiori, Catuscia Palamidessi, Proving concurrent constraint programs correct, ACM Trans. Program. Lang. Syst. 19 (5) (1997) 685–725.
[14] Frank S. de Boer, Maurizio Gabbrielli, Maria Chiara Meo, A timed concurrent constraint language, Inform. and Comput. 161 (1) (2000) 45–83.
[15] François Fages, Paul Ruet, Sylvain Soliman, Linear concurrent constraint programming: operational and phase semantics, Inform. and Comput. 165 (1) (2001) 14–41.
[16] Moreno Falaschi, Maurizio Gabbrielli, Kim Marriott, Catuscia Palamidessi, Confluence in concurrent constraint programming, Theoret. Comput. Sci. 183 (2) (1997) 281–315.
[17] Gerhard Gentzen, Investigations into logical deductions, in: M.E. Szabo (Ed.), The Collected Papers of Gerhard Gentzen, North-Holland, Amsterdam, 1969, pp. 68–131.
[18] Jean-Yves Girard, Linear logic, Theoret. Comput. Sci. 50 (1987) 1–102.
[19] Rémy Haemmerlé, Observational equivalences for linear logic concurrent constraint languages, Theory Pract. Log. Program. 11 (4–5) (2011) 469–485.
[20] Radha Jagadeesan, Gopalan Nadathur, Vijay A. Saraswat, Testing concurrent systems: an interpretation of intuitionistic logic, in: Ramaswamy Ramanujam, Sandeep Sen (Eds.), FSTTCS, in: Lecture Notes in Computer Science, vol. 3821, Springer, 2005, pp. 517–528.
[21] Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, Frank D. Valencia, Spatial and epistemic modalities in constraint-based process calculi, in: Maciej Koutny, Irek Ulidowski (Eds.), CONCUR, in: Lecture Notes in Computer Science, vol. 7454, Springer, 2012, pp. 317–332.
[22] Elisabetta De Maria, Joëlle Despeyroux, Amy P. Felty, A logical framework for systems biology, in: François Fages, Carla Piazza (Eds.), Formal Methods in Macro-Biology – First International Conference, FMMB 2014, Nouméa, New Caledonia, September 22–24, 2014, in: Lecture Notes in Computer Science, vol. 8738, Springer, 2014, pp. 136–155.
[23] Dale Miller, The pi-calculus as a theory in linear logic: preliminary results, in: Evelina Lamma, Paola Mello (Eds.), Extensions of Logic Programming, Third International Workshop, Proceedings, ELP'92 Bologna, Italy, 26–28 February, 1992, in: Lecture Notes in Computer Science, vol. 660, Springer, 1992, pp. 242–264.
[24] Dale Miller, Alexis Saurin, From proofs to focused proofs: a modular proof of focalization in linear logic, in: Jacques Duparc, Thomas A. Henzinger (Eds.), Lecture Notes in Computer Science, vol. 4646, Springer, 2007, pp. 405–419.
[25] Robin Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92, Springer, 1980.
[26] Robin Milner, The Space and Motion of Communicating Agents, Cambridge University Press, 2009.
[27] Robin Milner, Joachim Parrow, David Walker, A calculus of mobile processes, Parts I and II, Inform. and Comput. 100 (1) (1992) 1–40.
[28] M. Nielsen, C. Palamidessi, F. Valencia, Temporal concurrent constraint programming: denotation, logic and applications, Nordic J. Comput. 9 (1) (2002) 145–188.
[29] Vivek Nigam, Dale Miller, Algorithmic specifications in linear logic with subexponentials, in: António Porto, Francisco Javier López-Fraguas (Eds.), PPDP, ACM, 2009, pp. 129–140.
[30] Vivek Nigam, Dale Miller, A framework for proof systems, J. Automat. Reason. 45 (2) (2010) 157–188.
[31] Vivek Nigam, Carlos Olarte, Elaine Pimentel, A general proof system for modalities in concurrent constraint programming, in: Pedro R. D'Argenio, Hernán C. Melgratti (Eds.), CONCUR, in: Lecture Notes in Computer Science, vol. 8052, Springer, 2013, pp. 410–424.
[32] Carlos Olarte, Davide Chiarugi, Moreno Falaschi, Diana Hermith, A proof theoretic view of spatial and temporal dependencies in biochemical systems, Theoret. Comput. Sci. 641 (2016) 25–42.
[33] Carlos Olarte, Vivek Nigam, Elaine Pimentel, Dynamic spaces in concurrent constraint programming, Electron. Notes Theor. Comput. Sci. 305 (2014) 103–121.
[34] Carlos Olarte, Elaine Pimentel, Proving concurrent constraint programming correct, revisited, Electron. Notes Theor. Comput. Sci. 312 (2015) 179–195.
[35] Carlos Olarte, Elaine Pimentel, Vivek Nigam, Subexponential concurrent constraint programming, Theoret. Comput. Sci. 606 (2015) 98–120.
[36] Carlos Olarte, Camilo Rueda, Frank D. Valencia, Models and emerging trends of concurrent constraint programming, Constraints 18 (4) (2013) 535–578.
[37] Carlos Olarte, Frank D. Valencia, Universal concurrent constraint programing: symbolic semantics and applications to security, in: Roger L. Wainwright, Hisham Haddad (Eds.), Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, 16–20 March, 2008, ACM, 2008, pp. 145–150.

[38] Prakash Panangaden, Vijay A. Saraswat, Philip J. Scott, R.A.G. Seely, A hyperdoctrinal view of concurrent constraint programming, in: J.W. de Bakker, Willem P. de Roever, Grzegorz Rozenberg (Eds.), REX Workshop, in: Lecture Notes in Computer Science, vol. 666, Springer, 1992, pp. 457–476.
[39] Elaine Pimentel, Dale Miller, On the specification of sequent systems, in: Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Proceedings, Montego Bay, Jamaica, December 2–6, 2005, pp. 352–366.
[40] Elaine Pimentel, Vivek Nigam, João Neto, Multi-focused proofs with different polarity assignments, Electron. Notes Theor. Comput. Sci. 323 (2016) 163–179.
[41] Paul Ruet, François Fages, Concurrent constraint programming and non-commutative logic, in: Mogens Nielsen, Wolfgang Thomas (Eds.), CSL'97, in: Lecture Notes in Computer Science, vol. 1414, Springer, 1997, pp. 406–423.
[42] Vijay Saraswat, Patrick Lincoln, Higher-order linear concurrent constraint programming, Technical report, 1992.
[43] Vijay A. Saraswat, Radha Jagadeesan, Vineet Gupta, Timed default concurrent constraint programming, J. Symbolic Comput. 22 (5/6) (1996) 475–520.
[44] Vijay A. Saraswat, Martin C. Rinard, Concurrent constraint programming, in: Frances E. Allen (Ed.), POPL'90, ACM Press, 1990, pp. 232–245.
[45] Vijay A. Saraswat, Martin C. Rinard, Prakash Panangaden, Semantic foundations of concurrent constraint programming, in: David S. Wise (Ed.), POPL, ACM, 1991, pp. 333–352.
[46] Dana S. Scott, Domains for denotational semantics, in: Mogens Nielsen, Erik Meineche Schmidt (Eds.), ICALP, in: LNCS, vol. 140, Springer, 1982, pp. 577–613.
[47] Gert Smolka, A foundation for higher-order concurrent constraint programming, in: J.-P. Jouannaud (Ed.), Proceedings of Constraints in Computational Logics, in: LNCS, vol. 845, Springer, 1994, pp. 50–72.
[48] Philip Wadler, Propositions as sessions, in: Peter Thiemann, Robby Bruce Findler (Eds.), ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, 9–15 September, 2012, ACM, 2012, pp. 273–286.