# Process-as-formula interpretation: A substructural multimodal view

**Elaine Pimentel**[1]
Department of Mathematics, UFRN, Brazil

**Carlos Olarte**
School of Science and Technology, UFRN, Brazil

**Vivek Nigam**
Huawei Munich Research Center, Germany

## Abstract

In this survey, we show how the processes-as-formulas interpretation, where computations and proof-search are strongly connected, can be used to specify different concurrent behaviors as logical theories. The proposed interpretation is parametric and modular, and it faithfully captures behaviors such as: Linear and spatial computations, epistemic state of agents, and preferences in concurrent systems. The key for this modularity is the incorporation of multimodalities in a resource aware logic, together with the ability of quantifying on such modalities. We achieve tight adequacy theorems by relying on a focusing discipline that allows for controlling the proof search process.

**2012 ACM Subject Classification** Theory of computation → Proof theory; Theory of computation → Process calculi

**Keywords and phrases** Linear logic, proof theory, process calculi

**Digital Object Identifier** 10.4230/LIPIcs.FSCD.2021.3

**Category** Invited Talk

**Funding** *Elaine Pimentel*: Partially funded by the project MOSAIC (MSCA RISE 101007627).
*Carlos Olarte*: Partially funded by CNPq.

## 1 Introduction

Computational logic research has produced deep and fruitful cross-fertilizations between programming languages and proof theory. Arguably, the most well-known one is the *Curry-Howard correspondence* (also known as types-as-formulas) where (functional) programs correspond to formal proofs and their execution to cut-elimination. A second type of correspondence, *processes-as-formulas* (also known as computation-as-proof-search), was initiated by Miller [21] where, instead, (logic) programs correspond to formulas and their execution to proof search. These two foundational correspondences have been exploited to propose new programming language paradigms as well as greatly extend the expressiveness of existing ones.

When processes or programs are specified as formulas, one has to be careful with the level of adequacy obtained. In particular, it is expected that logical steps in derivations correspond to steps of computations in programs. However, different from computational systems, where *one step of computation* is rigidly determined by the operation semantics, *one step of logical reasoning* depends strongly on the logical framework chosen. Also, the logic should capture, in a natural way, the behavior of programs. For instance, intuitionistic logic (IL) is not adequate to specify systems that may consume information (substructural behavior), execute

[1] Corresponding author.

© Elaine Pimentel, Carlos Olarte and Vivek Nigam;
licensed under Creative Commons License CC-BY 4.0

processes in different locations (spatial modalities) or time instances (timed reasoning), or when the information shared by processes is subject to quantitative information (such as preferences or costs).

Hence the need for a more expressive logic (such as *multimodal and resource aware logics*) and an appropriate notion of normal proofs as the logical counterpart of the processes-as-formulas correspondence. This paper surveys one of such choices: focused linear logic with subexponentials (SELLF) [28]. We present different mechanisms previously explored by the authors to both: extend SELLF with quantification over subexponentials; and give adequate characterizations of existing concurrent languages. This fruitful collaboration between the two areas has been useful to provide reasoning techniques for process calculi with the motto *reachability as entailment*, and also to propose declarative extensions of concurrent languages with solid logical grounds.

The *focusing discipline* [1] determines an alternating mechanism on proofs (between *focused* and *unfocused* phases), which controls the non-determinism during proof search, producing normal form proofs. Such normalization of proofs leads to a practical approach to identify logical steps: a *focused step* is a block determined by a focused phase followed by an unfocused one, in a (bottom-up) focused proof. In Section 2 we recall the proof theory of focused intuitionistic linear logic (ILLF), which will be the base logical language for the processes-as-formulas correspondences addressed in this paper. Section 3 then introduces the base computational counterpart of the correspondence, Concurrent Constraint Programming (CCP) [42], a declarative model for concurrency. We show how to adequately capture the behavior of CCP processes in ILLF.

The *level of adequacy* attained in such interpretations will be important in order to justify the choice of the underlying logic: the closer the two systems are, the easier is to prove the correspondence. Also, a strong adequacy allows for the use of the logical system for proving properties of the computational system, or reconstructing counter-examples from failing derivations. Following [29], we classify the level of adequacy into two classes:

- **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 should correspond to one step of logical reasoning.

In the first case, even though the outputs of a program are characterized by proofs in the underlying logic, it may be the case that there are steps in the logical reasoning that do not correspond to computational steps and vice-versa. In the second case, computational and (in our case, focused) logical steps are in one-to-one correspondence. We present a careful discussion about these different levels of adequacy regarding CCP and ILLF in Section 3.2, and indicate throughout the text, in each result, its level of adequacy.

Even though (focused, intuitionistic) linear logic is suitable for the encoding of (vanilla) CCP, the situation changes when *modalities* are added to concurrent systems: For that, linear logic *subexponentials* are needed. In Section 4 we present SELLF, which shares with ILL all its connectives except the exponential: instead of having a single !, it may contain as many subexponentials as one needs (written $!^a$). Such labels are organized in a pre-order, and different organizations give rise to different CCP flavors. Section 5 is then devoted to show how to add such structures *parametrically* to SELLF, obtaining strongly adequate specifications. In this way, processes may be executed and add/query constraints in different *locations*, where the meaning of such locations may vary, for example: Spaces of computation, the epistemic state of agents, time units, levels of preferences, etc. *Modularity* is guaranteed by the fact that the underline interpretation is the same: Locations in CCP become labels in

SELLF. Finally, Section 6 concludes the paper.

## 2 Focused intuitionistic linear logic

Linear logic (LL) is a substructural logic proposed by Girard [13] as a refinement of classical and intuitionistic logics, joining the dualities of the former with many of the constructive properties of the latter.

In this paper, we will concentrate in the *intuitionistic* version of linear logic (ILL) [13], with formulas built from the following grammar

$$F, G ::= A \mid 1 \mid 0 \mid \top \mid F \otimes G \mid F \,\&\, G \mid F \oplus G \mid F \multimap G \mid !\,F \mid \forall x.F \mid \exists x.F$$

Here, $A$ denotes an atomic formula; $\multimap$, $\otimes$, $1$ represent the *multiplicative* implication, conjunction and true, respectively; $\&$, $\top$, $\oplus$, $0$ are the *additive* conjunction, true, disjunction, and false, respectively; $!$ is the *exponential*; and $\exists, \forall$ represent the existential and universal quantifiers, respectively.[2]

These connectives can be separated into two classes, the *negative*: $\multimap, \&, \top, \forall$ and the *positive*: $\otimes, \oplus, !, 1, 0, \exists$. The polarity of non-atomic formulas is inherited from its outermost connective (e.g., $F \otimes G$ is a positive formula) and any bias can be assigned to atomic formulas.[3] This partition induces an alternating mechanism on proofs, known as *focusing*, which aims at reducing the non-determinism during proof search. In this sense, focused proofs can be interpreted as *normal form* proofs.

The focusing discipline [1] is determined by the alternation of *focused* and *unfocused* phases in the proof construction. In the unfocused phase, inference rules can be applied eagerly and no backtracking is necessary; in the focused phase, on the other hand, either context restrictions apply, or choices within inference rules can lead to failures for which one may need to backtrack. These phases are totally determined by the polarities of formulas: provability is preserved when applying right/left rules for negative/positive formulas respectively, but not necessarily in other cases.

The focused intuitionistic linear logic system (ILLF) is depicted in Figure 1.

There are three contexts on the left side of ILLF sequents: the set $\Theta$ denotes the *unbounded* context, containing only formulas with a banged scope; $\Gamma$ is a *linear* context containing only negative or atomic formulas; and $\Delta$ is the general linear context. Observe that formulas in the context $\Theta$ behave as in classical logic: they can be weakened (erased) or contracted (duplicated). Formulas in the other contexts are linear, and are consumed when used.

The phase distinction is reflected in the design of sequents in ILLF: the presence of "$\Uparrow$" indicates unfocused sequents, while "$\Downarrow$" marks the formula under focus in focused sequents. Sequents in ILLF have one of the following shapes:

i. $\Theta; \Gamma \Uparrow \Delta \vdash F \Uparrow$ is an unfocused sequent.
ii. $\Theta; \Gamma \Uparrow \cdot \vdash \cdot \Uparrow F$ is an unfocused sequent representing the end of an unfocused phase.
iii. $\Theta; \Gamma \vdash F \Downarrow$ is a sequent focused on the right.
iv. $\Theta; \Gamma \Downarrow F \vdash R$ is a sequent focused on the left.

The swing between focused and unfocused phases is described below.

[2] Observe that the multiplicative false $\bot$ could be added to ILL's syntax. However, this would break the nice feature of having *exactly* one formula on succedent of sequents.

[3] Although the bias assigned to atoms does not interfere with provability, it changes *considerably* the shape of proofs (see, *e.g.*, [19]).

Unfocused introduction rules

$$\frac{\Theta;\Gamma \Uparrow F, \Delta \vdash G \Uparrow}{\Theta;\Gamma \Uparrow \Delta \vdash F \multimap G \Uparrow} \multimap_r \qquad \frac{\Theta;\Gamma \Uparrow F, G, \Delta \vdash \mathcal{R}}{\Theta;\Gamma \Uparrow F \otimes G, \Delta \vdash \mathcal{R}} \otimes_l \qquad \frac{F, \Theta;\Gamma \Uparrow \Delta \vdash \mathcal{R}}{\Theta;\Gamma \Uparrow !F, \Delta \vdash \mathcal{R}} !_l$$

$$\frac{\Theta;\Gamma \Uparrow \Delta \vdash F \Uparrow \quad \Theta;\Gamma \Uparrow \Delta \vdash G \Uparrow}{\Theta;\Gamma \Uparrow \Delta \vdash F \,\&\, G \Uparrow} \,\&_r \qquad \frac{\Theta;\Gamma \Uparrow F, \Delta \vdash \mathcal{R} \quad \Theta;\Gamma \Uparrow G, \Delta \vdash \mathcal{R}}{\Theta;\Gamma \Uparrow F \oplus G, \Delta \vdash \mathcal{R}} \oplus_l$$

$$\frac{\Theta;\Gamma \Uparrow \Delta \vdash F[y/x] \Uparrow}{\Theta;\Gamma \Uparrow \Delta \vdash \forall x.F \Uparrow} \forall_r \qquad \frac{\Theta;\Gamma \Uparrow F[y/x], \Delta \vdash \mathcal{R}}{\Theta;\Gamma \Uparrow \exists x.F, \Delta \vdash \mathcal{R}} \exists_l$$

$$\frac{}{\Theta;\Gamma \Uparrow \Delta \vdash \top \Uparrow} \top_r \qquad \frac{\Theta;\Gamma \Uparrow \Delta \vdash \mathcal{R}}{\Theta;\Gamma \Uparrow 1, \Delta \vdash \mathcal{R}} 1_l \qquad \frac{}{\Theta;\Gamma \Uparrow 0, \Delta \vdash \mathcal{R}} 0_l$$

Focused introduction rules

$$\frac{\Theta;\Gamma_1 \vdash F \Downarrow \quad \Theta;\Gamma_2 \Downarrow G \vdash R}{\Theta;\Gamma_1, \Gamma_2 \Downarrow F \multimap G \vdash R} \multimap_l \qquad \frac{\Theta;\Gamma \vdash F_i \Downarrow}{\Theta;\Gamma \vdash F_1 \oplus F_2 \Downarrow} \oplus_{r_i} \qquad \frac{\Theta;\Gamma \Downarrow F_i \vdash R}{\Theta;\Gamma \Downarrow F_1 \,\&\, F_2 \vdash R} \,\&_{l_i}$$

$$\frac{\Theta;\Gamma_1 \vdash F \Downarrow \quad \Theta;\Gamma_2 \vdash G \Downarrow}{\Theta;\Gamma_1, \Gamma_2 \vdash F \otimes G \Downarrow} \otimes_r \qquad \frac{\Theta;\cdot \Uparrow \cdot \vdash F \Uparrow}{\Theta;\cdot \vdash !F \Downarrow} !_r$$

$$\frac{\Theta;\Gamma \Downarrow F[t/x] \vdash R}{\Theta;\Gamma \Downarrow \forall x.F \vdash R} \forall_l \qquad \frac{\Theta;\Gamma \vdash F[t/x] \Downarrow}{\Theta;\Gamma \vdash \exists x.F \Downarrow} \exists_r \qquad \frac{}{\Theta;\cdot \vdash 1 \Downarrow} 1_r$$

Structural and identity rules

$$\frac{\Theta;\Gamma \Downarrow N \vdash R}{\Theta;\Gamma, N \Uparrow \cdot \vdash \cdot \Uparrow R} \mathsf{D}_l \qquad \frac{\Theta, F;\Gamma \Downarrow F \vdash R}{\Theta, F;\Gamma \Uparrow \cdot \vdash \cdot \Uparrow R} \mathsf{Du} \qquad \frac{\Theta;\Gamma \vdash P \Downarrow}{\Theta;\Gamma \Uparrow \cdot \vdash \cdot \Uparrow P} \mathsf{D}_r$$

$$\frac{\Theta;\Gamma \Uparrow P \vdash \cdot \Uparrow R}{\Theta;\Gamma \Downarrow P \vdash R} \mathsf{R}_l \qquad \frac{\Theta;\Gamma \Uparrow \cdot \vdash N \Uparrow}{\Theta;\Gamma \vdash N \Downarrow} \mathsf{R}_r \qquad \frac{\Theta;C, \Gamma \Uparrow \Delta \vdash \mathcal{R}}{\Theta;\Gamma \Uparrow C, \Delta \vdash \mathcal{R}} \mathsf{S}_l \qquad \frac{\Theta;\Gamma \Uparrow \cdot \vdash \cdot \Uparrow D}{\Theta;\Gamma \Uparrow \cdot \vdash D \Uparrow} \mathsf{S}_r$$

$$\frac{}{\Theta;A \vdash A \Downarrow} \mathsf{I} \qquad \frac{}{\Theta, A;\cdot \vdash A \Downarrow} \mathsf{I}_c$$

Here, $P$ is positive, $N$ is negative, $C$ is a negative formula or positive atom, $D$ a positive formula or negative atom, and $A$ is a positive atom. Other formulas are arbitrary. $\mathcal{R}$ denotes $\Delta_1 \Uparrow \Delta_2$ where the union of $\Delta_1$ and $\Delta_2$ contains exactly one formula. In the rules $\forall_r$ and $\exists_l$ the eigenvariable $y$ does not occur free in any formula of the conclusion.

**Figure 1** The focused intuitionistic linear sequent calculus ILLF.

- At the beginning of an unfocused phase, sequents have the shape (i) and: non-atomic negative formulas appearing in the right context, and positive non-atomic formulas appearing in $\Delta$ are eagerly introduced; atomic/negative left formulas are stored in $\Gamma$ using the store rule $\mathsf{S}_l$; atomic/positive right formulas are stored in the outermost right context using the store rule $\mathsf{S}_r$.
  When this phase ends, sequents have the form (ii).
- The focused phase begins by choosing, via one of the decide rules $\mathsf{D}_l, \mathsf{Du}$ or $\mathsf{D}_r$, a formula to be focused on, 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; or a negative formula on the right or a positive formula on the left is derived. At this point, focusing will be lost, and the proof switches to the unfocused phase again.

We will call a *focused step* a focused phase followed by an unfocused one, in a (bottom-up) focused proof.

Observe that the design of the axioms $\mathsf{I}$ and $\mathsf{I}_c$ in ILLF induces a *positive* polarity to atoms. As it will become clear in Section 3.2, this is necessary for guaranteeing the higher level of adequacy on encodings.

Sequents in ILL will be denoted by $\Gamma \vdash A$. Rules for ILL are the same as in ILLF, only not considering focusing, and the structural rules being substituted by the usual bang left rules: dereliction (D), weakening (W) and contraction (C):

$$\frac{\Gamma, F \vdash G}{\Gamma, !F \vdash G}\ \mathsf{D} \qquad \frac{\Gamma \vdash G}{\Gamma, !F \vdash G}\ \mathsf{W} \qquad \frac{\Gamma, !F, !F \vdash G}{\Gamma, !F \vdash G}\ \mathsf{C}$$

Note that, in ILLF, dereliction is embedded into the bang left ($!_l$) and unbounded decide (Du) rules.

## 3 Concurrent Constraint Processes as LL Formulas

In this section we shall see how the process-as-formula interpretation can be used for both, providing verification techniques for a process calculus and characterizing different semantics for it in a uniform way. We start by describing the model of computation of *Concurrent Constraint Programming* (CCP) to later show that ILLF provides a suitable framework for interpreting CCP processes.

Concurrent Constraint Programming (CCP) [41, 42, 43, 37] is a model for concurrency based upon the shared-variables communication model. CCP traces its origins back to the ideas of *computing with constraints* [25], *Concurrent Logic Programming* [45] and *Constraint Logic Programming* (CLP) [15]. Different from other models for concurrency, based on point-to-point communication as in CCS [23], the $\pi$-calculus [24], CSP [14] among several others, the CCP model focuses on the concept of *partial information*, traditionally referred to as *constraints*. Under this paradigm, the conception of *store as valuation* in the von Neumann model is replaced by the notion of *store as constraint*, and processes are seen as *information transducers*.

The model of concurrency in CCP is quite simple: concurrent agents (or processes) interact with each other and their environment by posting and asking information (i.e., constraints) in a medium, a so-called *store*. As we shall see, CCP processes can be seen as both computing processes (behavioral style) and as formulas in logic (logical declarative style). In particular, we shall see a strong connection between ILL and CCP originally developed in [11] and later refined in [34].

## 3.1 Constraint system and processes

We start by defining the language of processes and constraints. The type of constraints processes may act on is not fixed but parametric in a constraint system. Such systems can be formalized as Scott information systems [44] as in [40], or they can be built upon a suitable fragment of logic *e.g.* as in [46, 11, 26]. Here we shall follow the second approach. More precisely, a constraint system is a tuple $\mathbf{C} = (\mathcal{C}, \models_\Delta)$ where the set of constraints $\mathcal{C}$ is 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 in $\mathcal{C}$. The entailment relation $\models_\Delta$ is parametric on a set of non-logical axioms $\Delta$ 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 $d \models_\Delta c$, iff the sequent $\Delta, d \vdash c$ is provable in intuitionistic logic (IL). Intuitively, the entailment relation specifies inter-dependencies between constraints: $c \models_\Delta d$ means that the information $d$ can be deduced from the information represented by $c$, *e.g.* $x > 42 \models_\Delta x > 0$.

The constraint store, shared by processes, is a conjunction of constraints and `true` denotes the empty store. The existential quantifier is used to specify variable hiding.

Processes are built from constraint as follows:

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

A process $\mathbf{tell}(c)$ adds the constraint $c$ to the store, thus incrementing the information in it. The guarded choice $\sum_{i \in I} \mathbf{ask}\ c_i\ \mathbf{then}\ P_i$, where $I$ is a finite set of indexes, chooses non-deterministically one of the processes $P_j$ whose guard $c_j$ can be deduced from the current store. If none of the guards can be deduced, this process remains blocked until more information is added. Hence, ask agents implement a synchronization mechanism based on entailment of constraints. The interleaved parallel composition of $P$ and $Q$ is denoted as $P \parallel Q$. The agent $(\mathbf{local}\, x)\, P$ behaves as $P$ and binds the variable $x$ to be local to it. Finally, given a possibly recursive process definition $p(\overline{y}) \stackrel{\Delta}{=} P$, where all free variables of $P$ are in the set of pairwise distinct variables $\overline{y}$, the process $p(\overline{x})$ evolves into $P[\overline{x}/\overline{y}]$.

The operational semantics of CCP is given by the transition relation $\gamma \longrightarrow \gamma'$ satisfying the rules in Figure 2. Here we follow the semantics in [11] and a *configuration* $\gamma$ is a triple of the form $(X; \Gamma; c)$, where $c$ is a constraint specifying the store, $\Gamma$ is a multiset of processes, and $X$ is the set of hidden (local) variables of $c$ and $\Gamma$. The multiset $\Gamma = P_1, P_2, \ldots, P_n$ represents the process $P_1 \parallel P_2 ... \parallel P_n$. 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, $\Gamma = \{P_1, ..., P_n\} \cong \{P'_1, ..., P'_n\} = \Gamma'$ iff $P_i \cong P'_i$ for all $1 \leq i \leq n$. Finally, $(X; \Gamma; c) \cong (X'; \Gamma'; c')$ iff $X = X'$, $\Gamma \cong \Gamma'$ and $c \equiv_\Delta c'$ (*i.e.*, $c \models_\Delta c'$ and $c' \models_\Delta c$).

Rules $\mathrm{R_T}$ and $\mathrm{R_C}$ are self-explanatory. Rule $\mathrm{R_{EQUIV}}$ says that structurally congruent processes have the same transitions. Rule $\mathrm{R_L}$ adds the variable $x$ to the set of variables $X$ when it is fresh (otherwise, Rule $\mathrm{R_{EQUIV}}$ can be used to apply alpha conversion). The rule $\mathrm{R_A}$ says that the process $\sum_{i \in I} \mathbf{ask}\ c_i\ \mathbf{then}\ P_i$ evolves into $P_j$ if the current store $d$ entails $c_j$.

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

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

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

**Figure 2** Operational semantics of CCP. In $\mathrm{R}_{\mathrm{L}}$, $x \notin X$ and it does not occur free in $\Gamma$ nor in $d$.

▶ **Definition 1** (Observables). *Let $\longrightarrow^*$ be the reflexive and transitive closure of $\longrightarrow$. If $(X;\Gamma;d) \longrightarrow^* (X';\Gamma';d')$ and $\exists X'.d' \models_\Delta c$ 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 `true`.

## 3.2 Interpretation and adequacy

We shall present different encodings for processes ($\mathcal{P}[\![\cdot]\!]$) and constraints ($\mathcal{C}[\![\cdot]\!]$) as formulas in ILL. Our goal is to show that the outputs of a process $P$ can be characterized by proofs in ILLF. More precisely, we shall show that $P$ outputs $c$ iff a sequent of the form $\mathcal{P}[\![\Psi]\!], \mathcal{C}[\![\Delta]\!] \ : \ \cdot \Uparrow \mathcal{P}[\![P]\!] \vdash \mathcal{C}[\![c]\!] \otimes \top \Uparrow$ is provable in ILLF, where $\Psi$ is a set of process definitions and $\Delta$ is the set of non-logical axioms in the constraint system. Note the use of $\top$: we shall erase the formulas corresponding to processes that were not executed. Below, we will see how to tune the process interpretation to get the highest level of adequacy possible.

▶ **Definition 2.** *Constraints and axioms in CCP are encoded in* ILL *as follows:*

$$\begin{array}{lcllcllcl} \mathcal{C}[\![\mathtt{true}]\!] & = & 1 & \mathcal{C}[\![A]\!] & = & !\,A & \mathcal{C}[\![F_1 \wedge F_2]\!] & = & \mathcal{C}[\![F_1]\!] \otimes \mathcal{C}[\![F_2]\!] \\ \mathcal{C}[\![\exists x.F]\!] & = & \exists x.\mathcal{C}[\![F]\!] & \mathcal{C}[\![\forall \overline{x}.(c \supset d)]\!] & = & \forall \overline{x}.(\mathcal{C}[\![c]\!] \multimap \mathcal{C}[\![d]\!]) & & & \end{array}$$

*For the processes and process definition, the interpretation is the following:*

$$\begin{array}{lcllcl} \mathcal{P}[\![\mathbf{tell}(c)]\!] & = & \mathcal{C}[\![c]\!] & \mathcal{P}[\![P \parallel Q]\!] & = & \mathcal{P}[\![P]\!] \otimes \mathcal{P}[\![Q]\!] \\ \mathcal{P}[\![\sum_{i\in I} \mathbf{ask}\ c_i\ \mathbf{then}\ P_i]\!] & = & \mathop{\&}_{i\in I}(\mathcal{C}[\![c_i]\!] \multimap \mathcal{P}[\![P_i]\!]) & \mathcal{P}[\![(\mathbf{local}\, x)\, P]\!] & = & \exists x.\mathcal{P}[\![P]\!] \\ \mathcal{P}[\![p(\overline{y})]\!] & = & p(\overline{y}) & \mathcal{P}[\![p(\overline{x}) \triangleq P]\!] & = & \forall \overline{x}.(p(\overline{x}) \multimap \mathcal{P}[\![P]\!]) \end{array}$$

Since the store in CCP is monotonic, i.e., constraints cannot be removed, we mark atomic formulas with a bang (to be stored in the unbounded context). Parallel composition is identified with multiplicative conjunction and the act of choosing one of the branches in a non-deterministic choice is specified with additive conjunction. The action of querying the store in ask agents is specified with a linear implication. Similarly, the unfolding of a process definition is guarded by the atomic proposition $p(\vec{y})$ (denoting the call).

If $\Gamma$ is a set of constraints, or axioms of the form $\forall \overline{x}.[c \supset c']$, we write $\mathcal{C}[\![\Gamma]\!]$ to denote the set $\{\mathcal{C}[\![d]\!] \mid d \in \Gamma\}$. A similar convention applies for $\mathcal{P}[\![\cdot]\!]$. Moreover, $!\Gamma = \{!F \mid F \in \Gamma\}$.

▶ **Theorem 3** (Adequacy – ILL [11]). *Let $(\mathcal{C}, \models_\Delta)$ be a constraint system, $P$ be a process and $\Psi$ be a set of process definitions. Then, for any constraint $c$, $P \Downarrow_c$ iff there is a proof of the sequent $!\,\mathcal{P}[\![\Psi]\!], !\,\mathcal{C}[\![\Delta]\!], \mathcal{P}[\![P]\!] \vdash \mathcal{C}[\![c]\!] \otimes \top$ in* ILL. *The level of adequacy is* **FCP**.

Without focusing (as originally done in [11]), the proof of this theorem is not straightforward and a low level of adequacy is obtained: there may be logical steps not corresponding

to any operational step and vice-versa. Let us focus first in the case where logical steps do not correspond to the operational ones. We will come back to the other direction later.

Consider the two derivations bellow.

$$\frac{\overset{\pi_1}{\Gamma, c_1 \multimap F_1 \vdash d}}{\Gamma, (c_1 \multimap F_1) \,\&\, (c_2 \multimap F_2) \vdash d}\ \&_l \qquad\qquad \frac{\overset{\pi_2}{\Gamma_1, F_1 \vdash d} \quad \overset{\pi_3}{\Gamma_2 \vdash c_1}}{\Gamma_1, \Gamma_2, c_1 \multimap F_1 \vdash d}\ \multimap_l \tag{1}$$

In the first, one of the branches is chosen but, in $\pi_1$, it could be the case that $c_1$ is never proved (and $F_1$ is never added to the context). This is not the intended meaning in Rule $\mathrm{R_A}$, that first checks the entailment of $c_j$ to *immediately* add the corresponding process $P_j$ to the context. In the second example, $\pi_3$ could contain sub-derivations that have nothing to do with the proof of the guard $c_1$. For instance, process 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 $\Delta$ and the current store. For instance, it may be the case, in $\pi_3$, that $c_1$ will be later produced by a process $Q$ such that $\mathcal{P}[\![Q]\!] \in \Gamma_2$. This is clearly not allowed by the operational semantics.

Let's now put focusing into play. An inspection in the encoding reveals that the fragment of ILL used is restricted to the following grammar:

$$\begin{array}{lcll} G & := & 1 \mid !\,A \mid G \otimes G \mid \exists x.G & \textit{Guards and Goals} \\ P & := & G \mid P \otimes P \mid P \,\&\, P \mid G \multimap P \mid \exists x.P \mid p(\overline{t}) & \textit{Processes} \\ PD & := & \forall \overline{x}.p(\overline{x}) \multimap P. & \textit{Process Definitions} \end{array}$$

where $A$ is an atomic formula (constraint) in $\mathcal{C}$ and $p$ (a process identifier) is also atomic but $p \notin \mathcal{C}$. In any derivation, the only formulas that can appear on the right are guards/goals $G$ and heads $p$. The other formulas, including processes, process definitions and axioms, appear on the left. Hence, only instances of the unfocused rules $1_l, \otimes_l, \exists_l, !_l, \top_r$ and the focused rules $\otimes_r, \multimap_l, \exists_r, !_r, \&_l, \forall_l$ are used.

Observe that formulas $G, p$ are *strictly positive*. Thus, focusing on such a formula on the right either forces finishing the proof, or the formula will be *entirely* decomposed into formulas of the shape 1 or $!\,A$. This means that a proof of $A$ can use only the theory $\Delta$, the encoding of constraints and process definitions (since all of them are unbounded). In fact, we can show that the encoding of process definitions can be weakened (since calls of the form $p(\vec{y})$ are necessarily stored in the linear context). Hence, when a goal is focused on, it must be completely decomposed, and the atomic constraints must be proved only from the current store and the non-logical axioms.

Formulas occurring on the left of sequents can be positive or negative. *Positive formulas* on the left (that cannot be focused on) come from the interpretation of *tell*, *parallel composition* and *locality* that do not need any interaction with the context. Note, for instance, that the formula $\exists x.!\,G_1 \otimes !\,G_2$, resulting from the encoding of $\mathbf{tell}(\exists x.G_1 \wedge G_2)$, can be entirely decomposed in an unfocused phase using the rules $\otimes_l$, $\exists_l$ and $!_l$. On the other hand, *negative formulas* on the left (that can be chosen for focusing) come from the encodings of *guarded choices* and *process definitions*. 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).

Due to completeness of focusing [1], Theorem 3 trivially holds if we replace in it ILL with ILLF. But using directly the focused system, the proof of the theorem becomes simpler. For instance, it is a routine exercise to show that non-logical axioms permute up, and it is always possible to apply them at the top of proofs. Moreover, situations as the ones described

after the derivations in Equation (1) are not longer valid in the focused system: focusing over $c_1 \multimap F_1$ implies immediately proving $c_1$ (from the logical axioms and accumulated constraints), thus reflecting *exactly* the operational semantics of CCP.

▶ **Example 4.** Consider a *community coffee machine*, which is triggered by the insertion of a coin, always available at the side of the machine. When the user inserts the coin, the machine delivers a coffee and returns the coin, which will be available for the next user. This machine can be specified as the CCP process

$$P = \mathbf{tell}(\mathit{coin}) \parallel \mathtt{m()} \text{ where } \mathtt{m()} \triangleq \mathbf{ask}\ \mathit{coin}\ \mathbf{then}\ (\mathbf{tell}(\mathit{coffee}) \parallel \mathtt{m()})$$

Hence, $P \Downarrow_c$, where $c = \mathit{coin} \wedge \mathit{coffee}$:

$$\langle \emptyset, P, \mathtt{true}\rangle \longrightarrow \langle \emptyset, \mathtt{m()}, \mathit{coin}\rangle \longrightarrow \langle \emptyset, \mathbf{tell}(\mathit{coffee}) \parallel \mathtt{m()}, \mathit{coin}\rangle \longrightarrow \langle \emptyset, \mathtt{m()}, \mathit{coin} \wedge \mathit{coffee}\rangle$$

On the other hand, the sequent $\mathcal{P}[\![P]\!] \vdash \mathcal{C}[\![c]\!] \otimes \top$ has the following focused proof

$$\dfrac{\dfrac{\dfrac{}{\mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \cdot \vdash\ !\,\mathit{coin} \Downarrow}\ !_r, \mathsf{S}_r, \mathsf{D}_r, \mathsf{I} \quad \dfrac{\dfrac{\dfrac{}{\mathit{coffee}, \mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \cdot \vdash\ !\,\mathit{coffee} \Downarrow}\ !_r, \mathsf{S}_r, \mathsf{D}_r, \mathsf{I} \quad \dfrac{}{\mathit{coffee}, \mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \cdot \vdash\ !\,\mathit{coin} \Downarrow}\ !_r, \mathsf{S}_r, \mathsf{D}_r, \mathsf{I}}{\mathit{coffee}, \mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \mathtt{m()} \vdash (!\,\mathit{coffee} \otimes\ !\,\mathit{coin}) \otimes \top \Downarrow} \otimes_r, \top_r}{\mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \cdot \Downarrow\ !\,\mathit{coffee} \otimes \mathtt{m()} \vdash (!\,\mathit{coffee} \otimes\ !\,\mathit{coin}) \otimes \top}\ \mathsf{R}_l, \otimes_l, !_l, \mathsf{D}_r}{\mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \mathtt{m()} \Downarrow \mathtt{m()} \multimap (!\,\mathit{coin} \multimap (!\,\mathit{coffee} \otimes \mathtt{m()})) \vdash (!\,\mathit{coffee} \otimes\ !\,\mathit{coin}) \otimes \top}\ \multimap_l, \mathsf{I}}{\mathit{coin}, \mathcal{P}[\![\mathtt{m()}]\!]; \mathtt{m()} \Uparrow \cdot \vdash \cdot \Uparrow (!\,\mathit{coffee} \otimes\ !\,\mathit{coin}) \otimes \top}\ \mathsf{D}_l}{\mathcal{P}[\![\mathtt{m()}]\!]; \cdot \Uparrow\ !\,\mathit{coin} \otimes \mathtt{m()} \vdash \cdot \Uparrow (!\,\mathit{coffee} \otimes\ !\,\mathit{coin}) \otimes \top}\ \otimes_l, !_l, \mathsf{S}_l$$

Bottom up, we introduce the tell process in the unfocused phase. Then, after focusing on the encoding of the ask agents, the guard *coin* is deduced (left-most derivation), and the token *coin* is stored into the classical context, thus reflecting the final configuration in the execution of the process.

Unfortunately, even with focusing, the adequacy level continues to be **FCP**. In fact, the focusing discipline causes that some CCP computations do not have a corresponding proof in ILLF. To see that, consider the following process

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

We denote the two external ask agents in $P$ as $Q_1$ and $Q_2$ 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 by executing $\mathbf{tell}(a \wedge b)$:

$$\begin{array}{lrcl}
\mathtt{Trace\ 1:} & \langle \emptyset, P, \mathtt{true}\rangle & \longrightarrow & \langle \emptyset, Q_1 \parallel Q_2, a \wedge b\rangle \longrightarrow \langle \emptyset, \mathbf{ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel Q_2, a \wedge b\rangle \\
& & \longrightarrow & \langle \emptyset, \mathbf{tell}(\mathtt{ok}) \parallel Q_2, a \wedge b\rangle \longrightarrow \langle \emptyset, Q_2, a \wedge b \wedge \mathtt{ok}\rangle \longrightarrow^* \langle \emptyset, \cdot, d\rangle \not\longrightarrow \\
\mathtt{Trace\ 2:} & \langle \emptyset, P, \mathtt{true}\rangle & \longrightarrow & \langle \emptyset, Q_1 \parallel Q_2, a \wedge b\rangle \longrightarrow \langle \emptyset, Q_1 \parallel \mathbf{ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}'), a \wedge b\rangle \\
& & \longrightarrow & \langle \emptyset, Q_1 \parallel \mathbf{tell}(\mathtt{ok}'), a \wedge b\rangle \longrightarrow \langle \emptyset, Q_1, a \wedge b \wedge \mathtt{ok}'\rangle \longrightarrow^* \langle \emptyset, \cdot, d\rangle \not\longrightarrow \\
\mathtt{Trace\ 3:} & \langle \emptyset; P; \mathtt{true}\rangle & \longrightarrow & \langle \emptyset; Q_1 \parallel Q_2; a \wedge b\rangle \longrightarrow \langle \emptyset; \mathbf{ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel Q_2; a \wedge b\rangle \\
& & \longrightarrow & \langle \emptyset, \mathbf{ask}\ b\ \mathbf{then\ tell}(\mathtt{ok}) \parallel \mathbf{ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}'), a \wedge b\rangle \\
& & \longrightarrow & \langle \emptyset, \mathbf{tell}(\mathtt{ok}) \parallel \mathbf{ask}\ a\ \mathbf{then\ tell}(\mathtt{ok}'), a \wedge b\rangle \longrightarrow \langle \emptyset, \mathbf{tell}(\mathtt{ok}) \parallel \mathbf{tell}(\mathtt{ok}'), a \wedge b\rangle \\
& & \longrightarrow^* & \langle \emptyset, \cdot, d\rangle
\end{array}$$

`Trace 1` and `Trace 2` correspond exactly to a different focused proof of the sequent $\mathcal{P}[\![P]\!] \vdash \mathcal{C}[\![d]\!]$: one focusing first on $\mathcal{P}[\![Q_1]\!]$ and the other focusing first on $\mathcal{P}[\![Q_2]\!]$. On the other hand, `Trace 3` corresponds to an *interleaved* execution of $Q_1$ and $Q_2$. We note that such a trace does not have any correspondent derivation in ILLF. In fact, since $\multimap$ is a negative connective, focusing on $\mathcal{C}[\![Q_1]\!]$ will decompose the formula $!a \multimap !b \multimap !\mathtt{ok}$ producing the focused formula $!b \multimap !\mathtt{ok}$, which is still negative. Hence focusing cannot be lost and the inner ask has to be triggered.

This example shows something interesting: although the formulas $F \otimes G \multimap H$ and $F \multimap G \multimap H$ are logically equivalent, they are operationally different when *concurrent*

*computations* are considered. In fact, if we allow processes to consume constraints as the linear version of CCP in [11], an interleaving execution as the one in `Trace 3` may not output the constraint `ok`, since the two agents are competing for the same resources.

In order to recover interleaving executions as the one in `Trace 3`, *logical delays* [28] can be introduced.

▶ **Definition 5.** *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 $\mathcal{P}[\![\cdot]\!]_+$ as $\mathcal{P}[\![\cdot]\!]$ but replacing the following cases:

$$\begin{aligned} \mathcal{P}[\![\sum_{i\in I} \mathbf{ask}\ c_i\ \mathbf{then}\ P_i]\!]_+ &= \mathop{\&}_{i\in I} (\mathcal{C}[\![c_i]\!] \multimap \delta^+(\mathcal{P}[\![P_i]\!]_+)) \\ \mathcal{P}[\![p(\overline{x}) \stackrel{\Delta}{=} P]\!]_+ &= \forall \overline{x}.p(\overline{x}) \multimap \delta^+(\mathcal{P}[\![P]\!]_+) \end{aligned}$$

The use of delays forces the focused phase to end, *e.g.*, once the guard of the ask agent is entailed. In this encoding, we can prove a stronger adequacy theorem.

▶ **Theorem 6** (Strong adequacy [34])**.** *Let* $(\mathcal{C}, \models_\Delta)$ *be a constraint system,* $P$ *be a process and* $\Psi$ *be a set of process definitions. Then, for any constraint* $c$*,*

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

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

Now derivations in logic have a one-to-one correspondence with traces of a computation in a CCP program.

It is possible to modify the encoding to introduce negative actions (*tell*, *parallel* and *local*) during a focused phase (thus counting them as a focused step). For that, it suffices to introduce, in the encoding, negative delays $\delta^-(F)$. By using a multi-focusing systems [38], maximal parallelism semantics [9] - where all the enabled agents must all proceed in one step - can be also captured. Finally, if recursive definitions are interpreted as fixed points, more interesting properties of infinite computations can be specified and proved. See [34] for further details.

## 4 LL with multi-modalities

A careful analysis of the rules for the exponential ! in Figure 1 reveals that this connective has a differentiated behavior w.r.t. the other ones. In fact, ! is the only operator having a positive/negative behavior: the application of the right rule ($!_r$) immediately breaks focusing. Also, this is the only rule in ILLF that is *context dependent*, in the sense that it demands the linear context $\Gamma$ to be empty in order to be applied.

This distinguished character of the exponential in linear logic is akin to the behavior found in *modal connectives*. In particular, the connective ! is not *canonical*, in the sense that, if we label ! with different colors, say $b$ (for blue – $!^b$) and $r$ (for red – $!^r$), but with the same introduction rules, then it is not possible to prove, in the resulting proof system, the equivalence $!^r A \equiv !^b A$ for an arbitrary formula $A$, where $H \equiv G$ denotes the formula $(H \multimap G) \,\&\, (G \multimap H)$. Not surprisingly, this exercise would have a different outcome for any other linear logic connective. For instance, if we construct a proof system with two labeled connectives, *e.g.*, $\otimes^r$ and $\otimes^b$, together with their introduction rules, then it would be possible to prove $A \otimes^b B \equiv A \otimes^r B$ for any $A$ and $B$. This opens the possibility of defining new connectives: the colored exponentials, known as *subexponentials* [8].

## 4.1 Linear logic with subexponentials

Linear logic with subexponentials (SELL)[4] shares with intuitionistic linear logic all its connectives except the exponential: instead of having a single !, SELL may contain as many subexponentials, written $!^a$ for a label (or color) $a$, as one needs.

Such labels are organized in a pre-order, giving rise to a *subexponential signature* $\Sigma = \langle I, \preceq, U\rangle$, where $I$ is a set of labels, $U \subseteq I$ is a set specifying which subexponentials behave classically (*i.e.*, those labels that allow for weakening and contraction), and $\preceq$ is a pre-order among the elements of $I$. We shall use $a, b, \ldots$ to range over elements in $I$, and we will assume that $\preceq$ is upwardly closed with respect to $U$, *i.e.*, if $a \in U$ and $a \preceq b$, then $b \in U$.

The division of *unbounded* ($a \in U$) and linear or *bounded* ($a \notin U$) subexponentials induces also a partition of the subexponential context $\Theta$, which is split into two: a set $\Theta^u$ and a multiset $\Theta^b$ of labeled formulas, having the form

$$\Theta^u = \{a_1 : \Theta^u_1, \ldots, a_n : \Theta^u_n\} \qquad \Theta^b = \{b_1 : \Theta^b_1, \ldots, b_m : \Theta^b_m\}$$

The formulas in $\Theta^u_i$ are under the scope of the unbounded subexponential $!^{a_i}$, and formulas in $\Theta^b_j$ are under the scope of the bounded subexponential $!^{b_j}$. The linear context $\Gamma$ continues containing only negative or atomic formulas, as in ILLF.

The focused proof system SELLF [28] is constructed by adding all the rules for the intuitionistic linear logic connectives as shown in Figure 1,[5] except for the exponentials. The rules for subexponentials are the following:

- A formula $F$ under the scope of $!^a$ is stored in the exponential context $\Theta$ accordingly: if $a$ is unbounded/bounded, then $F$ is added to the set/multiset $\Theta_a$, which is created if it does not exist. This action is represented by $\Theta \uplus \{a : F\}$.

$$\frac{\Theta \uplus \{a : F\}; \Gamma \Uparrow \Delta \vdash \mathcal{R}}{\Theta; \Gamma \Uparrow !^a F, \Delta \vdash \mathcal{R}}\ !^a{}_l$$

- The unbounded decide rule in ILLF is split into bounded and unbounded versions, depending of the nature of the subexponential.

$$\frac{\Theta^u, \Theta^b; \Gamma \Downarrow F \vdash R}{\Theta^u, \Theta^b \uplus \{a : F\}; \Gamma \Uparrow \cdot \vdash \cdot \Uparrow R}\ \mathsf{Db} \qquad \frac{\Theta^u \uplus \{a : F\}, \Theta^b; \Gamma \Downarrow F \vdash R}{\Theta^u \uplus \{a : F\}, \Theta^b; \Gamma \Uparrow \cdot \vdash \cdot \Uparrow R}\ \mathsf{Du}$$

- The promotion rule has the form

$$\frac{\Theta^u_{\geq a}, \Theta^b; \cdot \Uparrow \cdot \vdash F \Uparrow}{\Theta^u, \Theta^b; \cdot \vdash !^a F \Downarrow}\ !^a{}_r$$

with the proviso that, for all $b_j : \Theta^b_j$ in $\Theta^b$, it must be the case that $a \preceq b_j$. In the premise of the rule, $\Theta^u_{\geq a} \subseteq \Theta^u$ contains only elements of the form $a_i : \Theta^u_i$ where $a \preceq a_i$ (the other contexts are weakened). That is, $!^a F$ is provable only if $F$ can be proved in the presence of subexponentials greater than $a$.

It is known that subexponentials greatly increase the expressiveness of the system when compared to linear logic. For instance, subexponentials can be used to represent contexts

[4] Although in this paper we are mostly interested in the intuitionistic version of SELL, it was proven in [3] that classical and intuitionistic subexponential logics are equally expressive. Hence we will abuse the notation and use SELL for intuitionistic linear logic system with subexponentials.

[5] Taking the extra-care of splitting the bounded context $\Theta^b$ for the multiplicative rules $\multimap_l$ and $\otimes_r$.

of proof systems [32], to mark the epistemic state of agents [27], or to specify locations in sequential computations [28]. The key difference is that, while linear logic has only seven logically distinct prefixes of bangs and question-marks (? is the dual of !), SELL allows for an unbounded number of such prefixes, *e.g.*, $!^i$, or $!^i?^j$. As we show later, by using different prefixes, we can interpret subexponentials in more creative ways, such as linear constraints, epistemic modalities or preferences. The interested reader can also check in [30, 35, 31] the interpretation of subexponentials as temporal units, and the study of dynamical subexponentials in distributed systems.

The organization of subexponentials in pre-orders brings at least two interesting aspects that can be further investigated: what kind of refinements of the proof system can be obtained by adopting richer algebraic structures for subexponentials (Section 4.2 below); and what is the proof-theoretic notion of quantification over modalities (Section 4.3 below).

Being able to quantify over subexponentials is important, *e.g.*, for specifying properties that are valid in an unbounded number of locations or agents. It is also crucial for establishing a certain notion of *mobility*, or *permissibility* of resources, that can be available, *e.g.*, iff they are marked with a label of some specific sort. But one has to be careful here: the pre-order structure is a minimal requirement in subexponential signatures in order to guarantee the *cut-elimination* property [8]. Since, in the presence of quantifiers, proving cut-elimination requires *substitution lemmas*, a naive approach of exchanging labels could invalidate such results (see [31] for an extensive discussion on the topic).

On the other hand, if we move above the pre-order minimality and consider, *e.g.*, $\wedge$-semi-lattices as subexponential structures, then the side condition in the promotion rule, $a \preceq a_i$ for all $a_i \in \Theta_{\geq a}$, is equivalent to $a \preceq \bigwedge_i a_i$. And this reflects certain kinds of preferences, as explained next.

## 4.2 Richer subexponential signatures

We now explore a refinement of SELLF, where richer structures are considered as subexponential signatures. For that, we shall use an algebraic structure that defines a mean to compare ($\preceq$) and accumulate ($\bullet$) values.

More precisely, a complete lattice monoid [12] is a tuple $CLM = \langle \mathcal{D}, \preceq, \bullet \rangle$ such that $\langle \mathcal{D}, \preceq \rangle$ is a complete lattice, $\bot$ and $\top$ are, respectively, the least and the greatest elements of $\mathcal{D}$ and $\{\mathcal{D}, \bullet, \top\}$ is a abelian monoid. Moreover, $\bullet$ distributes over lubs, *i.e.*, for all $v \in \mathcal{D}$ and $X \subseteq \mathcal{D}$, $v \bullet \sqcup X = \sqcup\{v \bullet x \mid x \in X\}$. Due to distributivity, $\bullet$ is monotone and decreasing: $a \bullet b \preceq a$.

Observe that, if the SELL signature structure is a lattice, then $a \preceq \{b, c\}$ is equivalent to $a \preceq glb(b, c)$. Moreover, in the presence of $\bullet$, promotion can be refined so to consider the combination of values as follows.

Given a SELL signature $\Sigma = \langle \mathcal{D}, \preceq, U \rangle$ with $\langle \mathcal{D}, \preceq, \bullet \rangle$ a $CLM$, the promotion rule ${!^a}_{r\bullet}$ is defined as:

$$\frac{\Theta^u_{\geq a}, \Theta^b; \cdot \Uparrow \cdot \vdash F \Uparrow}{\Theta^u, \Theta^b; \cdot \vdash !^a F \Downarrow} \; {!^a}_{r\bullet}, \text{ provided } a \preceq \bullet\{a_i, b_j\}$$

Note that, if the $CLM$ is $\bullet$-*idempotent* (*i.e.* $a \bullet a = a$), then $glb(a, b) = a \bullet b$, and the above rule coincides with SELLF's promotion rule.

▶ **Example 7.** Consider the signature $\Sigma = \langle \mathcal{D}, \preceq, \mathcal{D} \rangle$, with the following instances of $CLM$.

- $\langle \{\mathtt{pub}, \mathtt{sec}\}, \preceq, \wedge \rangle$, where `pub` and `sec` represent public and private information, respectively. The ordering is $\mathtt{pub} \prec \mathtt{sec}$ and $a \wedge b = \mathtt{sec}$ iff $a = b = \mathtt{sec}$. Hence, any proof of $\Theta; \cdot \vdash !^{\mathsf{sec}} F \Downarrow$ does not make use of any *public* information.

- $\langle [0,1], \leq_{\mathbb{R}}, \min \rangle$ (fuzzy), where $[0,1] \subset \mathbb{R}$, and $\leq_{\mathbb{R}}$ is the usual order in $\mathbb{R}$. In this case, we can interpret $!^{0.2}c$ as "$c$ is believed with preference 0.2". Note that the sequent $!^{0.2}c \otimes !^{0.7}d \vdash !^{a}(c \otimes d)$ is provable only if $a \leq_{\mathbb{R}} 0.2$.
- $\langle [0,1], \leq_{\mathbb{R}}, \times \rangle$ (probabilistic), where $\times$ is the multiplication operator in $\mathbb{R}$. This is a non-idempotent $CLM$, and the sequent $!^{0.2}c \otimes !^{0.7}d \vdash !^{a}(c \otimes d)$ is provable only if $a \leq_{\mathbb{R}} 0.14$.

In [39] we have showed that this new version of the promotion rule is not at all ad-hoc. The resulting system, SELLS, is a smooth extension of ILLF and it is a closed subsystem of SELLF, which is strict when non-idempotent $CLM$s are considered. Hence SELLS inherits all SELLF good properties such as cut-elimination.

The SELLS system has inspired the development of new CCP-based calculi where processes can tell and ask soft constraints, understood as formulas of the form $!^{a}c$ where $a$ is an element of a given $CLM$ [39]. Also, since the underlying logic is the same, it is possible to obtain adequate interpretations of processes as formulas as the ones in Section 3.2. More interestingly, it is also possible to combine, in a uniform way, different modalities [35], all of them grounded on linear logic principles. Some of these modalities will be explored in Section 5.

## 4.3 Subexponential Quantifiers

This section introduces the focused system $\mathsf{SELLF}^{\Cap}$, containing two novel connectives $\Cap$ and $\Cup$, representing, respectively, a universal and existential quantifiers over *subexponentials*.[6]

As mentioned in Section 4.1, in order to guarantee cut-elimination of the resulting system, the substitution of subexponentials in the rules for quantification should be done carefully. As showed in [31], it is enough to require that labels are substituted, bottom-up, for *smaller* ones. Also, the possibility of creating new labels dynamically implies that there should be two sorts of labels: constants and variables. This justifies the next definition.

▶ **Definition 8.** *Given a pre-order $(I, \preceq)$ and $a \in I$, the* ideal generated by *$a$ is the set $\downarrow a = \{b \in I \mid b \preceq a\}$.*

*The* subexponential signature *of* $\mathsf{SELL}^{\Cap}$ *is the triple $\Sigma = \langle I, \preceq, U \rangle$, where $I$ is a set of* subexponential constants*, $\preceq$ is a pre-order over $I$ and $U \subseteq I$ is the upwardly closed set of* unbounded constants*.*

*The sets of* typed subexponential constants *and* typed subexponential variables *are denoted respectively by*

$$\mathcal{T}_{\Sigma} = \{b : a \mid b \in \downarrow a\} \qquad \mathcal{T}_{x} = \{l_{x_1} : a_1, \ldots, l_{x_n} : a_n\}$$

*where $\{l_{x_1}, \ldots, l_{x_n}\}$ is a disjoint set of* subexponential variables*, and $\{a_1, \ldots, a_n\} \subseteq I$ are subexponential constants.*

Formally, only these subexponential constants and variables may appear free in an index of subexponential bangs and question marks.

Sequents in $\mathsf{SELLF}^{\Cap}$ have the same form as in SELLF, with the difference that there is an extra context $\mathcal{T} = \mathcal{T}_{\Sigma} \cup \mathcal{T}_{x}$.

The rules for for $\Cap$ and $\Cup$ are the novelty with respect to the focused proof system for SELLF. They behave exactly as the first-order quantifiers: the $\Cap_r$ and $\Cup_l$ belong to the

[6] Some motivation for the symbols $\Cap$ and $\Cup$. The former resembles the symbol for intersection, which is the usual semantics assigned to for all quantifiers, namely, the intersection of all models, while the latter is same for exists and union.

negative phase because they are invertible, while $\Cap_l$ and $\Cup_r$ are positive since they are not invertible.

$$\frac{\mathcal{T} \cup \{l_e : a\}; \Theta; \Gamma \Uparrow \Delta \vdash F[l_e/l_x] \Uparrow}{\mathcal{T}; \Theta; \Gamma \Uparrow \Delta \vdash \Cap l_x : a.F \Uparrow} \Cap_r \qquad \frac{\mathcal{T} \cup \{l_e : a\}; \Theta; \Gamma \Uparrow \Delta, F[l_e/l_x] \vdash \mathcal{R}}{\mathcal{T}; \Theta; \Gamma \Uparrow \Delta, \Cup l_x : a.F \vdash \mathcal{R}} \Cup_l$$

$$\frac{\mathcal{T}; \Theta; \Gamma \Downarrow F[l/l_x] \vdash R}{\mathcal{T}; \Theta; \Gamma \Downarrow \Cap l_x : a.F \vdash R} \Cap_l \qquad \frac{\mathcal{T}; \Theta; \Gamma \vdash F[l/l_x] \Downarrow}{\mathcal{T}; \Theta; \Gamma \vdash \Cup l_x : a.F \Downarrow} \Cup_r$$

In the left rule of $\Cap$ and the right rule of $\Cup$, $l_x$ is substituted with a subexponential of the right type: $l : b \in \mathcal{T}$, $b \in\!\downarrow a$. In the rules $\Cap_r$ and $\Cup_l$, a fresh variable $l_e$ of type $a$ is created and added to the context $\mathcal{T}$.

Next, we shall see that the quantifiers allows for encoding, in a modular way, systems dealing with an unbounded number of modalities.

## 5 Parametric interpretations

This section illustrates how focusing, subexponentials and quantifiers in $\mathsf{SELLF}^{\Cap}$ can be used to give adequate interpretations to CCP calculi featuring different modalities. The interpretation is *modular*: there is only one base logic – $\mathsf{SELL}^{\Cap}$; and *parametric*: each modal flavor of CCP is specified by a signature in SELL having a particular algebraic structure. In this way, processes may be executed and add/query constraints in different *locations*, where the meaning of such locations may vary, for example: spaces of computation, the epistemic state of agents, time units, levels of preferences, etc. But the underline interpretation is the same: locations in CCP become labels in SELL.

Another modular aspect of our process-as-formula interpretation is the organization of the encodings of constraints, processes and process definitions, into non-comparable *families* of subexponentials, so that focusing on an element of a family forces all elements of the other families to be erased during proof search. This ensures the discipline necessary for guaranteeing the highest level of adequacy (**FCD**).

Formally, let $M$ be an underlying set of labels, with least and greatest elements represented by *nil* and $\infty$ respectively, ordered with a pre-order $\preceq_M$. The families of subexponentials are built with marked copies of elements of $M$: $\mathfrak{c}(\cdot)$ for constraints, $\mathfrak{p}(\cdot)$ for processes, and $\mathfrak{d}(\cdot)$ for process definitions. The subexponential signature $\Sigma = \langle I, \preceq, U\rangle$ is built from $M$ in the following way:

- The set of labels is: $I = \{l, \mathfrak{c}(l), \mathfrak{p}(l), \mathfrak{d}(l) \mid l \in M\}$; that is, besides the elements in $M$, we consider three additional distinct copies of the labels, each of them marked with the appropriate family.
- The subexponential pre-order is: $l \preceq l'$ iff $l \preceq_M l'$ and $\mathfrak{f}(l) \preceq \mathfrak{f}(l')$ iff $l \preceq_M l'$ where $\mathfrak{f} \in \{\mathfrak{c}, \mathfrak{p}, \mathfrak{d}\}$; note that subexponentials pertaining to different families are not related.
- The set $U$ of unbounded subexponentials will vary depending on the encoded system.

Constraints and CCP processes are encoded into $\mathsf{SELLF}^{\Cap}$ by using the functions $\mathcal{C}[\![\cdot]\!]_l$ and $\mathcal{P}[\![\cdot]\!]_l$ as in Definition 2, now parametric w.r.t. subexponentials $l \in M$ as follows.[7]

[7] We observe that, technically, the encoding functions should also consider subexponential variables. However, the encoded processes/axioms are stored on left contexts, and the left introduction rule for universal quantifiers does not create fresh variables.

▶ **Definition 9** (General Encoding). *Constraints and axioms of the constraint system are encoded in* $\mathsf{SELL}^{\Cap}$ *as:*

$$\mathcal{C}[\![\mathit{true}]\!]_l = 1 \qquad \mathcal{C}[\![A]\!]_l = !^{\mathfrak{c}(l)}A \qquad \mathcal{C}[\![c_1 \wedge c_2]\!]_l = \mathcal{C}[\![c_1]\!]_l \otimes \mathcal{C}[\![c_2]\!]_l$$

$$\mathcal{C}[\![\exists \overline{x}.c]\!]_l = \exists \overline{x}.\mathcal{C}[\![c]\!]_l \qquad \mathcal{C}[\![\forall \overline{x}.(d \supset c)]\!] = \Cap l_x : \infty.\forall \overline{x}.(\mathcal{C}[\![d]\!]_{l_x} \multimap \mathcal{C}[\![c]\!]_{l_x})$$

*The encoding of processes and process definitions is:*

$$\begin{array}{lcl}
\mathcal{P}[\![\mathbf{tell}(c)]\!]_l & = & !^{\mathfrak{p}(l)}[\Cap l_x : l.(\mathcal{C}[\![c]\!]_{l_x})] \\
\mathcal{P}[\![\sum_{i\in I} \mathbf{ask}\ c\ \mathbf{then}\ P]\!]_l & = & !^{\mathfrak{p}(l)}[\Cap l_x : l.(\mathop{\&}_{i\in I}[\mathcal{C}[\![c_i]\!]_{l_x} \multimap \mathcal{P}[\![P_i]\!]_{l_x}])] \\
\mathcal{P}[\![(\mathbf{local}\,\overline{x})\ P]\!]_l & = & !^{\mathfrak{p}(l)}[\Cap l_x : l.\exists \overline{x}.(\mathcal{P}[\![P]\!]_{l_x})] \\
\mathcal{P}[\![P \parallel Q]\!]_l & = & \mathcal{P}[\![P]\!]_l \otimes \mathcal{P}[\![Q]\!]_l \\
\mathcal{P}[\![p(\overline{x})]\!]_l & = & !^{\mathfrak{d}(l)}p(\overline{x}) \\
\mathcal{P}[\![p(\vec{x}) \triangleq P]\!] & = & \Cap l_x : \infty.\forall \overline{x}.(!^{\mathfrak{d}(l_x)}p(\overline{x}) \multimap \mathcal{P}[\![P]\!]_{l_x})
\end{array}$$

The main difference between the encodings in $\mathsf{SELL}^{\Cap}$ and ILL is the presence of *mobility* of processes, given by the universal quantifier $\Cap$ over subexponentials. This enables the specification of systems to govern an unbounded number of modalities.

Intuitively, when (left) focusing over a quantified clause of the form $\Cap l_x : l.!^{\mathfrak{f}(l_x)}F$, a location $a \in\downarrow l$ is chosen, and $F$ becomes available in the location $a$, inside a family $\mathfrak{f}$, which is totally determined by the nature of the encoded object: $\mathfrak{c}$ for constraints, $\mathfrak{p}$ for processes, $\mathfrak{d}$ for process definitions. In the special case of $l = \infty$, $F$ can be allocated *anywhere* inside the family. This is the case for example, of axioms and process definitions.

Let us now illustrate how the use of subexponentials and quantifiers allow for attaining the highest level of adequacy. The first thing to note is that, due to the shape of the encoding, the subexponential context can be divided into 3 zones: $\mathcal{C}, \mathcal{D}$ and $\mathcal{P}$, containing the formulas marked, respectively, with subexponentials of the form $\mathfrak{c}(\cdot), \mathfrak{d}(\cdot)$ and $\mathfrak{p}(\cdot)$.

Using simple logical equivalences, we can rewrite the encoding of a constraint $\mathcal{C}[\![c]\!]_l$ so that it has the following shape $\exists \overline{x}.\left(!^{\mathfrak{c}(l_1)}A_1 \otimes \cdots \otimes !^{\mathfrak{c}(l_n)}A_n\right)$, where $A_1, \ldots, A_n$ are atomic (positive) formulas. Whenever such a formula appears in the left-hand side, it is completely decomposed and stored in the $\mathcal{C}$ context:

$$\dfrac{\dfrac{\mathcal{C} \uplus \{\mathfrak{c}(l_1) : A_1, \cdots, \mathfrak{c}(l_n) : A_n\}, \mathcal{D}, \mathcal{P}; \cdot \Uparrow \Delta \vdash \mathcal{R}}{\mathcal{C}, \mathcal{D}, \mathcal{P}; \cdot \Uparrow !^{\mathfrak{c}(l_1)}A_1, \cdots, !^{\mathfrak{c}(l_n)}A_n, \Delta \vdash \mathcal{R}}\ !^a{}_l}{\mathcal{C}, \mathcal{D}, \mathcal{P}; \cdot \Uparrow !^{\mathfrak{c}(l_1)}A_1 \otimes \cdots \otimes !^{\mathfrak{c}(l_n)}A_n, \Delta \vdash \mathcal{R}}\ \exists_l, \otimes_l$$

That is, in the negative phase, the atomic formulas $A_1, \ldots, A_n$ appearing in the premise of this derivation are moved to the contexts $\mathcal{C}$.

Consider now a derivation that focuses on the encoding of a process. For instance, let $Q = \mathbf{ask}\ c\ \mathbf{then}\ P$, and $\mathcal{P}[\![Q]\!]_l = !^{\mathfrak{p}(l)}F$, with $F = \Cap l_x : a.(\mathcal{C}[\![c]\!]_{l_x} \multimap \mathcal{P}[\![P]\!]_{l_x})$. Focusing on $F$ results necessarily in a focused derivation of the following shape:

$$\dfrac{\dfrac{\mathcal{C}'; \cdot \vdash \overset{\pi}{\mathcal{C}}[\![c]\!]_{l'} \Downarrow \qquad \dfrac{\mathcal{C}'', \mathcal{D}, \mathcal{P}' \uplus \{\mathfrak{p}(l') : F_P\}; \cdot \Uparrow \cdot \vdash G \Uparrow}{\mathcal{C}'', \mathcal{D}, \mathcal{P}'; \cdot \Downarrow \mathcal{P}[\![P]\!]_{l'} \vdash G}\ \mathsf{R}_l, !^a{}_l}{\mathcal{C}, \mathcal{D}, \mathcal{P}'; \cdot \Downarrow \Cap l_x : a(\mathcal{C}[\![c]\!]_{l_x} \multimap \mathcal{P}[\![P]\!]_{l_x}) \vdash G}\ \Cap_l, \multimap_l}{\mathcal{C}, \mathcal{D}, \mathcal{P} \uplus \{\mathfrak{p}(l) : F\}; \cdot \Uparrow \cdot \vdash \cdot \Uparrow G}\ \mathsf{Du/Db}$$

If $\mathfrak{p}(l) \in U$ (resp. $\mathfrak{p}(l) \notin U$) the rule Du (resp. Db is applied) and $\mathcal{P}' = \mathcal{P} \uplus \{\mathfrak{p}(l) : F\}$ (resp. $\mathcal{P}' = \mathcal{P}$). Since $\mathcal{C}[\![c]\!]_{l'}$ contains only positive formulas, it will be totally decomposed, and

every exponential context in $\pi$ will be a $\mathcal{C}$ context. That is, only constraints and axioms from the constraint system can be used in the proof $\pi$.

A similar analysis can be done when a process definition is selected: only the context $\mathcal{D}$, storing all the calls, can be used to entail the needed guard.

In the following, we instantiate the general definition of the encoding for different flavors of CCP. The adequacy we obtain, in each case is at the **FCD** level.

## Classical and linear CCP

For encoding the language in Section 3, the set of modalities is the simplest one: $M = \{nil, \infty\}$. All the subexponentials but $\mathfrak{p}(nil)$ and $\mathfrak{d}(\cdot)$ are unbounded.

▶ **Theorem 10.** *Let $(\mathcal{C}, \models_\Delta)$ be a constraint system, $P$ be a CCP process and $\Psi$ be a set of process definitions. Then, for any constraint $c$,*

$P \Downarrow_c$ *iff* $\cdot \Uparrow !^{\mathfrak{c}(\infty)}\mathcal{C}[\![\Delta]\!], !^{\mathfrak{p}(\infty)}[\![\Psi]\!], \mathcal{P}[\![P]\!]_{nil} \vdash \mathcal{C}[\![c]\!]_{nil} \otimes \top \Uparrow$

It is worth noticing that all the processes remain in the location $nil$ (denoting "without modality") and then, the universal quantification in the encoding is always forced to instantiate $l_x$ with $nil$.

**Linear CCP.** As we already know, the store in CCP increases monotonically: once a constraint is added, it cannot be removed from the store. This can be problematic for the specification of systems where resources can be consumed. In linear CCP (`lcc`) [11], constraints are built from formulas in the following fragment of ILL:

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

In this setting, the empty store is 1 and constraints are accumulated using $\otimes$. The extra case $!F$, as expected, is used to denote persistent constraints.

▶ **Example 11.** The *vending coffee machine* has the same CCP specification as the community coffee machine presented in Example 4. However, as expected, linear asks consume constraints when querying the store and the coin does not come back after delivering the coffee:

$$\langle \emptyset, P, 1\rangle \quad \longrightarrow \quad \langle \emptyset, \mathtt{m()}, \mathit{coin}\rangle \longrightarrow \langle \emptyset, \mathbf{tell}(\mathit{coffee}) \parallel \mathtt{m()}, 1\rangle \longrightarrow \langle \emptyset, \mathtt{m()}, \mathit{coffee}\rangle$$

In order to characterize the semantics of `lcc`, we configure the encoding in Definition 9 as follows. We declare $\mathfrak{c}(nil) \notin U$ (*i.e.*, constraints can be consumed) and $\mathfrak{c}(\infty) \in U$. Moreover, the encoding is extended for the case of unbounded constraints: $\mathcal{C}[\![!\,c]\!]_l = \mathcal{C}[\![c]\!]_\infty$. In this way, we obtain an adequacy theorem as the one in Theorem 10, also at the **FCD** level, in contrast to the weakest level of adequacy (**FCP**) obtained originally in [11] (for linear logic and without focusing).

It is important to note that the characterization in Theorem 6, that uses (vanilla) linear logic, does not work for `lcc` at the **FCD** level. Take for instance the process $Q =$ **ask** $c \otimes d$ **then** $P$ being executed in the store $!(c \otimes d)$. Clearly, $Q$ reduces to $P$ and the store remains unchanged. If we were to use the encoding in Theorem 6, before focusing on $\mathcal{P}[\![Q]\!]$, we have to do an intermediary step without an operational counterpart: focus on $c \otimes d$, stored in the classical context, to produce a copy of $c$ and $d$ in the linear context. Only after that, the implication in $\mathcal{P}[\![Q]\!]$ is able to entail the guard $c \otimes d$. In the encoding of the present section, proving the query of $Q$ results in focusing on $!^{\mathfrak{c}(nil)}c \otimes !^{\mathfrak{c}(nil)}d$. After decomposing the tensor, focusing is lost and only linear $\mathfrak{c}(nil)$ and replicated ($\mathfrak{c}(\infty)$) constraints and the axioms of the constraint systems can be used to deduce the atoms $c$ and $d$. This adequately reflects the semantics of linear asks.

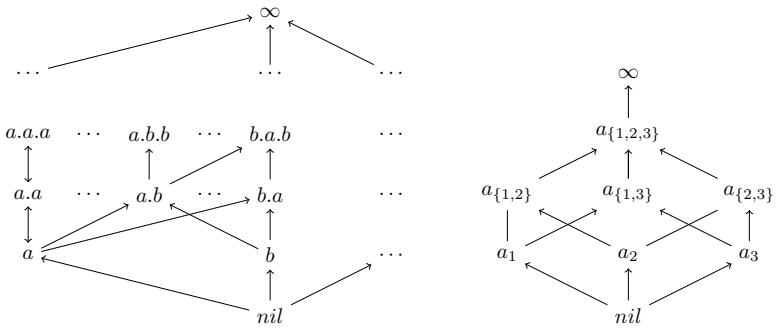


**Figure 3** Subexponential signature for `eccp`

$$\frac{(X;P,\Gamma;c)\longrightarrow(X';P',\Gamma;d)}{(X;[P]_a,\Gamma;c)\longrightarrow(X';[P]_a,P',\Gamma;d)}\ \mathrm{R_E}\qquad \frac{(X;P,\Gamma;d^a)\longrightarrow(X';P',\Gamma;d')}{(X;[P]_a,\Gamma;d)\longrightarrow(X';[P']_a,\Gamma;d\wedge s_a(d'))}\ \mathrm{R_S}$$

**Figure 4** Operational rules for `eccp` and `sccp`.

## Epistemic CCP

Now let us consider a richer system where different modalities will play a fundamental role. Epistemic CCP (`eccp`) [16] is a CCP-based language where systems of agents are considered for distributed and epistemic reasoning. In `eccp`, the constraint system is extended to consider space of agents, denoted as $s_a(c)$, and meaning "$c$ holds in the space –store– of agent $a$." The function $s_a(\cdot)$ satisfies certain conditions to reflect epistemic behaviors:

1. $s_a(1)=1$ (bottom preserving)
2. $s_a(c\wedge d)=s_a(c)\wedge s_a(d)$ (lub preserving)
3. If $d\vdash_{\Delta_e} c$ then $s_a(d)\vdash_{\Delta_e} s_a(c)$ (monotonicity)
4. $s_a(c)\vdash_{\Delta_e} c$ (believes are facts –extensiveness–)
5. $s_a(s_a(c))=s_a(c)$ (idempotence)

In `eccp`, the language of processes is extended with the constructor $[P]_a$ that represents $P$ running in the space of the agent $a$. The operational rules for $[P]_a$ are specified in Figure 4. In epistemic systems, agents are trustful, *i.e.*, if an agent $a$ knows some information $c$, then $c$ is necessarily true. Furthermore, if $b$ knows that $a$ knows $c$, then $b$ also knows $c$. For example, given a hierarchy of agents as in $[[P]_a]_b$, it should be possible to propagate the information produced by $P$ in the space $a$ to the outermost space $b$. This is captured exactly by the rule $\mathrm{R_E}$, which allows a process $P$ in $[P]_a$ to run also outside the space of agent $a$. Notice that the process $P$ is contracted in this rule. The rule $\mathrm{R_S}$, on the other hand, allows us to observe the evolution of processes inside the space of an agent. There, the constraint $d^a$ represents the information the agent $a$ may see or have of $d$, *i.e.*, $d^a=\bigwedge\{c\mid d\vdash_{\Delta_e} s_a(c)\}$. For instance, $a$ *sees* $c$ from the store $s_a(c)\wedge s_b(c')$ but it does not see $c'$.

We now configure the encoding in Definition 9 so to capture the behavior of `eccp` processes. We consider a possibly infinite set of agents $\mathcal{A}=\{a_1,a_2,...\}$ and the set of locations/modalities $M$, besides $nil$ and $\infty$, contains the set $\mathcal{A}^+$ of non-empty strings of elements in $\mathcal{A}$; for example, if $a,b\in\mathcal{A}$, then $a,b,a.a,b.a,a.b.a,\ldots\in\mathcal{A}^+$. We use $\overline{a},\overline{b},etc$ to denote elements in $\mathcal{A}^+$ and $nil$ will denote the empty string. The only linear subexponentials are $\mathfrak{d}(nil)$ and $\mathfrak{p}(nil)$. This reflects the fact that both constraints and processes in the space of an agent are unbounded, as specified by rule $\mathrm{R_E}$. Intuitively, $!^{\mathfrak{p}(1.2.3)}$ specifies a

process in the structure $[[[\cdot]_3]_2]_1$, denoting "agent 1 knows that agent 2 knows that agent 3 knows" expressions. The connective $!^{\mathfrak{c}(1.2.3)}$, on the other hand, specifies a constraint of the form $s_1(s_2(s_3(\cdot)))$. We thus extend the encoding accordingly: $\mathcal{C}[\![s_i(c)]\!]_{\bar{l}} = \mathcal{C}[\![c]\!]_{\bar{l}.i}$ and $\mathcal{P}[\![[P]_i]\!]_{\bar{l}} = \mathcal{P}[\![P]\!]_{\bar{l}.i}$.

The pre-order $\preceq$ is as depicted in Figure 3 on the left. Note that for every two different agent names $a$ and $b$ in $\mathcal{A}$, the subexponentials $a$ and $b$ are unrelated. Moreover, $a \approx a.\overline{a}$ and $b_1.b_2.\ldots.b_n \preceq \overline{a}_1.b_1.\overline{a}_2.b_2.\ldots.\overline{a}_n.b_n.\overline{a}_{n+1}$ where each $\overline{a}_i$ is a possible empty string of elements in $\mathcal{A}$. The shape of the pre-order is key for our encoding. For instance, the formula $\Cap l_x : a.b.b.\mathcal{P}[\![P]\!]_{l_x}$ on the left, allows us to place $P$ on the (outer) location $a.b$ and $b$ as required by $\mathrm{R_E}$. In fact, we can show that the sequent $\mathcal{P}[\![P]\!]_{\bar{l}.i} \vdash \mathcal{P}[\![P]\!]_{\bar{l}}$ is provable in $\mathsf{SELL}^{\Cap}$ for any process $P$ and subexponentials $\bar{l}$ and $i$. We can also show that the encoding of constraints satisfy the axioms of an epistemic constraint system. For instance, the sequent $\mathcal{C}[\![s_i(c)]\!]_{\bar{l}} \vdash \mathcal{C}[\![c]\!]_{nil}$ is provable, showing that believes are facts. Hence, a tailored version of Theorem 10 applies for this language, with the same level of adequacy.

As an interesting example of epistemic behavior, it is possible to specify common knowledge by extending the subexponential signature as in Figure 3 on the right, where for all $\mathcal{S} \subseteq \mathcal{A}$, $\overline{a} \preceq a_{\mathcal{S}}$ for any string $\overline{a} \in \mathcal{S}^+$. Then, the announcement of $c$ on the group of agents $\mathcal{S}$ can be represented by $!^{\mathfrak{c}(a_{\mathcal{S}})}c$. Notice that the sequent $!^{\mathfrak{c}(a_{\mathcal{S}})}c \vdash !^{\mathfrak{c}(\overline{a})}c \otimes \top$ can be proved for any $\overline{a} \in \mathcal{S}^+$. For instance, if $\mathcal{S} = \{a_i, a_j\}$, from $!^{\mathfrak{c}(a_{\mathcal{S}})}c$ one can prove that $a_i$ knows that $a_j$ knows that $a_i$ knows that $a_i$ knows ... $c$, *i.e.*, $c$ is common knowledge between $a_i$ and $a_j$.

## Spatial CCP

Inconsistent information in CCP arises when considering theories containing axioms such as $c \wedge d \vdash_\Delta 0$. Unlike epistemic scenarios, in spatial computations, a space can be locally inconsistent and it does not imply the inconsistency of the other spaces (*i.e.*, $s_a(0)$ does not imply $s_b(0)$). Moreover, the information produced by a process in a space is not propagated to the outermost spaces (*i.e.*, $s_a(s_b(c))$ does not imply $s_a(c)$).

In [16], spatial computations are specified in spatial CCP (`sccp`) by considering processes of the form $[P]_a$ as in the epistemic case, but excluding the rule $\mathrm{R_E}$ in the system shown in Figure 4. Furthermore, some additional requirements are imposed on the representation of agents' spaces $s_a(\cdot)$. In particular, $s_a(\cdot)$ must satisfy false containment, *i.e.*, if $c \wedge d \models_\Delta 0$, it does not necessarily imply that $s_a(c) \wedge s_b(d) \models_\Delta 0$ if $a \neq b$.

We build the subexponential signature as we did in the epistemic case but the pre-order is much simpler: for any $\overline{a} \in \mathcal{A}^+$, $\overline{a} \preceq \infty$. That is, two different elements of $\mathcal{A}^+$ are unrelated. Moreover, since `sccp` does not contain the $\mathrm{R_E}$ rule, processes in spaces are again treated linearly. Thus: $U = \{\mathfrak{c}(a) \mid a \in I\} \cup \{\mathfrak{p}(\infty)\}$.

By modifying the pre-order we partially capture the behavior of spatial systems. However, it is not enough to confine inconsistencies. In particular, note that $!^a 0 \vdash G$ for any $a$ and $G$. The solution for information confinement, as shown in [31], is to consider combinations of bangs and question marks (the dual of bang). In this case, $!^a ?^a 0 \vdash !^a ?^a G$ but $!^a ?^a 0 \nvdash !^b ?^b G$ for $a, b$ not related. Hence, the encoding remains the same, but for the base cases: atomic propositions are encoded as $!^{\mathfrak{c}(l)} ?^{\mathfrak{c}(l)} A$, and procedure calls as $!^{\mathfrak{d}(l)} ?^{\mathfrak{d}(l)} p(\vec{x})$.

## 6 Conclusion and future work

We have shown that the process-as-formula interpretation can provide useful reasoning techniques for process calculi, by faithfully capturing the behavior of processes. The inter-

| General Encoding | |
|---|---|
| Connective | Meaning |
| $\bigtriangledown_s = !^s$ | $!^s P$ is **located** at $s$. |
| $\bigtriangledown_s = !^s?^s$ | $!^s?^s P$ is **confined** to $s$. |
| $\Cap l : a\ P$ | $P$ can **move** to locations below (outside) $a$ |
| **Epistemic Modalities** | |
| Pre-order | Meaning |
| $a.a \sim a$ | Modalities are **idempotent**: $[[P]_a]_a \sim [P]_a$ |
| $a \preceq a.b$ | Processes can **move outside** $[[P]_b]_a \longrightarrow [P \parallel [P]_b]_a$ |
| **Spatial Modalities** | |
| Pre-order | Meaning |
| $a \not\preceq b$ | $P$ does not communicate with $Q$ in $[P]_a \parallel [Q]_b$ |
| $a.a \not\sim a$ | Modalities **are not** necessarily idempotent. |
| $a \not\preceq a.b$ | Processes are confined: $[[P]_b]_a \not\sim [P \parallel [P]_b]_a$ |

**Table 1** Encoding of CCP modalities in SELL$^{\Cap}$

pretations we have achieved are *modular* and *parametric*, and they can capture different modal behaviors as Table 1 summarizes.

Other examples of processes-as-formulas interpretations, relating computation and proof search, include linear logic-based models for the $\pi$-calculus [22], abstract transition systems and operational semantics [20], CCS [10], Bigraphs [5], P-systems [33] and concurrent object oriented programming languages [36]. Also, in [4] we have tailored the notion of fixed points in linear logic [2] to the system SELL$^{\Cap}$, and this allowed the encoding of CTL (Computational Tree Logic) formulas as SELL theories, thus opening the possibility of specifying and proving temporal properties inside the same logical framework.

Regarding future work, in [17] we have shown how to incorporate other modal behaviors (besides the structural ones of weakening and contraction) in linear logic, thus extending the multiplicative and additive fragment of LL with *simply dependent* multi-modalities. The interpretations we have presented here have inspired new CCP-based calculi [35]. We foresee that the finer control of modalities given in [17], as well as the extensions with *non-normal modalities* [6, 18, 7], may contribute with other declarative models of concurrency with strong logical foundations.

## References


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

2 David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Log.*, 13(1):2:1–2:44, 2012.

3 Kaustuv Chaudhuri. Classical and intuitionistic subexponential logics are equally expressive. In Anuj Dawar and Helmut Veith, editors, *CSL 2010*, volume 6247 of *LNCS*, pages 185–199. Springer, 2010.

4 Kaustuv Chaudhuri, Joëlle Despeyroux, Carlos Olarte, and Elaine Pimentel. Hybrid linear logic, revisited. *Math. Struct. Comput. Sci.*, 29(8):1151–1176, 2019.

5 Kaustuv Chaudhuri and Giselle Reis. An adequate compositional encoding of bigraph structure in linear logic with subexponentials. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *LPAR-20*, volume 9450 of *LNCS*, pages 146–161. Springer, 2015.

6 Brian F. Chellas. *Modal Logic*. Cambridge University Press, 1980. doi:10.1017/CBO9780511621192.

7 Tiziano Dalmonte, Björn Lellmann, Nicola Olivetti, and Elaine Pimentel. Hypersequent calculi for non-normal modal and deontic logics: countermodels and optimal complexity. *J. Log. Comput.*, 31(1):67–111, 2021. doi:10.1093/logcom/exaa072.

8 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Kurt Gödel Colloquium*, volume 713 of *LNCS*, pages 159–171. Springer, 1993.

9 Frank S. de Boer, Maurizio Gabbrielli, and Maria Chiara Meo. Proving correctness of timed concurrent constraint programs. *ACM Trans. Comput. Log.*, 5(4):706–731, 2004.

10 Yuxin Deng, Robert J. Simmons, and Iliano Cervesato. Relating reasoning methodologies in linear logic and process algebra. *Math. Struct. Comput. Sci.*, 26(5):868–906, 2016.

11 François Fages, Paul Ruet, and Sylvain Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Information and Computation*, 165(1):14–41, 2001.

12 Fabio Gadducci, Francesco Santini, Luis Fernando Pino, and Frank D. Valencia. Observational and behavioural equivalences for soft concurrent constraint programming. *J. Log. Algebr. Meth. Program.*, 92:45–63, 2017.

13 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

14 C. A. R. Hoare. *Communications Sequential Processes*. Prentice-Hall, Englewood Cliffs (NJ), USA, 1985.

15 Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.

16 Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, and Frank D. Valencia. Spatial and epistemic modalities in constraint-based process calculi. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *LNCS*, pages 317–332. Springer, 2012.

17 Björn Lellmann, Carlos Olarte, and Elaine Pimentel. A uniform framework for substructural logics with modalities. In Thomas Eiter and David Sands, editors, *LPAR-21*, volume 46 of *EPiC Series in Computing*, pages 435–455. EasyChair, 2017.

18 Björn Lellmann and Elaine Pimentel. Modularisation of sequent calculi for normal and non-normal modalities. *ACM Trans. Comput. Log.*, 20(2):7:1–7:46, 2019. doi:10.1145/3288757.

19 Chuck Liang and Dale Miller. Focusing and polarization in intuitionistic logic. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 451–465. Springer, 2007.

20 Raymond McDowell, Dale Miller, and Catuscia Palamidessi. Encoding transition systems in sequent calculus. *Theor. Comput. Sci.*, 294(3):411–437, 2003.

21 Dale Miller. Hereditary harrop formulas and logic programming. In *Proceedings of the VIII International Congress of Logic, Methodology, and Philosophy of Science*, pages 153–156, Moscow, August 1987.

22 Dale Miller. The pi-calculus as a theory in linear logic: Preliminary results. In Evelina Lamma and Paola Mello, editors, *ELP'92*, volume 660 of *LNCS*, pages 242–264. Springer, 1992.

23 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

24 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

25 Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.

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

27 Vivek Nigam. On the complexity of linear authorization logics. In *LICS*, pages 511–520. IEEE, 2012.

28 Vivek Nigam and Dale Miller. Algorithmic specifications in linear logic with subexponentials. In António Porto and Francisco Javier López-Fraguas, editors, *Proc. of PPDP'09*, pages 129–140. ACM, 2009.

29 Vivek Nigam and Dale Miller. A framework for proof systems. *J. Autom. Reasoning*, 45(2):157–188, 2010.

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

31 Vivek Nigam, Carlos Olarte, and Elaine Pimentel. On subexponentials, focusing and modalities in concurrent systems. *Theor. Comput. Sci.*, 693:35–58, 2017.

32 Vivek Nigam, Elaine Pimentel, and Giselle Reis. An extended framework for specifying and reasoning about proof systems. *J. Log. Comput.*, 26(2):539–576, 2016.

33 Carlos Olarte, Davide Chiarugi, Moreno Falaschi, and Diana Hermith. A proof theoretic view of spatial and temporal dependencies in biochemical systems. *Theor. Comput. Sci.*, 641:25–42, 2016.

34 Carlos Olarte and Elaine Pimentel. On concurrent behaviors and focusing in linear logic. *Theor. Comput. Sci.*, 685:46–64, 2017.

35 Carlos Olarte, Elaine Pimentel, and Vivek Nigam. Subexponential concurrent constraint programming. *Theor. Comput. Sci.*, 606:98–120, 2015.

36 Carlos Olarte, Elaine Pimentel, and Camilo Rueda. A concurrent constraint programming interpretation of access permissions. *Theory Pract. Log. Program.*, 18(2):252–295, 2018.

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

38 Elaine Pimentel, Vivek Nigam, and João Neto. Multi-focused proofs with different polarity assignments. In Mario R. F. Benevides and René Thiemann, editors, *Proc. of LSFA'15*, volume 323 of *ENTCS*, pages 163–179. Elsevier, 2015.

39 Elaine Pimentel, Carlos Olarte, and Vivek Nigam. A proof theoretic study of soft concurrent constraint programming. *Theory Pract. Log. Program.*, 14(4-5):649–663, 2014.

40 V. Saraswat and Martin Rinard. Concurrent constraint programming. In *17th ACM Symp. on Principles of Programming Languages*, pages 232–245, San Francisco, CA, January 1990.

41 Vijay A. Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.

42 Vijay A. Saraswat and Martin C. Rinard. Concurrent constraint programming. In Frances E. Allen, editor, *POPL*, pages 232–245. ACM Press, 1990.

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

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

45 Ehud Shapiro. The family of concurrent logic programming languages. *ACM Comput. Surv.*, 21(3), 1989.

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