On subexponentials, focusing and modalities in concurrent systems

Vivek Nigam

Universidade Federal da Paraíba. João Pessoa, Brazil.

Carlos Olarte

Pontificia Universidad Javeriana. Cali, Colombia.

Elaine Pimentel

Universidade Federal do Rio Grande do Norte. Natal, Brazil.
Universidade Federal de Minas Gerais. Belo Horizonte, Brazil.

Abstract

In this work we present the focused proof system SELLF$^{\Cap}$, which extends intuitionistic linear logic with subexponentials with the ability of quantifying over them, hence allowing for the use of an arbitrary number of modalities. We show that the view of subexponentials as specific modalities is general enough to give a modular encoding of different flavors of Concurrent Constraint Programming (CCP), a simple and powerful model of concurrency. More precisely, we encode CCP calculi capturing time, spatial and epistemic behaviors into SELLF$^{\Cap}$, thus providing a proof theoretic foundation for those calculi and, at the same time, setting SELLF$^{\Cap}$ as a general framework for specifying such systems.

Keywords: Linear Logic, Concurrent Constraint Programming, Proof Systems.

1. Introduction

In order to specify the behavior of distributed agents or the policies governing a distributed system, it is often necessary to reason by using different types of modalities, such as time, space, or even the epistemic state of agents. For instance, the access-control policies of a building might allow Bob to have access only in some pre-defined time, such as its opening hours. Another policy might also allow Bob to ask Alice who has higher credentials to grant him access to the building, or even specify that Bob has only access to some specific rooms of the building. Following this need, many formalisms have been proposed to specify, program and reason about such policies,

Email addresses: `vivek.nigam@gmail.com` (Vivek Nigam), `carlos.olarte@gmail.com` (Carlos Olarte), `elaine.pimentel@gmail.com` (Elaine Pimentel)

e.g., Ambient Calculus [1], Concurrent Constraint Programming [2, 3], Authorization Logics [4], just to name a few.

Logic and proof theory have often inspired the design of many of these formalisms. For example, Saraswat *et al.* proposed Concurrent Constraint Programming (CCP), a model for concurrency that combines the traditional operational view of process calculi with a declarative view based on logic [3, 5] (see [6] for a survey). Agents in CCP *interact* with each other by *telling* and *asking* information represented as *constraints* to a global store. Later, Fages *et al.* in [7] proposed Linear Concurrent Constraint (`lcc`), inspired by linear logic [8], to allow the use of linear constraints, that is, tokens of information that once used by an agent are removed from the global store.

In order to capture the behavior of distributed systems which take into account spatial, temporal and/or epistemic properties, new formalisms have been proposed. For instance, Saraswat *et al.* proposed `tcc` [9], which extends CCP with time modalities. Later, Knight *et al.* [10] proposed a CCP-based language with spatial and epistemic modalities. Some of these developments have also been followed by a similar development in proof theory. For instance, Nigam proposed a framework for linear authorization logics [11], which allow the specification of access control policies that may mention the affirmations, possessions and knowledge of principals and demonstrated that a wide range of linear authorization policies can be specified in linear logic with subexponentials (SELL) [12, 13].

This paper shows that time, spatial, and epistemic modalities can be *uniformly* specified in a single logical framework called $\text{SELLF}^{\Cap}$. Our first contribution is the introduction of the proof system $\text{SELL}^{\Cap}$, which extends intuitionistic SELL with universal ($\Cap$) and existential ($\Cup$) quantifiers over subexponentials. We demonstrate that $\text{SELL}^{\Cap}$ has good proof-theoretic properties: it admits cut-elimination and it has a complete focusing discipline [14], giving rise to the focused system $\text{SELLF}^{\Cap}$.

For our second contribution, we show that subexponentials can be interpreted as spatial, epistemic and temporal modalities, thus providing a framework for specifying concurrent systems with these modalities. This is accomplished by encoding in $\text{SELLF}^{\Cap}$ different CCP languages, for which the proposed quantifiers play an important role. For instance, they enable the use of an *arbitrary number of subexponentials*, required to model the unbounded nesting of modalities, which is a common feature in epistemic and spatial systems. This does not seem possible in existing logical frameworks such as [15] which do not contain subexponentials nor its quantifiers. Finally, the focusing discipline enforces that the obtained encodings are *faithful* w.r.t. CCP's operational semantics in a strong sense: one operational step matches exactly one logical phase. This is the strongest level of adequacy called adequacy on the level of derivations [16]. Such level of adequacy is not possible for similar encodings of linear CCP systems, such as [7].

Another important feature of subexponentials is that they can be organized into a pre-order, which specifies the provability relation among them. By coupling subexponential quantifiers with a suitable pre-order, it is possible to specify *declaratively* the rules in which agents can manipulate information. For example, an agent cannot see the information contained in a space that she does not have access to. The boundaries are naturally implied by the pre-order of subexponentials.

This work opens a number of possibilities for specifying the behavior of distributed systems. For instance, unlike [10], it seems possible in our framework to handle an infinite number of agents. Moreover, we discuss how linearity of constraints can be straightforwardly included to these systems to represent, *e.g.*, agents that can *update/change* the content of the distributed spaces. Also, by changing the underlying subexponential structure, different modalities can be put in the hands of the modelers and programmers. Finally, all the linear logic meta-theory becomes available for reasoning about distributed systems featuring modalities.

Organization. After reviewing the basic proof theory of intuitionistic linear logic and subexponentials (SELL) in Section 2, including its limitations, we propose in Section 3 an extension for it (SELL$^{\Cap}$) allowing for the quantification of subexponentials ($\Cap$ and $\Cup$). We prove that SELL$^{\Cap}$ admits cut-elimination. Section 4 discusses SELLF$^{\Cap}$, a focused proof system for SELL$^{\Cap}$. Section 5 reviews some background on CCP, for which we provide a sound and faithful encoding in SELLF$^{\Cap}$. As we shall show, our encoding is modular enough to extend it so to specify new constructs involving modalities, namely, constructs for epistemic (Section 7), spatial (Section 8) and temporal modalities (Section 9). Section 10 concludes the paper.

A preliminary short version of this paper without proofs was published in [17]. In this paper we give many more details and explanations. We also refine several technical details. Moreover, in Section 4, we present at length the focused proof system SELLF$^{\Cap}$ that is used in Sections 7, 8 and 9 for proving the adequacy results.

2. Intuitionistic linear logic and subexponentials

Although we assume that the reader is familiar with linear logic, we review some of its basic proof theory (see [18] for more details). Intuitionistic linear logic is a substructural logic proposed by Girard [8], where not all formulas are allowed to be contracted or weakened.

The grammar for formulas in intuitionistic linear logic (without exponentials) is shown below, and the proof rules for the first-order fragment of intuitionistic linear logic without exponentials are depicted in Figure 1.

$$F ::= 0 \mid 1 \mid \top \mid A \mid F_1 \otimes F_2 \mid F_1 \multimap F_2 \mid F_1 \,\&\, F_2 \mid \exists x.F \mid \forall x.F.$$

Contraction and weakening of formulas in linear logic are controlled by using the connectives ! and ? called exponentials, whose inference rules are shown below:

$$\frac{\Gamma, F \longrightarrow G}{\Gamma, !\,F \longrightarrow G}\ !_L \qquad \frac{!\,\Gamma \longrightarrow G}{!\,\Gamma \longrightarrow !\,G}\ !_R \qquad \frac{!\,\Gamma, F \longrightarrow ?G}{!\,\Gamma, ?F \longrightarrow ?G}\ ?_L \qquad \frac{\Gamma \longrightarrow G}{\Gamma \longrightarrow ?G}\ ?_R$$

$$\frac{\Gamma \longrightarrow G}{\Gamma, !\,F \longrightarrow G}\ W \qquad \frac{\Gamma, !\,F, !\,F \longrightarrow G}{\Gamma, !\,F \longrightarrow G}\ C$$

Notice that, one is only allowed to introduce a ! on the right (or a ? on the left) if all formulas in the context on the left-hand-side of the sequent must be marked with a ! and the formula on right-hand-side be marked with a ?. The rules $!_R$ and $?_L$ are commonly called *promotion rules*, while the rules $!_L$ and $?_R$ are called *dereliction rules*.

$$\frac{}{A \longrightarrow A}\ I \qquad \frac{\Gamma_1 \longrightarrow F \quad \Gamma_2, F \longrightarrow G}{\Gamma_1, \Gamma_2 \longrightarrow G}\ \text{Cut}$$

$$\frac{\Gamma, F, H \longrightarrow G}{\Gamma, F \otimes H \longrightarrow G}\ \otimes_L \quad \frac{\Gamma_1 \longrightarrow F \quad \Gamma_2 \longrightarrow H}{\Gamma_1, \Gamma_2 \longrightarrow F \otimes H}\ \otimes_R$$

$$\frac{\Gamma, F_i \longrightarrow G}{\Gamma, F_1 \,\&\, F_2 \longrightarrow G}\ \&_{L_i} \quad \frac{\Gamma \longrightarrow F \quad \Gamma \longrightarrow H}{\Gamma \longrightarrow F \,\&\, H}\ \&_R$$

$$\frac{\Gamma_1 \longrightarrow F \quad \Gamma_2, H \longrightarrow G}{\Gamma_1, \Gamma_2, F \multimap H \longrightarrow G}\ \multimap_L \quad \frac{\Gamma, F \longrightarrow H}{\Gamma \longrightarrow F \multimap H}\ \multimap_R$$

$$\frac{\Gamma, F \longrightarrow G \quad \Gamma, H \longrightarrow G}{\Gamma, F \oplus H \longrightarrow G}\ \oplus_L \quad \frac{\Gamma \longrightarrow F_i}{\Gamma \longrightarrow F_1 \oplus F_2}\ \oplus_{R_i}$$

$$\frac{\Gamma \longrightarrow G}{\Gamma, 1 \longrightarrow G}\ 1_L \quad \frac{}{\longrightarrow 1}\ 1_R \quad \frac{}{\Gamma, 0 \longrightarrow G}\ 0_L \quad \frac{}{\Gamma \longrightarrow \top}\ \top_R$$

$$\frac{\Gamma, F[e/x] \longrightarrow G}{\Gamma, \exists x.F \longrightarrow G}\ \exists_L \quad \frac{\Gamma \longrightarrow G[t/x]}{\Gamma \longrightarrow \exists x.G}\ \exists_R \quad \frac{\Gamma, F[t/x] \longrightarrow G}{\Gamma, \forall x.F \longrightarrow G}\ \forall_L \quad \frac{\Gamma \longrightarrow G[e/x]}{\Gamma \longrightarrow \forall x.G}\ \forall_R$$

Figure 1: First-order fragment of intuitionistic linear logic. As usual in the $\exists_L$ and $\forall_R$ rules, e is fresh, *i.e.*, it does not appear in Γ nor G.

As pointed out in [12, 13], the exponentials are not canonical in the following sense: consider a linear logic system containing two pairs of exponentials, one labelled with b (for blue), $!^b, ?^b$, and the other pair labeled with r (for red), $!^r, ?^r$, and their corresponding promotion and dereliction rules:

$$\frac{\Gamma, F \longrightarrow G}{\Gamma, !^r F \longrightarrow G}\ !^r{}_L \quad \frac{!^r\Gamma \longrightarrow G}{!^r\Gamma \longrightarrow !^r G}\ !^r{}_R \quad \frac{\Gamma, F \longrightarrow G}{\Gamma, !^b F \longrightarrow G}\ !^b{}_L \quad \frac{!^b\Gamma \longrightarrow G}{!^b\Gamma \longrightarrow !^b G}\ !^b{}_R$$

$$\frac{!^r\Gamma, F \longrightarrow ?^r G}{!^r\Gamma, ?^r F \longrightarrow ?^r G}\ ?^r{}_L \quad \frac{\Gamma \longrightarrow G}{\Gamma \longrightarrow ?^r G}\ ?^r{}_R \quad \frac{!^b\Gamma, F \longrightarrow ?^b G}{!^b\Gamma, ?^b F \longrightarrow ?^b G}\ ?^b{}_L \quad \frac{\Gamma \longrightarrow G}{\Gamma \longrightarrow ?^b G}\ ?^b{}_R$$

It is not possible to prove in the resulting proof system neither the equivalence $!^r F \equiv !^b F$ nor the equivalence $?^r F \equiv ?^b F$ for an arbitrary formula F, where $H \equiv G$ denotes the formula $(H \multimap G) \,\&\, (G \multimap H)$. This opens the possibility of defining new connectives: the colored exponentials. These new connectives are called *subexponentials* [13].

Not surprisingly, this exercise would have a different outcome for any other linear logic connective. That is, if we construct a proof system with two labelled connectives, e.g. $\otimes^r$ and $\otimes^b$ together with their introduction rules, then it would be possible to prove

$F \otimes^b G \equiv F \otimes^r G$ for any formulas F, G. Hence, the exponentials are the only connectives in linear logic that are not canonical.

2.1. Linear logic with subexponentials

Linear logic with subexponentials (SELL) shares with linear logic all its connectives except the exponentials: instead of having a single pair of exponentials ! and ?, SELL may contain as many *subexponentials* [12, 13], written $!^a$ and $?^a$, as one needs. The grammar of formulas in intuitionistic SELL is as follows[1]:

$$F ::= 0 \mid 1 \mid \top \mid A \mid F_1 \otimes F_2 \mid F_1 \multimap F_2 \mid F_1 \,\&\, F_2 \mid \exists x.F \mid \forall x.F \mid !^a F \mid ?^a F$$

where A denotes atomic formulas.

Formally, the proof system for SELL is specified by a *subexponential signature* $\Sigma = \langle I, \preceq, U \rangle$, where I is a set of labels (or colors), $U \subseteq I$ is a set specifying which subexponentials allow 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 system SELL is constructed by adding all the rules for the linear logic connectives as shown in Figure 1 except for the exponentials. The rules for subexponentials are added according to the subexponential signature Σ as follows: we add the introduction rules corresponding to dereliction and promotion of the subexponential labelled with $a \in I$:

$$\frac{\Gamma, F \longrightarrow G}{\Gamma, !^a F \longrightarrow G}\ !^a{}_L \qquad \frac{!^{a_1} F_1, \ldots !^{a_n} F_n \longrightarrow G}{!^{a_1} F_1, \ldots !^{a_n} F_n \longrightarrow !^a G}\ !^a{}_R$$

$$\frac{!^{a_1} F_1, \ldots !^{a_n} F_n, F \longrightarrow ?^{a_{n+1}} G}{!^{a_1} F_1, \ldots !^{a_n} F_n, ?^a F \longrightarrow ?^{a_{n+1}} G}\ ?^a{}_L \qquad \frac{\Gamma \longrightarrow G}{\Gamma \longrightarrow ?^a G}\ ?^a{}_R$$

Here, the rules $!^a{}_R$ and $?^a{}_L$ have the side condition that $a \preceq a_i$ for all i. That is, one can only introduce a $!^a$ on the right (or a $?^a$ on the left) if all other formulas in the sequent are marked with indices that are greater or equal than a.

For all indices $a \in U$, we add the following structural rules:

$$\frac{\Gamma, !^a F, !^a F \longrightarrow G}{\Gamma, !^a F \longrightarrow G}\ C \qquad \frac{\Gamma \longrightarrow G}{\Gamma, !^a F \longrightarrow G}\ W$$

That is, we are also free to specify which indices are *unbounded*, namely those appearing in the set U, and which indices are *linear* or *bounded*, namely the remaining indices.

One can show that for any subexponential signature, SELL admits cut-elimination. The proof is similar to the one given in [20].

Theorem 1. *SELL admits cut-elimination for any subexponential signature* Σ.

[1] Although in this paper we are mostly interested in the intuitionistic version of SELL, it was proven in [19] 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.

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 of proof systems [21], to mark the epistemic state of agents [11], or to specify locations in sequential computations [13].

The key difference to standard presentations of linear logic is that while linear logic has only seven logically distinct prefixes of bangs and question-marks, SELL allows for an unbounded number of such prefixes, *e.g.*, $!^i$, or $!^i?^j$. As we show later, by using different prefixes (written generically as $\bigtriangledown$), we will also be able to interpret subexponentials in more creative ways, such as temporal units [9] or spatial and epistemic modalities [10] in distributed systems.

However, SELL has a serious limitation: it does not have any sort of quantification over subexponentials. Therefore, given the interpretation above for subexponentials, it is not feasible in SELL to specify properties that are valid in an unbounded number of locations or agents. Another way of visualizing this limitation is that any sequent in any derivation in SELL has the same subexponential signature Σ; that is, the subexponential signature does not change. It does not seem possible without such quantification to encode the CCP languages with modalities that we encode later in this paper.

3. Linear Logic and Subexponential Quantifiers

This section tackles SELL's lack of ability to quantify over subexponentials by introducing the system SELL$^{\Cap}$. The system SELL$^{\Cap}$ contains two novel connectives $\Cap$ and $\Cup$, representing, respectively, a universal and existential quantifiers over *subexponentials*.[2] We first review the proof theory of the first-order quantifiers in Section 3.1 and propose new quantifiers for subexponentials in Section 3.2.

3.1. Quantifiers in The Sequent Calculus

Before we introduce formally SELL$^{\Cap}$, let us first briefly review the proof theory for the ordinary first-order quantifiers $\forall$ and $\exists$. The introduction rules for $\forall$ proof rules can be written as below [22, 23], where we show explicitly the first-order signature $\mathcal{L}$ of the terms of the language. The rules for $\exists$ are dual.

$$\frac{\mathcal{L}; \Gamma, P[t/x] \longrightarrow G}{\mathcal{L}; \Gamma, \forall x.P \longrightarrow G}\ \forall_L \qquad \frac{\mathcal{L}, e; \Gamma \longrightarrow P[e/x]}{\mathcal{L}; \Gamma \longrightarrow \forall x.P}\ \forall_R$$

Here e is a fresh constant, called *eigenvariable*, not appearing in $\mathcal{L}$, Γ and G, t is a witness, and $[t/x]$ is the usual capture-avoiding substitution of t for x. The context $\mathcal{L}$ is a set of eigenvariables used to capture the freshness of eigenvariables [23]. Intuitively, the introduction rule for the universal quantifiers says that if it is possible to prove the formula $P[e/x]$ with a generic constant e under the assumption Γ, then it is possible to

[2]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. We thank Dale Miller for this notation.

prove P for any instantiation of x under the same assumptions. This fact is reflected in the cut-elimination procedure, where the derivation with a cut

$$\dfrac{\dfrac{\overset{\Xi}{\mathcal{L}, e; \Gamma \longrightarrow P[e/x]}}{\mathcal{L}; \Gamma \longrightarrow \forall x.P}\ \forall_R \quad \dfrac{\overset{\Xi'}{\mathcal{L}; \Gamma, P[t/x] \longrightarrow G}}{\mathcal{L}; \Gamma, \forall x.P \longrightarrow G}\ \forall_L}{\mathcal{L}; \Gamma \longrightarrow G}\ cut$$

is replaced by the following derivation with a simpler cut

$$\dfrac{\overset{\Xi[t/e]}{\mathcal{L}; \Gamma \longrightarrow P[t/x]} \quad \overset{\Xi'}{\mathcal{L}; \Gamma, P[t/x] \longrightarrow G}}{\mathcal{L}; \Gamma \longrightarrow G}\ cut$$

The key observation is that $\Xi[t/e]$ is indeed a valid proof, a fact that can be verified by induction on the height of proofs [22]. This is a powerful proof theoretic insight, which says that, in order to prove $\forall x.P$, we only need to construct *one single proof with a generic variable, even if the alphabet used allows for infinitely many instantiations*.

It is desirable to have a quantification over subexponentials for which the same elegant proof theoretic argument would work. However, a main difference between eigenvariables and subexponentials is that the latter are organized in a pre-order ($\preceq$), while there is no such relation among eigenvariables. The same cut-elimination procedure would work if such a pre-order is simply the identity relation, *i.e.*, all subexponentials are disjoint. But then, all the applications of subexponentials described in [21, 13, 11] as well as the encoding of CCP languages in Sections 7, 8 and 9 would no longer be feasible, as these encodings heavily rely on the pre-order among subexponentials.

On the other hand, if we are too liberal on the relation $\preceq$ between the generic subexponential l_e appearing in the premise of the universal quantifier $\Cap$ right introduction rule, for example, then the procedure above might not work. In particular, it would no longer be possible to guarantee that the object $\Xi[l/l_e]$ obtained by replacing a fresh subexponential name l_e by a concrete subexponential l is a valid proof. This is because the induction argument used for showing that this object is a proof would fail for the case of the promotion rule, whose side-condition relies on $\preceq$.

The challenge, therefore, is to find a proof system that allows expressing more properties and, at the same time, that admits cut-elimination.

3.2. Subexponential Constants and Variables

In order to introduce $\text{SELL}^{\Cap}$, we need some terminology from lattice theory. Given a pre-order $(I, \preceq)$, the *ideal* of an element $a \in I$ in $\preceq$, written $\downarrow a$, is the set $\{x \mid x \preceq a\}$.

The subexponential signature of $\text{SELL}^{\Cap}$ is of the form

$$\Sigma = \langle I, \preceq, F, U \rangle,$$

where I is a set of *subexponential constants* and $\preceq$ is a pre-order among these constants. The new component $F = \{\mathfrak{f}_1, \ldots, \mathfrak{f}_n\}$ specifies families of subexponentials indices. In particular, a family $\mathfrak{f} \in F$ takes an element of $a \in I$ and returns a subexponential index $\mathfrak{f}(a)$. As it will be clear below, these families allow for the specification of disjoint pre-orders based on $\langle I, \preceq \rangle$. Finally, the set $U \subseteq \{\mathfrak{f}(a) \mid a \in I, \mathfrak{f} \in F\}$ is a set of unbounded subexponentials generated from families, and as before, it is upwardly closed with respect to $\preceq$: if $b \preceq a$, where $a, b \in I$, and $\mathfrak{f}(b) \in U$ then $\mathfrak{f}(a) \in U$. Notice that if $\mathfrak{f}$ is the identity function (id), then $\mathrm{SELL}^{\Cap}$ is a conservative extension of SELL. That is, the $\mathrm{SELL}^{\Cap}$ system obtained from the signature $\langle I, \preceq, \{id\}, U\rangle$ conservatively extends the SELL system obtained from $\langle I, \preceq, U\rangle$.

For our subexponential quantification, we will be interested in determining whether a subexponential b belongs or not to the ideal $\downarrow a$ of a given subexponential a. This is formally achieved by adding a typing information to subexponentials. Given a subexponential signature $\Sigma = \langle I, \preceq, F, U\rangle$, the judgment $b : a$ is true whenever $b \in \downarrow a$, *i.e.*, $b \preceq a$. Thus we obtain the following set of typed *subexponential constants*:

$$\mathcal{A}_\Sigma = \{b : a \mid a, b \in I, b \preceq a\}.$$

As with the universal quantifier $\forall$, which introduces *eigenvariables* to the signature, the universal quantification for subexponentials $\Cap$ introduces *subexponential variables* $l_x : a$, where a is a subexponential constant, *i.e.*, $a \in I$. Thus, $\mathrm{SELL}^{\Cap}$ sequents have the form $\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow G$, where

$$\mathcal{A} = \mathcal{A}_\Sigma \cup \{l_{x_1} : a_1, \ldots, l_{x_n} : a_n\},$$

$\{l_{x_1}, \ldots, l_{x_n}\}$ 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.

The grammar of the formulas of $\mathrm{SELL}^{\Cap}$ extends the formulas of SELL by lifting the definition of families to typed subexponentials and by adding the subexponential quantifiers as follows:

$$F ::= 0 \mid 1 \mid \top \mid A \mid \cdots \mid !^s F \mid ?^s F \mid \Cap l_x : a.F \mid \Cup l_x : a.F$$

where $l_x : a$ is a (typed) subexponential variable, and s is a subexponential index, *i.e.*, either $s = \mathfrak{f}(l_x : a)$ or $s = \mathfrak{f}(a : a')$. The introduction rules for the subexponential quantifiers look similar to those introducing the first-order quantifiers, but instead of manipulating the context $\mathcal{L}$, they manipulate the context $\mathcal{A}$:

$$\frac{\mathcal{A}; \mathcal{L}; \Gamma, F[l/l_x] \longrightarrow G}{\mathcal{A}; \mathcal{L}; \Gamma, \Cap l_x : a.F \longrightarrow G}\ \Cap_L \qquad \frac{\mathcal{A}, l_e : a; \mathcal{L}; \Gamma \longrightarrow G[l_e/l_x]}{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow \Cap l_x : a.G}\ \Cap_R$$

$$\frac{\mathcal{A}, l_e : a; \mathcal{L}; \Gamma, F[l_e/l_x] \longrightarrow G}{\mathcal{A}; \mathcal{L}; \Gamma, \Cup l_x : a.F \longrightarrow G}\ \Cup_L \qquad \frac{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow G[l/l_x]}{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow \Cup l_x : a.G}\ \Cup_R$$

where $l : b \in \mathcal{A}$, $b \preceq a$ and l_e is fresh, *i.e.*, not appearing in $\mathcal{A}$ nor $\mathcal{L}$.

Intuitively, subexponential variables play a similar role as eigenvariables. The generic variable $l_x : a$ represents *any subexponential, constant or variable,* that is in the ideal of a. Hence it can be substituted by any subexponential l of type b, with $b \preceq a$. This is formalized by defining a pre-order, called *sequent pre-order* and written $\preceq_{\mathcal{A}}$, from the context $\mathcal{A}$ of a given sequent, and the subexponential signature $\langle I, \preceq, F, U\rangle$. This pre-order is formally used in the side condition of the promotion rule and it is defined as the *transitive* and *reflexive closure* of the sets below.

$$\{\mathfrak{f}(a_i : b_i) \preceq_{\mathcal{A}} \mathfrak{f}(a_j : b_j) \mid \mathfrak{f} \in F, a_i, a_j \in I \text{ and } a_i \preceq a_j\} \ \cup$$
$$\{\mathfrak{f}(l_x : b_i) \preceq_{\mathcal{A}} \mathfrak{f}(a_j : b_j) \mid \mathfrak{f} \in F, l_x : b_i \in \mathcal{A}, l_x \notin I, a_j \in I \text{ and } b_i \preceq a_j\}$$

The first component of this set specifies that families preserve the pre-order $\preceq$ in Σ only involving subexponential constants; thus $\preceq_{\mathcal{A}}$ is a conservative extension of $\preceq$. The second component is the interesting one, which relates subexponential obtained from variables and subexponentials obtained from constants: $l_x : b_i$ means that l_x belongs to the ideal of b_i and if $b_i \preceq a_j$, then $\mathfrak{f}(l_x : b_i) \preceq_{\mathcal{A}} \mathfrak{f}(a_j : b_j)$. Notice that $\mathfrak{f}(l_x : a)$ and $\mathfrak{f}(l_y : b)$ are unrelated for *any* two different subexponential variables l_x and l_y.

The pre-order $\preceq_{\mathcal{A}}$ is used in the right-introduction of bangs and the left-introduction of question-marks in a similar way as before in SELL.

$$\dfrac{\mathcal{A}; \mathcal{L}; !^{\mathfrak{f}(l_1 :\, a_1)} F_1, \ldots !^{\mathfrak{f}(l_n :\, a_n)} F_n \longrightarrow G}{\mathcal{A}; \mathcal{L}; !^{\mathfrak{f}(l_1 :\, a_1)} F_1, \ldots, !^{\mathfrak{f}(l_n :\, a_n)} F_n \longrightarrow !^{\mathfrak{f}(l \,:\, a)} G} \; {!^{\mathfrak{f}(l:a)}}_R$$

$$\dfrac{\mathcal{A}; \mathcal{L}; !^{\mathfrak{f}(l_1 :\, a_1)} F_1, \ldots !^{\mathfrak{f}(l_n :\, a_n)} F_n, P \longrightarrow ?^{\mathfrak{f}(l_{n+1} :\, a_{n+1})} G}{\mathcal{A}; \mathcal{L}; !^{\mathfrak{f}(l_1 :\, a_1)} F_1, \ldots, !^{\mathfrak{f}(l_n :\, a_n)} F_n, ?^{\mathfrak{f}(l \,:\, a)} P \longrightarrow ?^{\mathfrak{f}(l_{n+1} :\, a_{n+1})} G} \; {?^{\mathfrak{f}(l:a)}}_L$$

where $\{l : a, l_1 : a_1, \ldots, l_{n+1} : a_{n+1}\} \in \mathcal{A}$ and with the side condition that for all $1 \leq i \leq n+1$, $\mathfrak{f}(l : a) \preceq_{\mathcal{A}} \mathfrak{f}(l_i : a_i)$.

Notice that bangs and question marks use families, while quantifiers use only constants and variables. This interplay allows us to bind formulas with different families, such as in the formula:

$$\Cap l_x : a.[!^{\mathfrak{f}(l_x \,:\, a)} P \otimes !^{\mathfrak{g}(l_x \,:\, a)} P'].$$

As pointed out in [12], for cut-elimination, one needs to be careful with the structural properties of subexponentials. For subexponential variables, we define $\mathfrak{f}(l_x : a)$ to be always bounded, while for subexponential constants, it is similar as before: if $\mathfrak{f}(a : b) \in U$, then structural rules can be applied.

We can now show our desired result, namely, that SELL$^{\Cap}$ admits cut-elimination.

Theorem 2. *For any signature Σ, the proof system SELL$^{\Cap}$ admits cut-elimination.*

Proof. We show only the new principal case that arises from the inclusion of $\Cap, \Cup$. The reduction follows the same idea as for the first-order quantifiers: the deduction

$$\dfrac{\dfrac{\overset{\Xi}{\mathcal{A}, l_e : a; \mathcal{L}; \Gamma \longrightarrow F[l_e/l_x]}}{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow \Cap l_x : a.F} \; \Cap_R \quad \dfrac{\overset{\Xi'}{\mathcal{A}; \mathcal{L}; \Gamma, F[l/l_x] \longrightarrow G}}{\mathcal{A}; \mathcal{L}; \Gamma, \Cap l_x : a.F \longrightarrow G} \; \Cap_L}{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow G} \; cut$$

is replaced by

$$\dfrac{\overset{\Xi[l/l_e]}{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow F[l/l_x]} \quad \overset{\Xi'}{\mathcal{A}; \mathcal{L}; \Gamma, F[l/l_x] \longrightarrow G}}{\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow G} \; cut$$

Observe that we have the typing $l_x : a$ and $l : b$ with $b \preceq a$ for some $b \in I$. We can show by induction that the object $\Xi[l/l_e]$ is indeed a SELL$^{\Cap}$ proof. The only interesting cases are for the right introduction rule for $!^s$ and the left introduction rule for $?^s$. We show only the former, as the latter follows similarly. There are two sub-cases to consider, when s is of the form $\mathfrak{f}(l_e : a)$ or when s is of the form $\mathfrak{f}(a' : a)$ and a' is a subexponential constant. We only show the former case, as the latter follows similarly. Assume that the formula $!^{\mathfrak{f}(l_e:a)}H$ is introduced. Then all formulas in the context are either of the shape $!^{\mathfrak{f}(l_e:a)}H'$ or $!^{\mathfrak{f}(b_i:a_i)}H_i$ with $b_i \in I$ and with $a \preceq b_i$. As b is in the ideal of a, the formula $!^{\mathfrak{f}(l:b)}H$ can be introduced and $\Xi[l/l_e]$ is a proof. ◻

Finally, we observe that there seems to be other ways of quantifying subexponentials. For instance, while here different subexponential variables are not related to each other, it seems possible to specify proof systems where these can be related. However, as this is not needed in our encodings of process calculi with modalities, we leave this possibility as future work.

Notation 1. *Since at some points we may have too many sub and super scripts, families, types, etc, we will set some notation for the remainder of the paper. As already stablished, we will use: $a, b, a_1, b_1, \ldots$ for subexponential constants, (belonging to I); $l_e, l_h, l_y, l_x, l_{x_1}, \ldots$ for subexponential variables; and $l, l_1, \ldots$ for representing subexponentials in general (constants or variables). We shall also write ℓ for $(l : a)$ and $!^{\mathfrak{f}(\ell)}$ instead of $!^{\mathfrak{f}(l:\,a)}$ when the type "a" can be inferred from the context. Also, for the sake of readability, we will continue writing $\mathfrak{f}(a)$ instead of $\mathfrak{f}(a : a)$, for $a \in I$. Finally, we shall use k, s to denote subexponential indices when the type "$: a$" and the family "$\mathfrak{f}$" are unimportant. That is, when we write $!^s F$, we mean $!^{\mathfrak{f}(\ell)}F$. Similarly for "?".*

4. Focused Proof System for SELL$^{\Cap}$

Focusing is a discipline on proofs first proposed for linear logic by Andreoli in the context of logic programming to reduce the non-determinism during proof search [14]. Focused proofs can be interpreted as the *normal form proofs for proof search*. We use focusing in Sections 6, 7 and 8 to prove the adequacy of our encodings of CCP languages with modalities mentioned in the introduction.

The focused proof system (SELLF) for classical linear logic with subexponentials was proposed in [20]. This section extends with the subexponential quantifiers the intuitionistic version of SELLF. The rules for the resulting system, called SELLF$^{\Cap}$, is depicted in Figure 3.

In order to explain SELLF$^{\Cap}$, however, we need some more terminology. We classify as *negative* all formulas whose main connective is $\&, \multimap, \forall, ?^s, \Cap$ and the unit $\top$, and classify the remaining formulas (both non-atomic and atomic) as *positive*. Similarly, *positive* rules are those that introduce positive formulas to the right-hand-side of sequents and negative formulas to the left-hand-side of sequents, *e.g.*, $\exists_R, \multimap_L$. *Negative* rules are those that introduce negative formulas to the right-hand-side of sequents and positive formulas to the left-hand-side of sequents, *e.g.*, $\forall_R, \otimes_L$.

This distinction between positive and negative phases is natural as all negative rules are invertible rules, that is, provability is not affected when applying such a rule. For example, the rule $\forall_R$ belongs to the negative phase, as the choice of the name used for the eigenvariable is not important for provability, as long as it is fresh. A positive rule, on the other hand, is possibly non-invertible and therefore provability may be lost. For instance, the $\exists_R$ rule belongs to the positive phase: one needs to provide a witness t for that rule.

$$\bullet\ (\mathcal{K}_1 \otimes \mathcal{K}_2)[s] = \begin{cases} \mathcal{K}_1[s] \uplus \mathcal{K}_2[s] & \text{if } s \notin U \\ \mathcal{K}_1[s] & \text{if } s \in U \end{cases} \qquad \bullet\ \mathcal{K}[\mathcal{S}] = \biguplus\{\mathcal{K}[s] \mid s \in \mathcal{S}\}$$

$$\bullet\ (\mathcal{K} +_k F)[s] = \begin{cases} \mathcal{K}[s] \cup \{F\} & \text{if } (s = k) \wedge (k \in U) \\ \mathcal{K}[s] \uplus \{F\} & \text{if } (s = k) \wedge (k \notin U) \\ \mathcal{K}[s] & \text{otherwise} \end{cases} \qquad \bullet\ \mathcal{K} \leq_k [s] = \begin{cases} \mathcal{K}[s] & \text{if } k \preceq_{\mathcal{A}} s \\ \emptyset & \text{if } k \not\preceq_{\mathcal{A}} s \end{cases}$$

$$\bullet\ (\mathcal{K}_1 \star \mathcal{K}_2) \mid_{\mathcal{S}} \text{ is true if and only if } (\mathcal{K}_1[s] \star \mathcal{K}_2[s]) \text{ for all } s \in \mathcal{S}.$$

Figure 2: Operations on contexts. Here, $s \in \mathcal{A}$, $\mathcal{S} \subseteq \mathcal{A}$, and the binary connective $\star \in \{=, \subset, \subseteq\}$.

As in the focused system for classical linear logic with subexponentials [13], we make use of indexed contexts $\mathcal{K}$ that maps a subexponential index to multiset of formulas, *e.g.*, if s is a subexponential index, then $\mathcal{K}[s]$ is a multiset of formulas, where intuitively they are all marked with $!^s$. That is, $\mathcal{K}[s] = \{F_1, \ldots, F_n\}$ should be interpreted as the multiset of formulas $!^s F_1, \ldots, !^s F_n$. We also make use of the operations on contexts depicted in Figure 2. Most of the operations are straightforward. For instance, $\mathcal{K}_1 \otimes \mathcal{K}_2[s]$ is used to specify the tensor right introduction rule ($\otimes_R$) and linear implication left rule ($\multimap_L$). $\mathcal{K}_1 \otimes \mathcal{K}_2[s]$ is defined as follows: when s is a bounded subexponential index, $\mathcal{K}_1 \otimes \mathcal{K}_2[s]$ is obtained by multiset union of $\mathcal{K}_1[s]$ and $\mathcal{K}_2[s]$, and when s is an unbounded subexponential index, then it is $\mathcal{K}_1[s]$.[3]

The rules of the system are depicted in Figure 3 containing four types of sequents.

- $[\mathcal{K} : \Gamma], \Delta \longrightarrow \mathcal{R}$ is an unfocused sequent, where $\mathcal{R}$ is either a bracketed formula $[F]$ or an unbracketed one. Here Γ contains only atomic or negative formulas, while $\mathcal{K}$ is the indexed context containing formulas whose main connective is a $!^s$ for some subexponential index s.
- $[\mathcal{K} : \Gamma] \longrightarrow [F]$ is a sequent representing the end of the negative phase.
- $[\mathcal{K} : \Gamma] -_F\!\!\rightarrow$ is a sequent focused on the right.
- $[\mathcal{K} : \Gamma] \xrightarrow{F} G$ is a sequent focused on the left.

As one can see from inspecting the proof system in Figure 3, proofs are composed of two alternating phases: a *negative phase*, containing sequent of the first form above and where all the negative non-atomic formulas to the right and all the positive non-atomic formulas to the left are introduced. Atomic or positive formulas to the right and atomic or negative formulas to the left are bracketed by the $[]_L$ and $[]_R$ rules, while formulas whose main connective is a $!^s$ are added to the indexed context $\mathcal{K}$ by rule ${!^s}_L$. The second type of sequent above marks the end of the negative phase. A *positive phase* starts by using the decide rules to focus either on a formula on the right or on the left, resulting on the third and fourth sequents above. Then one introduces all the positive formulas to the right and the negative formulas to the left, until one is focused

[3]As specified by the side-condition of the $\otimes_R$ and $\multimap_L$ rule in Figure 3, there is an invariant that $\mathcal{K}_1[s] = \mathcal{K}_2[s]$ when s is unbounded.

Negative Phase

$$\dfrac{}{[\mathcal{K}:\Gamma],\Delta \longrightarrow \top}\ \top_R \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta,F,G \longrightarrow \mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,F\otimes G \longrightarrow \mathcal{R}}\ \otimes_L \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta,F \longrightarrow G}{[\mathcal{K}:\Gamma],\Delta \longrightarrow F \multimap G}\ \multimap_R$$

$$\dfrac{[\mathcal{K}:\Gamma],\Delta \longrightarrow G[x_e/x]}{[\mathcal{K}:\Gamma],\Delta \longrightarrow \forall x.G}\ \forall_R \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta,G[x_e/x] \longrightarrow \mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,\exists x.G \longrightarrow \mathcal{R}}\ \exists_L \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta \longrightarrow \mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,1 \longrightarrow \mathcal{R}}\ 1_L$$

$$\dfrac{[\mathcal{K}_{l_e}:\Gamma],\Delta \longrightarrow G[l_e/l_x]}{[\mathcal{K}:\Gamma],\Delta \longrightarrow \Cap l_x : a.G}\ \Cap_R \qquad \dfrac{[\mathcal{K}_{l_e}:\Gamma],\Delta,G[l_e/l_x] \longrightarrow \mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,\Cup l_x : a.G \longrightarrow \mathcal{R}}\ \Cup_L \qquad \dfrac{}{[\mathcal{K}:\Gamma],\Delta,0 \longrightarrow \mathcal{R}}\ 0_L$$

$$\dfrac{[\mathcal{K}+_s F:\Gamma],\Delta \longrightarrow \mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,!^s F \longrightarrow \mathcal{R}}\ !^s{}_L \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta \longrightarrow F \quad [\mathcal{K}:\Gamma],\Delta \longrightarrow G}{[\mathcal{K}:\Gamma],\Delta \longrightarrow F \,\&\, G}\ \&_R \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta,F \longrightarrow \mathcal{R} \quad [\mathcal{K}:\Gamma],\Delta,H \longrightarrow \mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,F\oplus H \longrightarrow \mathcal{R}}\ \oplus_L$$

Positive Phase

$$\dfrac{[\mathcal{K}_1:\Gamma_1]\!-_F\!\!\rightarrow \quad [\mathcal{K}_2:\Gamma_2]\!-_G\!\!\rightarrow}{[\mathcal{K}_1\otimes\mathcal{K}_2:\Gamma_1,\Gamma_2]\!-_{F\otimes G}\!\!\rightarrow}\ \otimes_R,\text{ where } (\mathcal{K}_1=\mathcal{K}_2)|_U \qquad \dfrac{[\mathcal{K}:\Gamma]\xrightarrow{F[l/l_x]}[G]}{[\mathcal{K}:\Gamma]\xrightarrow{\Cap l_x:a.F}[G]}\ \Cap_L$$

$$\dfrac{[\mathcal{K}_1:\Gamma_1]\!-_F\!\!\rightarrow \quad [\mathcal{K}_2:\Gamma_2]\xrightarrow{H}[G]}{[\mathcal{K}_1\otimes\mathcal{K}_2:\Gamma_1,\Gamma_2]\xrightarrow{F\multimap H}[G]}\ \multimap_L\text{ , where } (\mathcal{K}_1=\mathcal{K}_2)|_U \qquad \dfrac{[\mathcal{K}:\Gamma]\!-_{G[l/l_x]}\!\!\rightarrow}{[\mathcal{K}:\Gamma]\!-_{\Cup l_x:a.G}\!\!\rightarrow}\ \Cup_R$$

$$\dfrac{[\mathcal{K}:\Gamma]\!-_{G_i}\!\!\rightarrow}{[\mathcal{K}:\Gamma]\!-_{G_1\oplus G_2}\!\!\rightarrow}\ \oplus_{R_i} \qquad \dfrac{[\mathcal{K}:\Gamma]\xrightarrow{F_i}[G]}{[\mathcal{K}:\Gamma]\xrightarrow{F_1\& F_2}[G]}\ \&_{L_i} \qquad \dfrac{}{[\mathcal{K}:\Gamma]\!-_1\!\!\rightarrow}\ 1_R$$

$$\dfrac{[\mathcal{K}:\Gamma]\!-_{G[t/x]}\!\!\rightarrow}{[\mathcal{K}:\Gamma]\!-_{\exists x.G}\!\!\rightarrow}\ \exists_R \qquad \dfrac{[\mathcal{K}:\Gamma]\xrightarrow{F[t/x]}[G]}{[\mathcal{K}:\Gamma]\xrightarrow{\forall x.F}[G]}\ \forall_L \qquad \dfrac{[\mathcal{K}\leq_s:\cdot]\longrightarrow F}{[\mathcal{K}:\cdot]\!-_{!^s F}\!\!\rightarrow}\ !^s{}_R\ \blacklozenge$$

$$\dfrac{[\mathcal{K}\leq_s:\cdot],F\longrightarrow[\cdot]}{[\mathcal{K}:\cdot]\xrightarrow{?^s F}[?^k G]}\ ?^s{}_L\ \blacklozenge\text{ and } k\in U\wedge s\not\preceq k \qquad \dfrac{[\mathcal{K}\leq_s:\cdot],F\longrightarrow[?^k G]}{[\mathcal{K}:\cdot]\xrightarrow{?^s F}[?^k G]}\ ?^s{}_L\ \blacklozenge\text{ and } s\preceq k$$

$$\dfrac{}{[\mathcal{K}:\Gamma]\!-_A\!\!\rightarrow}\ I_R\text{ given } A\in(\Gamma\uplus\mathcal{K}[\mathcal{I}])\text{ and }(\Gamma\uplus\mathcal{K}[\mathcal{I}\setminus\mathcal{U}])\subseteq\{A\}$$

Structural Rules

$$\dfrac{[\mathcal{K}:\Gamma,N_a],\Delta\longrightarrow\mathcal{R}}{[\mathcal{K}:\Gamma],\Delta,N_a\longrightarrow\mathcal{R}}\ []_L \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta\longrightarrow[P_a]}{[\mathcal{K}:\Gamma],\Delta\longrightarrow P_a}\ []_R \qquad \dfrac{[\mathcal{K}:\Gamma],P_a\longrightarrow[F]}{[\mathcal{K}:\Gamma]\xrightarrow{P_a}[F]}\ R_L \qquad \dfrac{[\mathcal{K}:\Gamma]\longrightarrow N}{[\mathcal{K}:\Gamma]\!-_N\!\!\rightarrow}\ R_R$$

$$\dfrac{[\mathcal{K}:\Gamma]\xrightarrow{NA}[G]}{[\mathcal{K}+_s NA:\Gamma]\longrightarrow[G]}\ D_L,\text{if } s\notin U \qquad \dfrac{[\mathcal{K}+_s NA:\Gamma]\xrightarrow{NA}[G]}{[\mathcal{K}+_s NA:\Gamma]\longrightarrow[G]}\ D_L,\text{if } s\in U$$

$$\dfrac{[\mathcal{K}:\Gamma]\xrightarrow{NA}[G]}{[\mathcal{K}:\Gamma,F]\longrightarrow[G]}\ D_L \qquad \dfrac{[\mathcal{K}:\Gamma]\!-_G\!\!\rightarrow}{[\mathcal{K}:\Gamma]\longrightarrow[G]}\ D_R \qquad \dfrac{[\mathcal{K}:\Gamma]\!-_G\!\!\rightarrow}{[\mathcal{K}:\Gamma]\longrightarrow[?^s G]}\ D_R^{?^s} \qquad \dfrac{[\mathcal{K}:\Gamma],\Delta\longrightarrow[?^s F]}{[\mathcal{K}:\Gamma],\Delta\longrightarrow ?^s F}\ ?^s{}_R$$

Figure 3: Focused Proof System for Intuitionistic Linear Logic with Subexponentials SELLF$^{\Cap}$. Here, $\mathcal{R}$ stands for either a bracketed context, $[F]$, or an unbracketed context. A is an atomic formula; P_a is a positive or atomic formula; N is a negative formula; NA is a non-atomic formula; and N_a is a negative or atomic formula. In the $?^s{}_L$ and $!^s{}_R$ rules, $\blacklozenge$ stands for "given $\mathcal{K}[\{x \mid s\not\preceq x\wedge x\notin U\}]=\emptyset$]." Finally $\mathcal{K}_{l_e}$ is obtained by extending the domain of $\mathcal{K}$ with $\{\mathfrak{f}(l_e:a)\mid \mathfrak{f}\in F\}$ and mapping these to the empty set.

either on a negative formula on the right or a positive formula on the left. This point marks the end of the positive phase by using the R_L and R_R rules and starting another negative phase.

The rules for for $\Cap$ and $\Cup$ are the novelty with respect to the focused proof system for SELL. They behave exactly as the first-order quantifiers: the $\Cap_R$ and $\Cup_L$ belong to the negative phase because they are invertible, while $\Cap_L$ and $\Cup_R$ are positive because they are not-invertible. Notice that in the premise of $\Cap_R$ and $\Cup_L$ rules, the context $\mathcal{K}$ is extended to $\mathcal{K}_{l_e}$ with new indices $\{\mathfrak{f}(l_e : a) \mid \mathfrak{f} \in F\}$ generated due to the creation of fresh subexponential constant. Since no formulas are yet in these contexts, these are mapped to the empty set.

One can prove the following completeness theorem following the same lines as the proof in [20] for the focused proof system for classical linear logic with subexponentials based on the methodology proposed in [24]. One shows that any SELL$^{\Cap}$ proof can be transformed into a focused proof. The proof relies on the following key permutation lemmas.

Lemma 3. *All rules permute over a negative rule, including $\Cap_R$ and $\Cup_L$.*

Proof. We show some of the cases involving $\Cup_L$. The other cases are similar.

- $\otimes_R$ permutes over $\Cup_L$:

$$\dfrac{\dfrac{\Gamma_1, F[l_e/l_x] \longrightarrow H}{\Gamma_1, \Cup l_x : a.F \longrightarrow H}\ \Cup_L \qquad \Gamma_2 \longrightarrow G}{\Gamma_1, \Gamma_2, \Cup l_x : a.F \longrightarrow H \otimes G}\ \otimes_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1, F[l_e/l_x] \longrightarrow H \quad \Gamma_2 \longrightarrow G}{\Gamma_1, \Gamma_2, F[l_e/l_x] \longrightarrow H \otimes G}\ \otimes_R}{\Gamma_1, \Gamma_2, \Cup l_x : a.F \longrightarrow H \otimes G}\ \Cup_L$$

- $\exists_R$ permutes over $\Cup_L$:

$$\dfrac{\dfrac{\Gamma, F[l_e/l_x] \longrightarrow G[t/y]}{\Gamma, \Cup l_x : a.F \longrightarrow G[t/y]}\ \Cup_L}{\Gamma, \Cup l_x : a.F \longrightarrow \exists y.G}\ \exists_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, F[l_e/l_x] \longrightarrow G[t/y]}{\Gamma, F[l_e/l_x] \longrightarrow \exists y.G}\ \exists_R}{\Gamma, \Cup l_x : a.F \longrightarrow \exists y.G}\ \Cup_L$$

- $\multimap_R$ permutes over $\Cup_L$:

$$\dfrac{\dfrac{\Gamma, F[l_e/l_x], G \longrightarrow H}{\Gamma, \Cup l_x : a.F, G \longrightarrow H}\ \Cup_L}{\Gamma, \Cup l_x : a.F \longrightarrow G \multimap H}\ \multimap_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, F[l_e/l_x], G \longrightarrow H}{\Gamma, F[l_e/l_x] \longrightarrow G \multimap H}\ \multimap_R}{\Gamma, \Cup l_x : a.F \longrightarrow G \multimap H}\ \Cup_L$$

- $\Cap_R$ permutes over $\Cup_L$:

$$\dfrac{\dfrac{\Gamma, F[l_e/l_x] \longrightarrow G[l_h/l_y]}{\Gamma, \Cup l_x : a.F \longrightarrow G[l_h/l_y]}\ \Cup_L}{\Gamma, \Cup l_x : a.F \longrightarrow \Cap l_y : b.G}\ \Cap_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, F[l_e/l_x] \longrightarrow G[l_h/l_y]}{\Gamma, F[l_e/l_x] \longrightarrow \Cap l_y : b.G}\ \Cap_R}{\Gamma, \Cup l_x : a.F \longrightarrow \Cap l_y : b.G}\ \Cup_L$$

∎

Lemma 4. *All positive rules permute over a positive rule, including $\Cap_L$ and $\Cup_R$.*

Proof. We show some of the cases involving $\Cap_L$. The other cases are similar.

- $\otimes_R$ permutes over $\Cap_L$:

$$\dfrac{\dfrac{\Gamma_1, F[l/l_x] \longrightarrow H}{\Gamma_1, \Cap l_x : a.F \longrightarrow H}\ \Cap_L \quad \Gamma_2 \longrightarrow G}{\Gamma_1, \Gamma_2, \Cap l_x : a.F \longrightarrow H \otimes G}\ \otimes_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1, F[l/l_x] \longrightarrow H \quad \Gamma_2 \longrightarrow G}{\Gamma_1, \Gamma_2, F[l/l_x] \longrightarrow H \otimes G}\ \otimes_R}{\Gamma_1, \Gamma_2, \Cap l_x : a.F \longrightarrow H \otimes G}\ \Cap_L$$

- $\exists_R$ permutes over $\Cap_L$:

$$\dfrac{\dfrac{\Gamma, F[l/l_x] \longrightarrow G[t/y]}{\Gamma, \Cap l_x : a.F \longrightarrow G[t/y]}\ \Cap_L}{\Gamma, \Cap l_x : a.F \longrightarrow \exists y.G}\ \exists_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, F[l/l_x] \longrightarrow G[t/y]}{\Gamma, F[l/l_x] \longrightarrow \exists y.G}\ \exists_R}{\Gamma, \Cap l_x : a.F \longrightarrow \exists y.G}\ \Cap_L$$

- $\Cup_R$ permutes over $\Cap_L$:

$$\dfrac{\dfrac{\Gamma, F[l_2/l_x] \longrightarrow G[l_1/l_y]}{\Gamma, \Cap l_x : a.F \longrightarrow G[l_1/l_y]}\ \Cap_L}{\Gamma, \Cap l_x : a.F \longrightarrow \Cup l_y : b.G}\ \Cup_R \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, F[l_2/l_x] \longrightarrow G[l_1/l_y]}{\Gamma, F[l_2/l_x] \longrightarrow \Cup l_y : b.G}\ \Cup_R}{\Gamma, \Cap l_x : a.F \longrightarrow \Cup l_y : b.G}\ \Cap_L$$

□

Theorem 5. *The sequent $\longrightarrow G$ is provable in SELL$^{\Cap}$ iff the sequent $[\mathcal{K} : \cdot], \cdot \longrightarrow G$ is also provable in SELLF$^{\Cap}$, where $\mathcal{K}[s] = \emptyset$ for all indices s.*

Proof. The proof follows the lines described in [24] and in [20], and the lemmas above are used to transform any arbitrary proof in SELL$^{\Cap}$ into a focused proof. In particular, Lemma 3 is used to permute negative rules downwards and Lemma 4 is used to organize the positive trunks in the resulting proof so that once a positive rule is applied then focusing is preserved on the resulting premises. □

5. CCP calculi

Concurrent Constraint Programming (CCP) [2, 3, 5] is a model for concurrency that combines the traditional operational view of process calculi with a *declarative* view based on logic. This allows CCP to benefit from the large set of reasoning techniques of both process calculi and logic.

Processes in CCP *interact* with each other by *telling* and *asking* constraints (pieces of information) in a common store of partial information. The type of constraints processes may act on is not fixed but parametric in a constraint system. Such systems can be formalized as a Scott information system [25] as in [5], or they can built upon a suitable fragment of logic *e.g.*, as in [26, 7, 27]. Here we specify constraints as formulas in intuitionistic first-order logic, namely in LJ [22].

Definition 1 (Constraint System [7])**.** *A constraint system is a tuple $(\mathcal{C}, \vdash_\Delta)$ where $\mathcal{C}$ is a set of formulas (constraints) built from a first-order signature and the grammar*

$$F \quad := \quad 1 \mid A \mid F \wedge F \mid \exists \overline{x}.F$$

where A is an atomic formula. We shall use c, c', d, d', etc, to denote elements of 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 *d', written as $d \vdash_\Delta d'$, iff the sequent $\Delta, d \longrightarrow d'$ is probable in LJ [22].*

As usual, $\exists \overline{x}(c)$ binds the variables $\overline{x}$ in c. We shall use $fv(c)$ to denote the set of free variables of c.

The language of determinate CCP processes [5] is 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. In other case, it remains blocked until more information is added to the store. This provides a powerful synchronization mechanism based on constraint entailment.

The process $(\mathbf{local}\ x)\ P$ behaves as P and binds the variable x to be local to it. We shall use $fv(P)$ to denote the set of free variables of P.

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

The operational semantics of CCP is given by the transition relation $\gamma \longrightarrow \gamma'$ satisfying the rules on Figure 4(a). These rules are straightforward realizing the operational intuitions given above. Moreover, they will form the core of transitions common to the other systems that we encode later.

Here we follow the operational semantics in [7] 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 $P_1 \parallel P_2 ... \parallel P_n$. We shall indistinguishably use both notations to denote parallel composition of processes.

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

1. $(\mathbf{local}\ x)\ P \cong (\mathbf{local}\ y)\ P[y/x]$ if $y \notin fv(P)$; – alpha conversion
2. $P \parallel Q \cong Q \parallel P$;
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.*, $c \vdash_\Delta c'$ and $c' \vdash_\Delta c$).

Let $\longrightarrow^*$ be the reflexive and transitive closure of $\longrightarrow$. If $(X; \Gamma; d) \longrightarrow^* (X'; \Gamma'; d')$ and $\exists X'.d' \vdash_\Delta c$ we write $(X; \Gamma; d) \Downarrow_c$. If $X = \emptyset$ and $d = 1$ we simply write $\Gamma \Downarrow_c$. Intuitively, if P is a process then $P \Downarrow_c$ says that P outputs c under input 1.

As processes manipulate the store of constraints, the constraint system (CS) used dictates much of the behavior of the system. Fages *et al.* in [7] showed that by using formulas in linear logic as CS, one obtains a more expressive language called Linear Concurrent Constraint (`lcc`) where ask processes can *consume* information (*i.e.*, constraints) from the store. In particular, one can specify when a constraint should be consumed or not by using or not a !. In fact, Fages *et al.* went even further and demonstrated that both, CCP and `lcc` processes can be characterized as intuitionistic linear

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

$$\frac{}{(X;\mathbf{tell}(c),\Gamma;d) \longrightarrow (X;\Gamma;c \wedge d)}\ \mathrm{R_T} \qquad \frac{d \vdash_\Delta c}{(X;\mathbf{ask}\ c\ \mathbf{then}\ P,\Gamma;d) \longrightarrow (X;P,\Gamma;d)}\ \mathrm{R_A}$$

$$\frac{x \notin X \cup fv(d) \cup fv(\Gamma)}{(X;(\mathbf{local}\ x)\ P,\Gamma;d) \longrightarrow (X \cup \{x\};P,\Gamma;d)}\ \mathrm{R_L} \quad \frac{p(\overline{x}) \stackrel{\Delta}{=} P}{(X;p(\overline{y}),\Gamma;d) \longrightarrow (X;P[\overline{y}/\overline{x}],\Gamma;d)}\ \mathrm{R_C}$$

(a) Operational rules for CCP.

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

(b) Operational rules for `eccp` and `sccp`

$$\frac{}{(X;\mathbf{always}\ P;\Gamma;d) \longrightarrow (X;P,\mathbf{next\ always}\ P;\Gamma;d)}\ \mathrm{R}_\Box \quad \frac{n \geq 0}{(X;\star P,\Gamma;d) \longrightarrow (X;\mathbf{next}^{\,n} P,\Gamma;d)}\ \mathrm{R}_\star$$

$$\frac{(\emptyset;P;c) \longrightarrow^* (X;\Gamma;d) \not\longrightarrow}{P \xRightarrow{(c,\exists X.d)} (\mathbf{local}\ X)\ F(R)}\ \mathrm{R_{Obs}}$$

(c) Operational rules for timed-ccp. **next** ^{n} means **next** ...**next** n-times. Function F is in Equation 4

Figure 4: Operational semantics for CCP calculi

logic (ILL) [8] formulas. That is, ILL also serves as a framework for specifying a wide range of concurrent systems. The same goal is achieved here, but by demonstrating that CCP with modalities can be encoded as SELLF$^{\Cap}$ formulas. However, as we explain later, our proofs have a stronger adequacy level than those achieved in [7]. Our adequacy is on the level of derivations [16], which means that proof search corresponds exactly to the execution of the encoded CCP specification.

Finally, although in this paper we do not encode CCP systems that use the linearity of formulas, it is straightforward to extend them to do so. In fact, we describe at the end of Section 6.1 how to encode `lcc` by simply changing the structural properties of subexponentials.

6. CCP as SELLF$^{\Cap}$ formulas

This section introduces an interpretation of the CCP language described above as formulas in SELLF$^{\Cap}$. This encoding will be used as basis in the subsequent sections to encode CCP calculi that include modalities.

For our encodings, we rely on the three following features of SELLF$^{\Cap}$. The first one is the subexponential quantifiers $\Cap$ and $\Cup$, which enable the specification of systems governing an unbounded number of modalities, *e.g.*, spaces or agents. In particular, by using these quantifiers, it is possible to specify that process definitions $p(\overline{x}) \stackrel{\Delta}{=} P$ are available to all entities in the system.

The second feature is the presence of non-equivalent subexponential prefixes (such as, *e.g.*, $!^s$ or $!^s?^s$) which will be written generically as ∇_s. This is the key for encoding correctly the different modalities, such as spatial, epistemic or temporal. Moreover, differently from the encoding of CCP in [7], both processes and constraints will be always marked with a subexponential bang, thus allowing for a better control on proofs.

Finally, the use of focusing provides better adequacy results while, at the same time, it will allow us to straightforwardly extend the encoding to include other modalities.

6.1. Basic Encoding

Assume a constraint system $(\mathcal{C}, \vdash_\Delta)$ and a set Ψ of process definitions of the form $p(\overline{y}) \stackrel{\Delta}{=} P$. For our encodings, we use a subexponential signature with three families and two distinguished elements, *nil* and ∞:

$$\Sigma = \langle I \cup \{nil, \infty\}, \preceq, \{\mathfrak{c}, \mathfrak{p}, \mathfrak{d}\}, U \rangle$$

In $\preceq$, ∞ is the greatest element, while *nil* is the least element. Moreover, $\mathfrak{c}(a) \in U$ for all $a \in I \cup \{nil, \infty\}$ and $\mathfrak{p}(\infty) \in U$, while $\mathfrak{p}(nil), \mathfrak{d}(nil) \notin U$.

Intuitively, the family $\mathfrak{c}$ is used to mark constraints; the family $\mathfrak{p}$ is used to mark processes; and the family $\mathfrak{d}$ is used to mark procedures $p(\overline{x})$ whose definition may be unfolded. As it will be clear later, the remaining subexponentials in I specify the modalities available in the system, where *nil* represents no modality. For instance, $\mathfrak{p}(nil)$ will mark a process that is not under any modality. Since process definitions, non-logical axioms and constraints can be used as many times, $\mathfrak{c}(a)$ and $\mathfrak{p}(\infty)$ are unbounded for any $a \in I$, while as processes and procedure calls are consumed when executed, $\mathfrak{p}(nil)$ and $\mathfrak{d}(nil)$ are bounded.

Encoding Constraints and Processes. Constraints and CCP processes are encoded into SELLF$^{\Cap}$ by using two functions: $\mathcal{P}[\![P]\!]_\ell$ for processes and $\mathcal{C}[\![c]\!]_\ell$ for constraints. These encodings will depend on the system that we want to encode and they are parametric on $\ell \in \mathcal{A}$. The definition below defines them for the set of basic processes and basic constraints shown in Section 5. Later, we will refine these encodings by adding new cases handling the specific constraints of each system.

Definition 2 (Encoding of Constraints and Processes)**.** *Let $\langle I \cup \{nil, \infty\}, \preceq, \{\mathfrak{c}, \mathfrak{p}, \mathfrak{d}\}, U \rangle$ be a subexponential signature, and let $\ell = (l : a) \in \mathcal{A}$. For any constraint c, $\mathcal{C}[\![c]\!]_\ell$ is defined as:*

- $\mathcal{C}[\![c_1 \wedge c_2]\!]_\ell = \mathcal{C}[\![c_1]\!]_\ell \otimes \mathcal{C}[\![c_2]\!]_\ell$
- $\mathcal{C}[\![\exists \overline{x}.c]\!]_\ell = \exists \overline{x}.\mathcal{C}[\![c]\!]_\ell$
- $\mathcal{C}[\![c]\!]_\ell = \bigtriangledown_{\mathfrak{c}(\ell)} c$ *if c is* 1 *or an atomic formula.*

For any process P, $\mathcal{P}[\![P]\!]_\ell$ is defined as:

- $\mathcal{P}[\![\mathbf{tell}(c)]\!]_\ell = !^{\mathfrak{p}(\ell)}[\Cap l_x : a.(\mathcal{C}[\![c]\!]_{(l_x:a)})]$
- $\mathcal{P}[\![\mathbf{ask}\ c\ \mathbf{then}\ P]\!]_\ell = !^{\mathfrak{p}(\ell)}[\Cap l_x : a.(\mathcal{C}[\![c]\!]_{(l_x:a)} \multimap \mathcal{P}[\![P]\!]_{(l_x:a)})]$
- $\mathcal{P}[\![(\mathbf{local}\ \overline{x})\ P]\!]_\ell = !^{\mathfrak{p}(\ell)}[\Cap l_x : a.\exists \overline{x}.(\mathcal{P}[\![P]\!]_{(l_x:a)})]$

- $\mathcal{P}[\![P_1, ..., P_n]\!]_\ell = \mathcal{P}[\![P_1]\!]_\ell \otimes ... \otimes \mathcal{P}[\![P_n]\!]_\ell$
- $\mathcal{P}[\![p(\overline{x})]\!]_\ell = \bigtriangledown_{\mathfrak{d}(\ell)} p(\overline{x})$.

Depending on the encoded system, we shall later instantiate $\bigtriangledown_s F$ as the formula $!^s F$ or $!^s ?^s F$.

Hence, atomic constraints are marked with subexponentials from the $\mathfrak{c}$ family, non-atomic processes with subexponentials from the $\mathfrak{p}$ family and procedures, $p(\overline{x})$, with subexponentials from the $\mathfrak{d}$ family. The role of the subexponential quantifiers in the encoding will become clear in the following sections. The idea is that they allow chosing in which modality a resulting process should be placed. This will be key for the encoding of epistemic CCP.

Notice that, by using simple logical equivalences (such as moving the existential outwards), we can rewrite the encoding of a constraint $\mathcal{C}[\![c]\!]_\ell$ so that it has the following shape:

$$\exists \overline{x}. \left[\bigtriangledown_{\mathfrak{c}(\ell_1)} A_1 \otimes \cdots \otimes \bigtriangledown_{\mathfrak{c}(\ell_n)} A_n \right] \tag{1}$$

where $A_1, \ldots, A_n$ are atomic formulas or the unit 1. The interesting bit here is that the store is specified by the atomic formulas it contains (A_i), marked with a subexponential prefix, $\bigtriangledown_{\mathfrak{c}(\ell_i)}$. Up to now, from Definition 2, we have a unique ℓ_i, namely $nil : nil$. The encodings of the CCP extensions will enable different subexponential indices to be used, illustrating the encoding's modularity. Moreover, by changing the definition of the pre-order of the subexponential signature, we will be able to specify different modalities in the system (see *e.g.*, Figure 5(a)).

Observe that the formula in Equation (1) is composed only by positive formulas. Thus, from the focusing discipline, whenever such a formula appears in the left-hand-side, it is completely decomposed as illustrated by the following derivation:

$$\dfrac{\dfrac{[\mathcal{K} : \Gamma], \Delta, \bigtriangledown_{\mathfrak{c}(\ell_1)} A_1, \ldots, \bigtriangledown_{\mathfrak{c}(\ell_n)} A_n \longrightarrow \mathcal{R}}{[\mathcal{K} : \Gamma], \Delta, \bigtriangledown_{\mathfrak{c}(\ell_1)} A_1 \otimes \cdots \otimes \bigtriangledown_{\mathfrak{c}(\ell_n)} A_n \longrightarrow \mathcal{R}}\ n-1 \times \otimes_L}{[\mathcal{K} : \Gamma], \Delta, \exists \overline{x}. \left[\bigtriangledown_{\mathfrak{c}(\ell_1)} A_1 \otimes \cdots \otimes \bigtriangledown_{\mathfrak{c}(\ell_n)} A_n\right] \longrightarrow \mathcal{R}}\ p \times \exists_L$$

Then the atomic formulas $A_1, \ldots, A_n$ appearing in the premise of this derivation are moved to the contexts $\mathfrak{c}(\ell_1), \ldots, \mathfrak{c}(\ell_n)$ of $\mathcal{K}$ respectively.

Encoding Non-Logical Axioms and Process Definitions. A non-logical axiom of the form $\forall \overline{x}(d \supset c)$ is encoded as:

$$\Cap l_x : \infty. \forall \overline{x}. (\mathcal{C}[\![d]\!]_{(l_x:\infty)} \multimap \mathcal{C}[\![c]\!]_{(l_x:\infty)}) \tag{2}$$

specifying that the non-logical axioms are available to all subexponentials in the ideal of ∞, *i.e.*, all elements in I. Similarly, a process definition of the form $p(\overline{x}) \stackrel{\Delta}{=} P$ is encoded as:

$$\Cap l_x : \infty. \forall \overline{x}. (\bigtriangledown_{\mathfrak{d}(l_x:\infty)} p(\overline{y}) \multimap \mathcal{P}[\![P]\!]_{(l_x:\infty)}) \tag{3}$$

We write $[\![\Delta]\!]$ and $[\![\Psi]\!]$ for the set of SELLF$^{\Cap}$ formulas encoding the non-logical axioms Δ and the process definitions Ψ.

Configurations as SELLF$^{\Cap}$ sequents. A CCP configuration $(X; \Gamma; c)$ is encoded as the SELLF$^{\Cap}$ sequent:

$$\mathcal{A}; \mathcal{L} \cup X; !^{\mathfrak{c}(\infty)}[\![\Delta]\!], !^{\mathfrak{p}(\infty)}[\![\Psi]\!], \mathcal{P}[\![\Gamma]\!]_{nil}, C[\![c]\!]_{nil} \longrightarrow G$$

The formula G on the right-hand side is the goal that we want to prove. Typically, it is the encoding of the constraint we are interested to know whether it can be outputted or not by the system. Finally, as normally done [28], the fresh values X are specified as eigenvariables in the logic.

The specification of processes, on the other hand, simply manipulates the set of constraints appearing on the left-hand side of sequents. For instance, the encoding of a **tell**(c) process adds the atomic constraints which compose c to the left-hand side of the sequent, as specified by the operational Rule R_T. Repeating this process we can prove the following adequacy theorem with respect to CCP. In fact, the adequacy we get is quite strong on the *level of derivations* [16], where proof search from the CCP process encoding corresponds exactly to the execution the CCP process.

Theorem 6. *Let P be a CCP process, $(C, \vdash_\Delta)$ be an CS, Ψ be a set of process definitions. Let $\bigtriangledown_\ell$ be instantiated to $!^\ell$. Then $P \Downarrow_c$ iff* $!^{\mathfrak{c}(\infty)}[\![\Delta]\!], !^{\mathfrak{p}(\infty)}[\![\Psi]\!], \mathcal{P}[\![P]\!]_{nil} \longrightarrow C[\![c]\!]_{nil} \otimes \top$ [4].

Proof. The proof relies on completeness of the focusing strategy (Theorem 5). We will keep the proofs general enough so that they can be easily adapted for the encoding of the CCP extensions. Recall that a configuration $(X; \Gamma; c)$ is encoded by a sequent of the form:

$$!^{\mathfrak{c}(\infty)}[\![\Delta]\!], !^{\mathfrak{p}(\infty)}[\![\Psi]\!], \mathcal{P}[\![\Gamma]\!]_{nil}, \bigtriangledown_{\mathfrak{c}(\ell_1)} A_1, \cdots, \bigtriangledown_{\mathfrak{c}(\ell_n)} A_n \longrightarrow G$$

This was shown by using using the fact that the left introduction rules of $\exists$ and $\otimes$ are negative. By using the same argument, $\mathcal{P}[\![\Gamma]\!]_{nil}$ reduces to

$$\mathcal{P}[\![P]\!]_{\ell_1}, \ldots, \mathcal{P}[\![P]\!]_{\ell_n}, !^{\mathfrak{d}(\ell'_1)} p_1(\overline{x}_1), \ldots, !^{\mathfrak{d}(\ell'_m)} p_m(\overline{x}_m).$$

So in fact, we can re-write the sequent above as follows

$$[\mathcal{C}, \mathcal{D}, \mathcal{P}] \longrightarrow [G]$$

where the context $\mathcal{K}$ is split into three contexts: $\mathcal{C}, \mathcal{D}$ and $\mathcal{P}$, containing all formulas marked, respectively, with bangs of the $\mathfrak{c}$, $\mathfrak{d}$ and $\mathfrak{p}$ families. For example, if $\mathcal{C}[\mathfrak{c}(\ell)] = \{F, G\}$, then the formulas $!^{\mathfrak{c}(\ell)} F$ and $!^{\mathfrak{c}(\ell)} G$ are in the context. Moreover, notice that the context C only contains subexponentials from the $\mathfrak{c}$ family, that is, $\mathcal{C}[\mathfrak{p}(\ell)] = \mathcal{C}[\mathfrak{d}(\ell)] = \emptyset$ for any $\ell \in \mathcal{A}$. Similarly, the context $\mathcal{D}$ contains only atomic procedure calls of the form $p(\overline{x})$ and $\mathcal{P}$ contains the encoding of processes $\mathcal{P}[\![P]\!]_\ell$.

We now show that the introduction of any formula following the focused discipline corresponds exactly to applying one rule in CCP's operational semantics.

[4]With the $\top$ unit on the righthand side of the sequent we capture the observables of a process regardless whether the final configuration has suspended asks processes.

For simplicity, all over the proof we will assume that

$$\mathcal{C}[\![c]\!]_{\ell'} = \exists\overline{x}.\left[!^{\mathfrak{c}(s_1)}[?^{\mathfrak{c}(s_1)}]A_1 \otimes \cdots \otimes !^{\mathfrak{c}(s_n)}[?^{\mathfrak{c}(s_n)}]A_n\right]$$

where the connectives $[?^{\mathfrak{c}(s_i)}]$ for $1 \leq i \leq n$ may appear or not, depending on the instantiation of $\bigtriangledown$.

We will also represent $\mathcal{P}[\![P]\!]_{\ell'} = !^{\mathfrak{p}(\ell')}F$ as the encoding of the process P.

- Case **tell**(c). Suppose $\mathcal{P}[\![\mathbf{tell}(c)]\!]_\ell = !^{\mathfrak{p}(\ell)}\Cap l_x : a.\mathcal{C}[\![c]\!]_{(l_x:a)}$ is in the context.

The focused derivation obtained by focusing on this formula is necessarily as follows:

$$\dfrac{\dfrac{\dfrac{[\mathcal{C}',\mathcal{D},\mathcal{P}'] \longrightarrow [G]}{[\mathcal{C},\mathcal{D},\mathcal{P}'], \exists\overline{x}.\left[!^{\mathfrak{c}(s_1)}[?^{\mathfrak{c}(s_1)}]A_1 \otimes \cdots \otimes !^{\mathfrak{c}(s_n)}[?^{\mathfrak{c}(s_n)}]A_n\right] \longrightarrow [G]}\ j\times\exists_L, n\times\otimes_L, n\times !_L}{\dfrac{[\mathcal{C},\mathcal{D},\mathcal{P}'], \mathcal{C}[\![c]\!]_{\ell'} \longrightarrow [G]}{[\mathcal{C},\mathcal{D},\mathcal{P}'] \xrightarrow{\Cap l_x:a.\mathcal{C}[\![c]\!]_{(l_x:a)}} [G]}\ \Cap_L, R_L}}{[\mathcal{C},\mathcal{D},\mathcal{P} +_{\mathfrak{p}(\ell)} \Cap l_x : a.\mathcal{C}[\![c]\!]_{(l_x:a)}] \longrightarrow [G]}\ D$$

where $\mathcal{P}' = \mathcal{P}$ if the subexponential $\mathfrak{p}(\ell)$ is linear (as in the case of CCP) and $\mathcal{P}' = \mathcal{P} +_{\mathfrak{p}(\ell)} \mathcal{P}[\![\mathbf{tell}(c)]\!]_\ell$ if the subexponential $\mathfrak{p}(\ell)$ is unbounded (as in the case of `eccp`).

Hence, from bottom-up, the focused derivation above of the encoding of **tell**(c) corresponds exactly to its CCP execution: the constraint c is decomposed into its atomic parts and then added to the store.

- Case **ask**(c). Suppose $\mathcal{P}[\![\mathbf{ask}\ c\ \mathbf{then}\ P]\!]_\ell = !^{\mathfrak{p}(\ell)}\Cap l_x : a.(\mathcal{C}[\![c]\!]_{(l_x:a)} \multimap \mathcal{P}[\![P]\!]_{(l_x:a)})$ is in the context. Focusing on this formula results necessarily in the following focused derivation:

$$\dfrac{\dfrac{\dfrac{\pi_1}{[\mathcal{C},\mathcal{D},\mathcal{P}']-_{\mathcal{C}[\![c]\!]_{\ell'}}\!\rightarrow} \quad \dfrac{[\mathcal{C},\mathcal{D},\mathcal{P}' +_{\mathfrak{p}(\ell')} F] \longrightarrow [G]}{[\mathcal{C},\mathcal{D},\mathcal{P}'] \xrightarrow{\mathcal{P}[\![P]\!]_{\ell'}} [G]}\ R_L, !^{\mathfrak{p}(\ell')}{}_L}{[\mathcal{C},\mathcal{D},\mathcal{P}'] \xrightarrow{\Cap l_x:a(\mathcal{C}[\![c]\!]_{(l_x:a)} \multimap \mathcal{P}[\![P]\!]_{(l_x:a)})} [G]}\ \Cap_L, \multimap_L}{[\mathcal{C},\mathcal{D},\mathcal{P} +_{\mathfrak{p}(\ell)} \Cap l_x : a(\mathcal{C}[\![c]\!]_{(l_x:a)} \multimap \mathcal{P}[\![P]\!]_{(l_x:a)})] \longrightarrow [G]}\ D$$

where $\mathcal{P}'$ is as in the previous case. Moreover, since $\mathcal{C}[\![c]\!]_{\ell'}$ contains only positive formulas, it will be totally decomposed resulting on a positive trunk with sequents of the form $[\mathcal{C},\mathcal{D},\mathcal{P}]-_{\bigtriangledown_{\mathfrak{c}(\ell_i)}A}\!\rightarrow$. Hence the sequents obtained in π_1 will necessarily end with derivations of the form:

$$\dfrac{\dfrac{\pi_2}{[\mathcal{C}\leq_{\mathfrak{c}(\ell_i)}] \longrightarrow [?^{\mathfrak{c}(\ell_i)}]A}}{[\mathcal{C},\mathcal{D},\mathcal{P}']-_{!^{\mathfrak{c}(\ell_i)}[?^{\mathfrak{c}(\ell_i)}]A}\!\rightarrow}\ !^{\mathfrak{c}(\ell_i)}{}_r$$

The important thing to notice is that the contexts $\mathcal{D}$ and $\mathcal{P}$ are necessarily weakened in the premise or its elements are moved to the right-premise. This is due to the fact that, for any ℓ_1, ℓ_2, ℓ_3, $\mathfrak{c}(\ell_1)$ is not related to $\mathfrak{p}(\ell_2)$ or $\mathfrak{d}(\ell_3)$. Hence, as A is atomic, it should be provable from the atomic formulas C_{atom} in C and the theory Δ. That is, $C_{atom} \vdash_\Delta A$. Finally, observe that formulas in C_{atom} are constraints, coming from *tells*, as described in the previous case. Thus, from bottom-up the derivation above corresponds exactly to the operational semantics of **ask** c **then** P, where c is deduced from the store and only then P can be executed.

Notice that $[C_{\leq \mathfrak{c}(\ell)}] \longrightarrow [?^{\mathfrak{c}(\ell)}]A$ is provable only from the the context $C_{\leq \mathfrak{c}(\ell)}$ containing all the subexponentials in family $\mathfrak{c}$ that are greater than $\mathfrak{c}(\ell)$. For the epistemic and spatial systems this will amount to proving A from the knowledge of information stored in a particular modality.

- Case **local**(c). Suppose $\mathcal{P}[\![(\textbf{local}\, y)\, P]\!]_\ell = \,!^{\mathfrak{p}(\ell)}(\Cap l_x : a.\exists y.(\mathcal{P}[\![P]\!]_{(l_x:a)}))$ is in the context:

$$\dfrac{\dfrac{\dfrac{[C, \mathcal{D}, \mathcal{P}' +_{\mathfrak{p}(\ell')} (F[z/y])] \longrightarrow [G]}{[C, \mathcal{D}, \mathcal{P}'], \exists y.\mathcal{P}[\![P]\!]_{\ell'} \longrightarrow [G]}\; n \times \exists_L, !^{\mathfrak{p}(\ell')}{}_L}{[C, \mathcal{D}, \mathcal{P}'] \xrightarrow{(\Cap l_x : a.\exists y.(\mathcal{P}[\![P]\!]_{(l_x:a)}))} [G]}\; \Cap_L, R_L}{[C, \mathcal{D}, \mathcal{P}' +_{\mathfrak{p}(\ell)} (\Cap l_x : a.\exists y.(\mathcal{P}[\![P]\!]_{(l_x:a)}))] \longrightarrow [G]}\; D$$

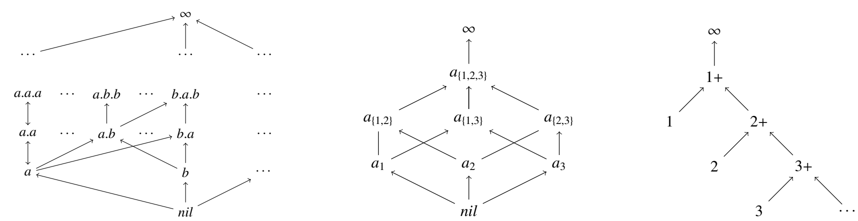
Thus, this derivation corresponds exactly to the operational semantics of $(\textbf{local}\, y)\, P$, where $P[z/y]$ can be executed for a fresh variable z.

- Case recursive calls. Focusing on the formula $\Cap l_x : \infty.\forall \overline{y}.(\bigtriangledown_{\mathfrak{d}(l_x:\infty)}\, p(\overline{y}) \multimap \mathcal{P}[\![P]\!]_{(l_x:\infty)})$ will give rise to the derivation below. Again let $\bigtriangledown_{\mathfrak{d}(\ell)}$ be of the form $!^{\mathfrak{d}(\ell)}[?^{\mathfrak{d}(\ell)}]p(\overline{x})$, where $[?^{\mathfrak{d}(\ell)}]$ may appear or not, depending on the instantiation of $\bigtriangledown_{\mathfrak{d}(\ell)}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[\mathcal{D}] \longrightarrow [?^{\mathfrak{d}(\ell')}]p(\overline{y})}}{[C, \mathcal{D}, \mathcal{P}'] \xrightarrow{!^{\mathfrak{d}(\ell')}[?^{\mathfrak{d}(\ell')}]p(\overline{y})}}\; !^{\mathfrak{d}(\ell')}{}_L \qquad \dfrac{[C, \mathcal{D}, \mathcal{P}' +_{\mathfrak{p}(\ell')} F] \longrightarrow [G]}{[C, \mathcal{D}, \mathcal{P}'] \xrightarrow{(\mathcal{P}[\![P]\!]_{\ell'})} [G]}\; R_L, !^{\mathfrak{p}(\ell')}{}_L}{[C, \mathcal{D}, \mathcal{P}'] \xrightarrow{\Cap l_x:\infty.\forall \overline{y}.(\bigtriangledown_{\mathfrak{d}(l_x:\infty)}\, p(\overline{y}) \multimap \mathcal{P}[\![P]\!]_{(l_x:\infty)})} [G]}\; \Cap_L, \forall_L, \multimap_L}{[C, \mathcal{D}, \mathcal{P} +_{\mathfrak{p}(\infty)} \Cap l_x : \infty.\forall \overline{y}.(\bigtriangledown_{\mathfrak{d}(l_x:\infty)}\, p(\overline{y}) \multimap \mathcal{P}[\![P]\!]_{(l_x:\infty)})] \longrightarrow [G]}\; D$$

Note that, as in the case for the asks processes, the contexts $\mathcal{P}$ and C are weakened in the rule $!^{\mathfrak{d}(\ell')}{}_L$ or necessarily moved to the right-premise because $\mathfrak{d}(\ell')$ is not related to $\mathfrak{p}(\ell_1)$ or $\mathfrak{c}(\ell_2)$ for any ℓ', ℓ_1, ℓ_2. Hence, since $p(\overline{x})$ is atomic, it should be provable from the formulas in $\mathcal{D}$. But all the formulas in $\mathcal{D}$ are atomic, so it should be the case that $p(\overline{x}) \in \mathcal{D}$ and hence the derivation on the right ends with an initial axiom. Thus, from bottom-up the derivation above corresponds exactly to the operational semantics of process calls, where $p(\overline{x})$ is substituted by its defined process P. □

We note that it is straightforward to modify the encodings in Definition 2 in order to encode `lcc`: simply declare $\mathfrak{c}(nil) \notin U$ (*i.e.*, constraints can be consumed) and $\mathfrak{c}(\infty) \in U$, then extend the encoding for the case of unbounded constraints: $C[\![!\, c]\!]_\ell = C[\![c]\!]_\infty$. The adequacy theorem obtained follows similarly from the one stated here, but the use of focusing gives a stronger adequacy result than the one stated in [7].



(a) Exponentials for epistemic reasoning.

(b) Complementation to represent common knowledge among agents a_1, a_2, a_3.

(c) Subexponential signature to specify temporal modalities.

Figure 5: Examples of subexponential signature for epistemic and time reasoning. Here $a \to b$ denotes that $a \leq b$.

7. Epistemic CCP

Knight *et al.* in [10] proposed Epistemic CCP (`eccp`), a CCP-based language where systems of agents are considered for distributed and epistemic reasoning. In `eccp`, the constraint system, seen as an Scott information system as in [5], is extended in order to consider space of agents. In a nutshell, each agent a has a space s_a and $s_a(c)$ means "c holds in the space –store– of agent a."

The following definition gives an instantiation of an epistemic constraint system where basic constraints are built as in Definition 1.

Definition 3 (Epistemic Constraint System (ECS))**.** *Let A be a countable set of agent names. An ECS $(C_e, \vdash_{\Delta_e})$ is a CS where, for any $a \in A$, $s_a : C_e \longrightarrow C_e$ is a mapping satisfying:*

1. $s_a(1) = 1$ *(bottom preserving)*
2. $s_a(c \wedge d) = s_a(c) \wedge s_a(d)$ *(lub preserving)*

Moreover, s_i is a closure operator, i.e., it satisfies:

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)*

The language of CCP processes is extended in `eccp` 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(b). 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 encodings shown in Section 6 so to encode epistemic modalities, starting by the subexponential signature that we use. Let $A = \{a_1, a_2, ...\}$ be a possible infinite set of agents and let A^* be the set of non-empty strings of elements in A; for example, if $a, b \in A$, then $a, b, a.a, b.a, a.b.a, \ldots \in A^*$. We shall use $\overline{a}, \overline{b}, etc$ to denote elements in A^*. We shall also consider *nil* to be the empty string, thus the string $\overline{a}.nil.\overline{b}$ is written as $\overline{a}.\overline{b}$. We define

$$I = A^* \cup \{nil, \infty\}$$
$$U = \{\mathfrak{c}(\overline{a}), \mathfrak{d}(\overline{a}), \mathfrak{p}(\overline{a}) \mid \overline{a} \in I\} \setminus \{\mathfrak{d}(nil), \mathfrak{p}(nil)\}$$

Intuitively, the connective $!^{\mathfrak{p}(1.2.3)}$, for example, 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 $s_1(s_2(s_3(\cdot)))$. Notice that all $\mathfrak{p}()$ and $\mathfrak{d}()$ subexponential indices, except the ones constructed using *nil*, are unbounded. This reflects the fact that both constraints and processes in the space of an agent are unbounded, as specified by rule R_E.

The pre-order $\preceq$ is as depicted in Figure 5(a). More precisely, we construct the pre-order inductively as follows: for every two different agent names a and b in A, the subexponentials a and b are unrelated; moreover, two sequences in A^* are related $a_1.a_2.\cdots.a_m \preceq b_1.b_2.\cdots.b_n$ whenever for any formula F and family $\mathfrak{f} \in \{\mathfrak{c}, \mathfrak{p}, \mathfrak{d}\}$ the sequent

$$!^{\mathfrak{f}(b_1)}!^{\mathfrak{f}(b_2)}\ldots!^{\mathfrak{f}(b_n)}F \longrightarrow !^{\mathfrak{f}(a_1)}!^{\mathfrak{f}(a_2)}\ldots!^{\mathfrak{f}(a_n)}F$$

is provable.

An alternative way of defining the pre-order on sequences of agent names is the following: $a \approx a.a.\ldots.a$ and $b_1.b_2.\ldots.b_n \preceq \overline{a}_1.b_1.\overline{a}_2.b_2.\ldots.\overline{a}_n.b_n$ where each $\overline{a}_i$ is a possible empty string of elements in A.

The shape of the pre-order is key for our encoding. In particular, we are using one subexponential index $\mathfrak{f}(a_1.a_2.\ldots.a_n)$, to denote a prefix of subexponential bangs $!^{\mathfrak{f}(a_1)}!^{\mathfrak{f}(a_2)}\ldots!^{\mathfrak{f}(a_n)}$. Thus, if two subexponentials $\overline{a}, \overline{a}'$ are equal in the pre-order, denoted as $\overline{a} \approx \overline{a}'$, it means that they represent the same equivalence class of prefixes. This way, we are able to quantify over such prefixes (or boxes) by using a single quantifier, as done for the encoding of the non-logical axioms and procedure calls.

Definition 4 (Epistemic constraints and processes)**.** *Let $\overline{\ell} = (\bar{l} : \overline{a})$ and $\overline{\ell}.i = (\bar{l}.i : \overline{a}.i)$, $\overline{\ell}, \overline{\ell}.i \in \mathcal{A}$. We extend $\mathcal{C}[\![\cdot]\!]_\ell$ in Definition 2 so that $\mathcal{C}[\![s_i(c)]\!]_{\overline{\ell}} = \mathcal{C}[\![c]\!]_{\overline{\ell}.i}$ and $\bigtriangledown_{\mathfrak{f}(\overline{\ell})}$ is instantiated as $!^{\mathfrak{f}(\overline{\ell})}$. Moreover, we extend $\mathcal{P}[\![\cdot]\!]_\ell$ in Definition 2 so that $\mathcal{P}[\![[P]_i]\!]_{\overline{\ell}} = \mathcal{P}[\![P]\!]_{\overline{\ell}.i}$.*

Observe that, in $\mathcal{P}[\![P]\!]_{\bar{l}:\overline{a}}$, $\overline{a}$ is the space-location where P is executed. The role of the quantifier subexponentials in encoding of processes in Definition 2 is key. For instance, recall that the encoding $\mathcal{P}[\![\textbf{ask } c \textbf{ then } P]\!]_{\overline{\ell}}$ is

$$!^{\mathfrak{p}(\overline{\ell})}[\Cap l_x : \overline{a}.(\mathcal{C}[\![c]\!]_{(l_x:\overline{a})} \multimap \mathcal{P}[\![P]\!]_{(l_x:\overline{a})})]$$

Here $!^{\mathfrak{p}(\overline{\ell})}$ specifies the epistemic state $[]_{\overline{a}}$ where the process is. On the other hand, the subexponential quantification $(\Cap l_x : \overline{a})$ specifies that one can move the process anywhere in the ideal of $\overline{a}$. From the pre-order shown in Figure 5(a), this means moving the process to anywhere outside the box $[]_{\overline{a}}$. This corresponds exactly to the Rule R_E. Moreover, since $\mathfrak{p}(\overline{\ell}) \in U$, the process is unbounded, thus the encoding $\mathcal{P}\llbracket \textbf{ask } c \textbf{ then } P\rrbracket_{\overline{\ell}}$ is not consumed.

This intuition is formalized by the following result: any process, $\mathcal{P}\llbracket P\rrbracket_{\overline{\ell}.i}$, can move to an outer box $\mathcal{P}\llbracket P\rrbracket_{\overline{\ell}}$.

Proposition 1. *Let $\mathcal{P}\llbracket\cdot\rrbracket_{\overline{\ell}}$ be as in Definition 4. The sequent $\mathcal{P}\llbracket P\rrbracket_{\overline{\ell}.i} \longrightarrow \mathcal{P}\llbracket P\rrbracket_{\overline{\ell}}$ is provable in SELL$^{\Cap}$ for any process P and subexponentials $\overline{\ell}$ and i.*

Proof. The proof is on induction on the size of P. The only interesting case is for **ask** c **then** P, shown below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C\llbracket c\rrbracket_{(l_e:\overline{a})} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_e:\overline{a})} \longrightarrow C\llbracket c\rrbracket_{(l_e:\overline{a})} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_e:\overline{a})}}\; I_R}{\Cap l_x : \overline{a}.i.C\llbracket c\rrbracket_{(l_x:\overline{a}.i)} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_x:\overline{a}.i)} \longrightarrow C\llbracket c\rrbracket_{(l_e:\overline{a})} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_e:\overline{a})}}\; \Cap_L}{\Cap l_x : \overline{a}.i.C\llbracket c\rrbracket_{(l_x:\overline{a}.i)} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_x:\overline{a}.i)} \longrightarrow \Cap l_y : \overline{a}.C\llbracket c\rrbracket_{(l_y:\overline{a})} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_y:\overline{a})}}\; \Cap_R}{!^{\mathfrak{p}(\overline{\ell}.i)}\Cap l_x : \overline{a}.i.C\llbracket c\rrbracket_{(l_x:\overline{a}.i)} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_x:\overline{a}.i)} \longrightarrow \Cap l_y : \overline{a}.C\llbracket c\rrbracket_{(l_y:\overline{a})} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_y:\overline{a})}}\; !^{\mathfrak{p}(\overline{\ell}.i)}{}_L}{!^{\mathfrak{p}(\overline{\ell}.i)}\Cap l_x : \overline{a}.i.C\llbracket c\rrbracket_{(l_x:\overline{a}.i)} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_x:\overline{a}.i)} \longrightarrow !^{\mathfrak{p}(\overline{\ell})}\Cap l_y : \overline{a}.C\llbracket c\rrbracket_{(l_y:\overline{a})} \multimap \mathcal{P}\llbracket P\rrbracket_{(l_y:\overline{a})}}\; !^{\mathfrak{p}(\overline{\ell})}{}_R$$

□

Observe that the fresh variable l_e introduced in $\Cap_R$ has type $\overline{a}$, which is in the ideal of $\overline{a}.i$. Hence, we may instantiate the variable $l_x : \overline{a}.i$ as $l_e : \overline{a}$ on applying the rule $\Cap_L$.

The following proposition shows that the proposed translation of constraints to formulas in SELL$^{\Cap}$ represents, indeed, an epistemic constraint system. The proof is immediate from the definition of $C\llbracket\cdot\rrbracket_{\overline{\ell}}$.

Proposition 2. *Let $(C_e, \vdash_{\Delta_e})$ be an ECS and $C\llbracket\cdot\rrbracket_{\overline{\ell}}$ be as in Definition 4. Then, for any $\overline{\ell}$:*

1. $C\llbracket 1\rrbracket_{\overline{\ell}} \equiv 1$ *(bottom preserving);*
2. $C\llbracket c \wedge d\rrbracket_{\overline{\ell}} \equiv C\llbracket c\rrbracket_{\overline{\ell}} \otimes C\llbracket d\rrbracket_{\overline{\ell}}$ *(lub preserving);*
3. *If* $d \vdash_{\Delta_e} c$ *then* $!^{\mathfrak{c}(\infty)}\llbracket\Delta_e\rrbracket, C\llbracket d\rrbracket_{\overline{\ell}} \longrightarrow C\llbracket c\rrbracket_{\overline{\ell}}$ *(monotonicity);*
4. $C\llbracket s_i(c)\rrbracket_{\overline{\ell}} \longrightarrow C\llbracket c\rrbracket_{nil}$ *(believes are facts);*
5. $C\llbracket s_i(s_i(c))\rrbracket_{\overline{\ell}} \equiv C\llbracket s_i(c)\rrbracket_{\overline{\ell}}$ *(idempotence).*

Example 1 (Epistemic Reasoning)**.** *Let $P = \textbf{tell}(c)$, $Q = \textbf{ask } c \textbf{ then tell}(d)$ and $R = [P \parallel [Q]_b]_a$. The following sequent is provable $\mathcal{P}\llbracket R\rrbracket_{nil} \longrightarrow !^{\mathfrak{c}(a)}c \otimes !^{\mathfrak{c}(nil)}c \otimes \top$. That is, c is* known *by agent a and the external environment (i.e., c is a* fact*). Also, $\mathcal{P}\llbracket R\rrbracket_{nil} \longrightarrow$ $!^{\mathfrak{c}(a)}d \otimes \top$ since Q also runs in the scope of a. This intuitively means that a knows that b knows that if c is true, then d is true. Thus, a knows c and d. Furthermore, the agent b does not know c, i.e., the sequent $\mathcal{P}\llbracket R\rrbracket_{nil} \longrightarrow !^{\mathfrak{c}(b)}c \otimes \top$* is not *provable.*

Now are are ready to state the main result of this section.

Theorem 7 (Adequacy)**.** *Let P be an* `eccp` *process, $(C_e, \vdash_e)$ be an ECS, Ψ be a set of process definitions and let $C\llbracket\cdot\rrbracket_{\overline{\ell}}$ and $\mathcal{P}\llbracket\cdot\rrbracket_{\overline{\ell}}$ be as in Definition 4. Then $P \Downarrow_c$ iff* $!^{\mathfrak{c}(\infty)}\llbracket\Delta_e\rrbracket, !^{\mathfrak{p}(\infty)}\llbracket\Psi\rrbracket, \mathcal{P}\llbracket P\rrbracket_{nil} \longrightarrow C\llbracket c\rrbracket_{nil} \otimes \top$.

Proof. The proof follows the lines of the proof of Theorem 6, only that now processes are unbounded and they can be moved outside boxes. For instance, focusing on the encoding of a process **tell**(c) we obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{[C', \mathcal{D}, \mathcal{P} +_{\mathfrak{p}(\bar{\ell})} \Cap l_x : \overline{a}.C[\![c]\!]_{(l_x:\overline{a})}] \longrightarrow [G]}{[C, \mathcal{D}, \mathcal{P} +_{\mathfrak{p}(\bar{\ell})} \Cap l_x : \overline{a}.C[\![c]\!]_{(l_x:\overline{a})}], C[\![c]\!]_{\bar{\ell}'} \longrightarrow [G]}}{[C, \mathcal{D}, \mathcal{P} +_{\mathfrak{p}(\bar{\ell})} \Cap l_x : \overline{a}.C[\![c]\!]_{(l_x:\overline{a})}] \xrightarrow{\Cap l_x:\overline{a}.C[\![c]\!]_{(l_x:\overline{a})}} [G]} \; \Cap_L, R_l}{[C, \mathcal{D}, \mathcal{P} +_{\mathfrak{p}(\bar{\ell})} \Cap l_x : \overline{a}.C[\![c]\!]_{(l_x:\overline{a})}] \longrightarrow [G]} \; D$$

Notice that differently from before, the process definition is unbounded. Thus the formula $\Cap l_x : \overline{a}.C[\![c]\!]_{(l_x:\overline{a})}$ is contracted. Moreover, there is choice of where to place the result of executing the process. In fact, $l_x : \overline{a}$ will be instantiated as $\overline{\ell}' = (l' : a')$ with a' in the ideal of $\overline{a}$, that is, l may be anywhere outside the box represented by $\overline{a}$. □

This result, besides giving an interesting interpretation of subexponentials as knowledge-spaces, gives a proof system for the verification of `eccp` processes.

So far, we have assumed that knowledge is not shared by agents. Next example shows how to handle common knowledge among agents. The approach is similar to the one given in [10], introducing *announcements* of constraints among group of agents, but by using our proof theoretic framework.

Example 2 (Common Knowledge)**.** *Assume a finite set of agents $A = \{a_1, ..., a_n\}$ and a definition of the form:*

$$global_P() \stackrel{\texttt{def}}{=} P \parallel [P \parallel global_P()]_{a_1} \parallel ... \parallel [P \parallel global_P()]_{a_n}$$

For example, the process $global_{\mathbf{tell}(c)}$ makes c available in all spaces and nested spaces involving agents in A. Instead of computing common knowledge by recursion, we can complement the subexponential signature as in Figure 5(b) where for all $\mathcal{S} \subseteq A$, $\overline{a} \preceq a_{\mathcal{S}}$ 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 $!^{\mathfrak{c}(a_{\mathcal{S}})}c \longrightarrow\, !^{\mathfrak{c}(\overline{a})}c \otimes \top$ 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.

8. Spatial CCP

Inconsistent information in CCP arises when considering theories containing axioms such as $c \wedge d \vdash_\Delta 0$. Notice that agents are not allowed to tell or ask false, *i.e.*, 0 is not a constraint. 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 [10], 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(b). Furthermore, some additional requirements are imposed on the representation of agents' spaces $s_a(\cdot)$.

Definition 5 (Spatial Constraint System (SCS))**.** *Let A be a countable set of agent names. An SCS $(\mathcal{C}_s, \vdash_{\Delta_s})$ is a CS where, for any $a \in A$, $s_a : \mathcal{C}_s \longrightarrow \mathcal{C}_s$ is a mapping satisfying bottom preserving, lub preserving, monotonicity and* false containment *(see Proposition 3 below).*

The subexponentials $I = A^* \cup \{nil, \infty\}$ are the same as in the encoding of the epistemic case but the pre-order is much simpler: for any $\overline{a} \in A^*$, $\overline{a} \preceq \infty$. That is, two different elements of A^* are unrelated. However, since `sccp` does not contain the $\mathrm{R_E}$ rule, processes in spaces are again treated linearly. Thus: $U = \{\mathfrak{c}(a) \mid a \in I\} \cup \{\mathfrak{p}(\infty)\}$.

The confinement of spatial information is captured by a different subexponential prefix $\bigtriangledown$ as follows.

Definition 6 (Spatial constraints in SELL$^{\Cap}$)**.** *The encodings $\mathcal{C}[\![\cdot]\!]_{\overline{\ell}}$ and $\mathcal{P}[\![\cdot]\!]_{\overline{\ell}}$ are as in Definition 4. In both cases, however, $\bigtriangledown_{\mathfrak{f}(\ell)}$ is instantiated as $!^{\mathfrak{f}(\ell)}?^{\mathfrak{f}(\ell)}$.*

Differently from the epistemic case, the encoding of $[P]_a$ runs P only the space of a and not outside it. This is captured by the pre-order above and by instantiating $\bigtriangledown_{\mathfrak{f}(\ell)}$ as $!^{\mathfrak{f}(\ell)}?^{\mathfrak{f}(\ell)}$. Notice that the ideal of any subexponenial $a \in I \setminus \{\infty\}$ is the singleton $\{a\}$. This means that the only way of instantiating, with a member of I, the subexponential quantifier $\Cap l_x : a$ in the encoding of processes is with the constant a itself. Hence, we are able to confine the information inside the location of agents as states the following proposition.

Proposition 3 (False confinement)**.** *Let $(\mathcal{C}_s, \vdash_{\Delta_s})$ be a SCS and $\mathcal{C}[\![\cdot]\!]_{\overline{\ell}}$ as in Definition 6. Then, monotonicity, bottom and lub preserving items in Proposition 2 hold. Furthermore, for any $\overline{a} \in A^*$, if we assume that $c \wedge d \vdash_{\Delta_s} 0$:*

1. $\mathcal{C}[\![0]\!]_{\overline{\ell}} \longrightarrow \mathcal{C}[\![c]\!]_{\overline{\ell}}$ *(any c can be deduced in the space $\overline{\ell}$ if its local store is inconsistent);*
2. $\mathcal{C}[\![0]\!]_{\overline{\ell}} \longrightarrow \mathcal{C}[\![0]\!]_{\overline{\ell}'}$ *is not provable (*false *is confined);*
3. $!^{\mathfrak{c}(\infty)}\mathcal{C}[\![\Delta_s]\!], \mathcal{C}[\![c]\!]_{\overline{\ell}}, \mathcal{C}[\![d]\!]_{\overline{\ell}} \longrightarrow \mathcal{C}[\![0]\!]_{\overline{\ell}}$ *(if space $\overline{\ell}$ contains both c and d, then it becomes inconsistent);*
4. $!^{\mathfrak{c}(\infty)}\mathcal{C}[\![\Delta_s]\!], \mathcal{C}[\![c]\!]_{\overline{\ell}}, \mathcal{C}[\![d]\!]_{\overline{\ell}'} \longrightarrow \mathcal{C}[\![0]\!]_{\overline{\ell}}$ *is not provable if $\overline{\ell}' \neq \overline{\ell}$ (*false *is not deduced if c and d are in different spaces);*
5. $\mathcal{C}[\![c]\!]_{\overline{\ell}} \longrightarrow c$ *and* $\mathcal{C}[\![c]\!]_{\overline{\ell}} \longrightarrow \mathcal{C}[\![c]\!]_{nil}$ *are both not provable (local information is not global).*

Proof. The proof follows trivially from the definition of $\mathcal{C}[\![\cdot]\!]_{\overline{\ell}}$. Note that if $s \neq s'$, the sequent $!^{s}?^{s}F \longrightarrow !^{s'}?^{s'}F$ is not provable. □

Example 3 (Local stores)**.** *Let $P = \mathbf{tell}(c)$ and $Q = \mathbf{ask}\ c\ \mathbf{then}\ \mathbf{tell}(d)$. Let $R = [P]_a \parallel [Q]_b$. Observe that Q remains blocked since the information c is only available on the space of a. In our encoding, as $!^{\mathfrak{c}(a)}?^{\mathfrak{c}(a)}c \longrightarrow !^{\mathfrak{c}(b)}?^{\mathfrak{c}(b)}c$ is not provable, the sequent $\mathcal{P}[\![R]\!]_{nil} \longrightarrow !^{\mathfrak{c}(b)}?^{\mathfrak{c}(b)}d \otimes \top$ is also not provable. Now let $R = [P]_a \parallel [Q]_a$. The process P adds d in the space of a and then, Q can evolve. Thus, $\mathcal{P}[\![R]\!]_{nil} \longrightarrow !^{\mathfrak{c}(a)}?^{\mathfrak{c}(a)}d \otimes \top$ is provable. Moreover, c does not propagate outside the scope of agent a, i.e., the sequent $\mathcal{P}[\![R]\!]_{nil} \longrightarrow !^{\mathfrak{c}(nil)}?^{\mathfrak{c}(nil)}c \otimes \top$ is not provable. Finally, consider $R = [[P]_a]_b \parallel [Q]_a$. Since $a \not\preceq b.a$ and $b.a \not\preceq a$, the sequent $!^{\mathfrak{c}(b.a)}?^{\mathfrak{c}(b.a)}c \longrightarrow !^{\mathfrak{c}(a)}?^{\mathfrak{c}(a)}c$ is not provable. Thus, the* *process Q inside the agent a remains blocked, i.e., the sequent $\mathcal{P}[\![R]\!]_{nil} \longrightarrow !^{\mathfrak{c}(a)}?^{\mathfrak{c}(a)}d \otimes \top$ is not provable. This intuitively means that the space that b confers to a may behave differently (i.e., it contains different information) from the own space of a. The same reasoning applies for the process $R = [[P]_a]_a \parallel [Q]_a$. This means that, in general, the space of a inside a is different from the space a ($a \not\preceq a.a$). If we want spaces to be idempotent, we simply need to add the equivalence $a.a \approx a$ to the pre-order.*

Theorem 8 (Adequacy)**.** *Let P be an* `sccp` *process, $(C_s, \vdash_s)$ be an SCS, Ψ be a set of process definitions. Let $\mathcal{C}[\![\cdot]\!]_{\bar{\ell}}$ and $\mathcal{P}[\![\cdot]\!]_{\bar{\ell}}$ be as in Definition 6. Then $P \Downarrow_c$ iff* $!^{\mathfrak{c}(\infty)}[\![\Delta_s]\!], !^{\mathfrak{p}(\infty)}[\![\Psi]\!], \mathcal{P}[\![P]\!]_{nil} \longrightarrow \mathcal{P}[\![c]\!]_{nil} \otimes \top$.

Proof. The only difference from the CCP case in Theorem 6 is the possibility of applying the rule R_S. Assume that the formula $!^{\bar{\ell}}F$ is the context, corresponding to the encoding of a process of the shape $[P]_{\bar{\ell}}$. By induction on P, we can show that if $P \longrightarrow P'$, once we focus on $!^{\bar{\ell}}F$, then the encoding $!^{\bar{\ell}}F'$ is is also in the context where F' corresponds to $\mathcal{P}[\![P']\!]_{\bar{\ell}}$.

The remaining cases are similar to the ones shown in the proof of adequacy of CCP. For example, when $P = \mathbf{tell}(c)$, consider the focused derivation below obtained when focusing on its encoding:

$$\dfrac{\dfrac{\dfrac{[C', \mathcal{D}, \mathcal{P}] \longrightarrow [G]}{[C, \mathcal{D}, \mathcal{P}], \mathcal{C}[\![c]\!]_{\bar{\ell}'} \longrightarrow [G]}\; j \times \exists_L, n \times \otimes_L, n \times !_L}{[C, \mathcal{D}, \mathcal{P}] \xrightarrow{\Cap l_x : \bar{a}.\mathcal{C}[\![c]\!]_{(l_x : \bar{a})}} [G]}\; \Cap_L, R_L}{[C, \mathcal{D}, \mathcal{P} +_{\mathfrak{p}(\bar{\ell})} \Cap l_x : \bar{a}.\mathcal{C}[\![c]\!]_{(l_x : \bar{a})}] \longrightarrow [G]}\; D$$

Notice that, due to the pre-order, the only instantiation of the quantified subexponential $l_x : \bar{a}$ is when $l_x = \bar{a}$. As before, the encoding of $\mathcal{C}[\![c]\!]_{\bar{\ell}'}$ is of the form $\bigtriangledown_{\mathfrak{c}(\bar{a}.\bar{\ell}_1)} A_1, \ldots, \bigtriangledown_{\mathfrak{c}(\bar{a}.\bar{\ell}_n)} A_n$, where all constraints are in the box $\bar{a}$. The remaining cases are similar. □

9. Timed CCP

Saraswat *et al.* proposed in [9] timed-CCP (`tcc`), an extension of the CCP model for the specification of reactive systems. In `tcc`, time is conceptually divided into *time intervals* (or *time units*). In a particular time interval, a CCP process P gets an input c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines also a residual process Q which is then executed in the next time unit. The resulting store d is not automatically transferred to the next time unit. Hence, computations during a time-unit proceed monotonically (by adding information to the store), but outputs of two different time-units are not supposed to be related to each other. This view of *reactive computation* is akin to synchronous languages such as Esterel [29], where the system reacts continuously with the environment at a rate controlled by the environment.

The syntax of the monotonic fragment of `tcc` is defined as:

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

The first kind of processes are the same as in CCP. The process **next** P delays the execution of P in one time-unit. The *replication* of P, written as **always** P, means $P \parallel \textbf{next}\ P \parallel \textbf{next}\ \textbf{next}\ P \parallel \ldots$, *i.e.*, unboundly many copies of P, but one at a time.

In `tcc`, recursive calls are assumed to be guarded by a **next** process to avoid non-terminating sequences of recursive calls during a time-unit [9]. Then, recursion can be encoded via replication [9, 27] and we omit it here. We also note that we considered here the monotonic fragment of the `tcc` calculus, *i.e.*, the fragment of `tcc` without the time-out **unless** c (**next** P) that executes P in the next time-unit when the guard c cannot be entailed in the current time-unit The reason is that this process lacks of a proper proof theoretic semantics: the reduction to P amounts to showing that there is no proof of c.

In `tcc`, we distinguish between internal ($\longrightarrow$) and observable ($\Longrightarrow$) transitions. The internal transition $(X; \Gamma; c) \longrightarrow (X'; \Gamma'; c')$ is similar to that of CCP plus the additional rules for the timed constructs (see Figure 4(c)). A process **always** P executes one copy of P in the current time-unit and then, executes again **always** P in the next time-unit (Rule $R_{\Box}$). The seemingly missing rule for **next** P is given by the observable transition relation.

Assume that $(\emptyset; \Gamma; c) \longrightarrow^* (X; \Gamma'; c') \not\longrightarrow$. We say that Γ under input c outputs $\exists X.c'$ and we write $\Gamma \stackrel{(c,\exists X.c')}{\Longrightarrow} \Upsilon$. The process $\Upsilon = (\textbf{local}\, X)\, F(\Gamma')$ corresponds to the *future* of Γ':

$$F(\Gamma') = \begin{cases} \emptyset & \text{if } \Gamma' = \textbf{ask}\ c\ \textbf{then}\ Q \\ F(\Gamma_1), \ldots, F(\Gamma_n) & \text{if } \Gamma' = \Gamma_1, \ldots, \Gamma_2 \\ \Gamma_1 & \text{if } \Gamma' = \textbf{next}\ \Gamma_1 \end{cases} \tag{4}$$

If, $\Gamma = \Gamma_1 \stackrel{(1,c_1)}{\Longrightarrow} \Gamma_2 ... \Gamma_n \stackrel{(1,c_n)}{\Longrightarrow} \Gamma_{n+1}$ and $c_n \vdash_{\Delta} c$, we say that Γ *eventually* outputs c and we write $\Gamma \Downarrow_c$.

Roughly, the future function drops any ask whose guard cannot be entailed from the final store of the current time-unit. Furthermore, it unfolds the processes guarded by a **next** operator. Notice that the definition of $F(\cdot)$ does not consider the processes **tell**(c), **always** P and (**local** x) P since all of them have an internal transition. Therefore, in a final configuration $(X, \Gamma, c) \not\longrightarrow$ they must occur within the scope of a **next** process.

As before, we use a specific subexponential signature but with only two families $\mathfrak{c}$ and $\mathfrak{p}$:

$$I = \{\infty, nil\} \cup \{i, i{+} \mid i \geq 1\} \text{ and } U = \{\mathfrak{c}(i), \mid i \in I\} \cup \{\mathfrak{p}(\infty)\}.$$

Notice that only the subexponentials marking constraints, $\mathfrak{c}(\cdot)$, and replicated processes, $\mathfrak{p}(\infty)$, are unbounded, as they can be used as many times as needed. On the other hand, subexponentials processes, $\mathfrak{p}(\cdot)$, are bounded.

The pre-order is as depicted in Figure 5(c), where a descending chain is formed with the numbers marked with +. Intuitively, the subexponential i is used to specify a given time-unit while $i+$ is used to store processes valid *from* the time-unit i on. This chain captures the semantics of **always** P: if **always** P appears in time i, then P should be available at any future time. Formally, by using such chain, we are able to specify, by using a quantifier $\Cap l_x : i+$, that P can be instantiated anywhere in the ideal of $i+$, *i.e.*, in future time units.

Definition 7 (Timed Constraint in SELL$^{\Cap}$)**.** *We instantiate $\bigtriangledown_\ell$ as $!^\ell ?^\ell$. The interpretation $C\llbracket\cdot\rrbracket_\ell$ is as in Definition 2, while we modify $\mathcal{P}\llbracket\cdot\rrbracket_\ell$ as follows:*

$$
\begin{array}{l}
\mathcal{P}\llbracket \mathbf{tell}(c)\rrbracket_\ell = !^{\mathrm{p}(\ell)} C\llbracket c\rrbracket_\ell \\
\mathcal{P}\llbracket \mathbf{ask}\ c\ \mathbf{then}\ P\rrbracket_\ell = !^{\mathrm{p}(\ell)}(C\llbracket c\rrbracket_\ell \multimap \mathcal{P}\llbracket P\rrbracket_\ell) \\
\mathcal{P}\llbracket (\mathbf{local}\,\overline{x})\ P\rrbracket_\ell = !^{\mathrm{p}(\ell)}(\exists \overline{x}.(\mathcal{P}\llbracket P\rrbracket_\ell)) \\
\mathcal{P}\llbracket \mathbf{next}\ P\rrbracket_i = \mathcal{P}\llbracket P\rrbracket_{i+1} \\
\mathcal{P}\llbracket \mathbf{always}\ P\rrbracket_i = !^{\mathrm{p}(\infty)} \Cap l_x : i+\ (\mathcal{P}\llbracket P\rrbracket_{(l_x:i+)})
\end{array}
$$

The encoding of the non-temporal operators are similar as before, just that we do not need the subexponential quantification. While the encoding of **next** P is straightforward, the encoding of **always** P is more interesting. If the process **always** P is executed in the time-unit i, then the encoding of P must be available in subexponentials representing the subsequent time-units. For example, let $P =$ **always ask** c **then** Q. The process P must execute Q in all time-units $j \geq i$ whenever c can be deduced in j. We make use of universal quantification over locations to capture this behavior. For instance, if c holds in time-unit j, we have a derivation of the form

$$
\dfrac{\dfrac{\dfrac{}{!^{\mathrm{c}(j)}?^{\mathrm{c}(j)}c \longrightarrow !^{\mathrm{c}(j)}?^{\mathrm{c}(j)}c} \quad \Gamma, \mathcal{P}\llbracket P\rrbracket_j \longrightarrow G}{\Gamma, !^{\mathrm{c}(j)}?^{\mathrm{c}(j)}c, !^{\mathrm{c}(j)}?^{\mathrm{c}(j)}c \multimap \mathcal{P}\llbracket P\rrbracket_j \longrightarrow G}}{\Gamma, !^{\mathrm{c}(j)}?^{\mathrm{c}(j)}c, !^{\mathrm{p}(\infty)}\Cap l_x : i+.(C\llbracket c\rrbracket_\ell \multimap \mathcal{P}\llbracket P\rrbracket_\ell) \longrightarrow G}
$$

We note that the observable transition results from a finite sequence of internal transitions (see rule R_{Obs} in Figure 4(c)). Proof theoretically, detecting that a given configuration can no longer be reduced is problematic in general. In fact, the adequacy theorem below is not on the level of proofs, as our previous theorems, but only at the level of provability [16]: P outputs c iff one can prove that there is a time-unit where c holds. Key for proving this theorem is the use of $!^\ell ?^\ell$ prefixes as for the `sccp` case. More precisely, some derived facts are confined to a determinate time unit: any formula derived in a subexponential representing a time unit is not spilled to other subexponentials, unless explicitly specified.

Theorem 9 (Adequacy)**.** *Let P be a `tcc` process, (C_t, Δ_t) be a CS and $\mathcal{P}\llbracket\cdot\rrbracket_\ell$ as in Definition 7. Then $P \Downarrow_c$ iff $!^{\mathrm{c}(\infty)}\llbracket \Delta_t\rrbracket, \mathcal{P}\llbracket P\rrbracket_1 \longrightarrow \Cup \ell : 1+.!^{\mathrm{c}(\ell)}?^{\mathrm{c}(\ell)}c \otimes \top$.*

Proof. Timed behavior is quite different from the cases analyzed before since there are two notions of barbs: internal and observable.

From the proof theoretical point of view, though, the cases are similar and simpler than the ones described in the CCP case since information is confined to time units due to the use of question marks and the encoding of non-replicated processes does not have quantification over subexponentials. The only different cases are focusing on $\mathcal{P}\llbracket \mathbf{next}\ P\rrbracket_i = \mathcal{P}\llbracket P\rrbracket_{i+1}$ and $\mathcal{P}\llbracket \mathbf{always}\ P\rrbracket_i = !^{\mathrm{p}(\infty)}\Cap l_x : i+(\mathcal{P}\llbracket P\rrbracket_{(l_x:i+)})$ but these cases are also trivial.

On the other hand, notice that if $P \Downarrow_c$, then, there is a derivation of the form:

$$P \equiv P_1 \xRightarrow{(1,c_1)} P_2 \xRightarrow{(1,c_2)} \cdots P_n \xRightarrow{(1,c_n)} P_{n+1}$$

and $c_n \vdash_{\Delta_t} c$. We shall discharge the proof by showing that the internal (Equation 5 below) and the observable (Equation 6 below) derivations preserve provability.

For the internal derivation, assume that $(X;\Gamma;d) \longrightarrow^* (X \cup X';\Gamma';d \wedge d')$. We shall show that for any $i \geq 1$, and $e \in \mathcal{C}_t$,

$$\begin{array}{ll} \textbf{if} & !^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \mathcal{P}[\![\Gamma]\!]_i, \mathcal{C}[\![d]\!]_i \longrightarrow \mathcal{C}[\![e]\!]_i \otimes \top \\ \textbf{then} & !^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \exists X'.(\mathcal{P}[\![\Gamma']\!]_i \otimes \mathcal{C}[\![d]\!]_i \otimes \mathcal{C}[\![d']\!]_i) \longrightarrow \mathcal{C}[\![e]\!]_i \otimes \top \end{array}$$

The proof proceeds by induction on the length of the derivation with case analysis on the last rule applied. The resulting cases are analogous to those in the proof of CCP adequacy and we only consider the case **always** P (recall that **next** P does not exhibit any internal transition). We know that $(X;\Gamma,\textbf{always}\ P;d) \longrightarrow (X;\Gamma,P,\textbf{next always}\ P;d)$. Consider the formulas $F = \mathcal{P}[\![\textbf{always}\ P]\!]_i = !^{\mathfrak{p}(\infty)} \Cap l_x : i{+}(\mathcal{P}[\![P]\!]_l)$ and $F' = \mathcal{P}[\![P]\!]_i \otimes \mathcal{P}[\![\textbf{next always}\ P]\!]_i = \mathcal{P}[\![P]\!]_i \otimes \mathcal{P}[\![\textbf{always}\ P]\!]_{i+1}$. Consider now the sequent

$$!^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \mathcal{P}[\![\Gamma]\!]_i, F, \mathcal{C}[\![d]\!]_i \longrightarrow \mathcal{C}[\![e]\!]_i \otimes \top$$

We notice that in any proof of such sequent, given that $\mathcal{C}[\![e]\!]_i = !^{\mathfrak{c}(i)}?^{\mathfrak{c}(i)}e$, none of the instances of F of the form $\mathcal{P}[\![P]\!]_j$ with $j > i$ can be used (since $\mathfrak{p}(i)$, $\mathfrak{p}(j)$, $\mathfrak{c}(i)$ and $\mathfrak{c}(j)$ are unrelated). On the other side, due to the connective $!^{\mathfrak{p}(\infty)}$ in F, several instances of the form $\mathcal{P}[\![P]\!]_i$ can be used in the proof of the sequent. Nevertheless, since P is a deterministic process, it is easy to prove by structural induction that for any G, $\mathcal{P}[\![P]\!]_i \longrightarrow G$ iff $\mathcal{P}[\![P \parallel P]\!]_i \longrightarrow G$. Hence the sequent above is provable iff the following one is provable:

$$!^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \mathcal{P}[\![\Gamma]\!]_i, \mathcal{P}[\![P]\!]_i, \mathcal{C}[\![d]\!]_i \longrightarrow \mathcal{C}[\![e]\!]_i \otimes \top$$

and the result follows.

From here we conclude:

$$\text{if } c_i \vdash_{\Delta_t} e \text{ then } !^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \mathcal{P}[\![P]\!]_i \longrightarrow \mathcal{C}[\![e]\!]_i \otimes \top \tag{5}$$

As for the observable derivation, assume that $(X;\Gamma,d) \not\longrightarrow$. Note that the process to be executed in the next time-unit corresponds to $(\textbf{local}\ X)\ F(\Gamma)$ where F is the future function in Equation 4. Let G be a formula of the form $!^{l_j}?^{l_j}G'$ where $i < j$, i.e., G' is an observation of a future time-unit j. We can show that

$$\begin{array}{ll} \textbf{if} & !^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \exists X.(\mathcal{P}[\![\Gamma]\!]_i \otimes \mathcal{C}[\![d]\!]_i) \longrightarrow !^{l_j}?^{l_j}G' \\ \textbf{then} & !^{\mathfrak{c}(\infty)}[\![\Delta_t]\!], \exists X \mathcal{P}[\![F(\Gamma)]\!]_{i+1} \longrightarrow !^{l_j}?^{l_j}G' \end{array} \tag{6}$$

For that, notice $\mathcal{C}[\![d]\!]_i$ takes the form $!^{\mathfrak{c}(i)}?^{\mathfrak{c}(i)}F$, and then, this formula has to be deleted in a proof for $!^j?^jG'$ (since $l_i \npreceq l_j$). This is, the current store is forgotten and it cannot be used to prove properties in the future. Now we analyze $\mathcal{P}[\![\Gamma]\!]_i$ and $\mathcal{P}[\![F(\Gamma)]\!]_{i+1}$. It is easy to see that $\mathcal{P}[\![\textbf{next}\ P]\!]_i \equiv \mathcal{P}[\![P]\!]_{i+1}$. Notice that $F(\textbf{ask}\ c\ \textbf{then}\ P) = \emptyset$. If Γ contains a process **ask** c **then** P, it must be the case that $d \nvdash_{\Delta_t} c$. Then, $\mathcal{P}[\![P]\!]_i$ could not be used in a proof for G. □

10. Concluding Remarks

This paper introduced SELLF$^{\Cap}$, a focused system which is the extension of SELL with subexponential quantifiers and proved that cut elimination is admissible for the SELLF$^{\Cap}$ system, reflecting a pleasant duality with the standard quantification over terms. We demonstrated that these quantifiers form a powerful tool for specifying languages involving modalities. This was done by proposing novel encodings for Constraint Concurrent Programming models that include epistemic, spatial and timed modalities.

We believe that there are many directions to follow from this work. For instance, in our encoding, we did not need the generation of *fresh* subexponential variables by using the rules $\Cap_R$ and $\Cup_L$. For example, as done with *eigenvariables* for modeling nonces in security protocol [28], it seems possible to create *new* modalities, such as new spaces or new agents not related to the ones already created. This would solve the limitation of `sccp` and `eccp` in [10] where the set of agents is fixed.

Although this paper does not consider non-determinism, some form of it can easily be captured. For instance non-deterministic choice of the form $P + Q$ can be encoded as the formula $F = \mathcal{P}[\![P]\!]_l \ \& \ \mathcal{P}[\![Q]\!]_l$ as it was shown in [7]. In fact, by adding/moving subexponential bangs, it is possible to model precisely don't-care and don't-know choices [13]. Thus, non-determinism (not-considered in [10] for neither `sccp` nor `eccp`) can be also introduced in `sccp`, where processes do not contract. For a second example, consider the `ntcc` calculus [27] which extends `tcc` with guarded non-deterministic choices and asynchrony. For the later, the process $\star P$ represents an arbitrary long, but finite delay for the activation of P; that is, $\star P$ non-deterministically chooses $n \geq 0$ and behaves as **next** $^n P$ (see Rule $\mathrm{R}_\star$ in Figure 4(c)). It seems possible to encode this behavior by extending $\mathcal{P}[\![\cdot]\!]_\ell$ with the following case: $\mathcal{P}[\![\star P]\!]_i = \Cup l_x : i{+}.\mathcal{P}[\![P]\!]_{(l_x:i+)}$. Roughly, if $\star P$ is executed in time-unit i, then *there is* a subexponential j such that $j \preceq i+$ (*i.e.*, a future time-unit j) and the encoding of P holds using that subexponential. However, for adequacy, some care has to be taken due to undesired interactions between **always** and non-deterministic processes (containing $\star$ or $+$), such as **always** P, where P is non-deterministic: $\mathcal{P}[\![\textbf{always } P]\!]_l$ yields a formula of the form $!^{\mathsf{p}(\infty)}F$. Due to the connective $!^{\mathsf{p}(\infty)}$ that precedes F, by contraction, it is possible to have a derivation with two copies of F representing the process $P \parallel P$ that does not behave as P, thus breaking adequacy. A way to overcome this is by imposing that non-deterministic processes are not bound by a **always** process.

We also envision that CCP research can greatly profit from this work. Due to the modularity of our encoding, it seems possible to design variants of CCP by simply configuring the subexponentials differently or using different prefixes. For instance, by using a mix of linear and unbounded $\mathfrak{c}(\cdot)$ subexponentials, it is possible to specify a spatial CCP calculus that allows constraints to be consumed. Moreover, we are currently exploring the idea of instantiating the subexpontial structure as a semiring to allow agents to ask and tell soft constraints [30] representing, e.g., preferences, probabilities, costs, etc.

Also, as discussed above, it seems possible to design CCP models that allow for the creation of new spaces or agents. One move in this direction was given in [31], where the use of existential ($\Cup$) and universal ($\Cap$) quantifications over subexponentials in $\text{SELLF}^{\Cap}$ were used in order to endow CCP with the ability to communicate location (space) names. The resulting CCP language obtained is a model of distributed computation where it is possible to dynamically establish new shared spaces for communication. We thus extended the sort of mobility achieved for variables to dynamically change the shared spaces among agents.

Finally, it would be fruitful understanding better the connections of our work with Hybrid Logics. Reed in this Ph.D. thesis [32] proposes a Hybrid Logical Framework. This framework is similar to Linear Logic with Subexponentials as it also combines the use of standard logic (first-order logic) with modal operators. A future work is to investigate the use and quantification of other types of subexponentials in the same spirit as in Hybrid Logics.

References

[1] L. Cardelli, A. D. Gordon, Mobile ambients, Theor. Comput. Sci. 240 (1) (2000) 177–213.

[2] V. A. Saraswat, M. C. Rinard, Concurrent constraint programming, in: F. E. Allen (Ed.), POPL, ACM Press, 1990, pp. 232–245.

[3] V. A. Saraswat, Concurrent Constraint Programming, MIT Press, 1993.

[4] M. Abadi, M. Burrows, B. W. Lampson, G. D. Plotkin, A calculus for access control in distributed systems, ACM Trans. Program. Lang. Syst. 15 (4) (1993) 706–734.

[5] V. A. Saraswat, M. C. Rinard, P. Panangaden, Semantic foundations of concurrent constraint programming, in: D. S. Wise (Ed.), POPL, ACM Press, 1991, pp. 333–352.

[6] C. Olarte, C. Rueda, F. D. Valencia, Models and emerging trends of concurrent constraint programming, Constraints 18 (4) (2013) 535–578.

[7] F. Fages, P. Ruet, S. Soliman, Linear concurrent constraint programming: Operational and phase semantics, Inf. Comput. 165 (1) (2001) 14–41.

[8] J.-Y. Girard, Linear logic, Theor. Comput. Sci. 50 (1987) 1–102.

[9] V. A. Saraswat, R. Jagadeesan, V. Gupta, Timed default concurrent constraint programming, J. Symb. Comput. 22 (5/6) (1996) 475–520.

[10] S. Knight, C. Palamidessi, P. Panangaden, F. D. Valencia, Spatial and epistemic modalities in constraint-based process calculi, in: M. Koutny, I. Ulidowski (Eds.), CONCUR, Vol. 7454 of Lecture Notes in Computer Science, Springer, 2012, pp. 317–332.

[11] V. Nigam, On the complexity of linear authorization logics, in: LICS, IEEE, 2012, pp. 511–520.

[12] V. Danos, J.-B. Joinet, H. Schellinx, The structure of exponentials: Uncovering the dynamics of linear logic proofs, in: G. Gottlob, A. Leitsch, D. Mundici (Eds.), Kurt Gödel Colloquium, Vol. 713 of Lecture Notes in Computer Science, Springer, 1993, pp. 159–171.

[13] V. Nigam, D. Miller, Algorithmic specifications in linear logic with subexponentials, in: A. Porto, F. J. López-Fraguas (Eds.), PPDP, ACM, 2009, pp. 129–140.

[14] J.-M. Andreoli, Logic programming with focusing proofs in linear logic, J. Log. Comput. 2 (3) (1992) 297–347.

[15] K. Watkins, I. Cervesato, F. Pfenning, D. Walker, A concurrent logical framework I: Judgments and properties, Tech. Rep. CMU-CS-02-101, Carnegie Mellon University, revised, May 2003 (2003).

[16] V. Nigam, D. Miller, A framework for proof systems, J. Autom. Reasoning 45 (2) (2010) 157–188.

[17] V. Nigam, C. Olarte, E. Pimentel, A general proof system for modalities in concurrent constraint programming, in: P. R. D'Argenio, H. C. Melgratti (Eds.), CONCUR, Vol. 8052 of Lecture Notes in Computer Science, Springer, 2013, pp. 410–424.

[18] A. S. Troelstra, Lectures on Linear Logic, CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford, California, 1992.

[19] K. Chaudhuri, Classical and intuitionistic subexponential logics are equally expressive, in: A. Dawar, H. Veith (Eds.), CSL, Vol. 6247 of Lecture Notes in Computer Science, Springer, 2010, pp. 185–199.

[20] V. Nigam, Exploiting non-canonicity in the sequent calculus, Ph.D. thesis, Ecole Polytechnique (Sep. 2009).

[21] V. Nigam, E. Pimentel, G. Reis, Specifying proof systems in linear logic with subexponentials, Electr. Notes Theor. Comput. Sci. 269 (2011) 109–123.

[22] G. Gentzen, Investigations into logical deductions, in: M. E. Szabo (Ed.), The Collected Papers of Gerhard Gentzen, North-Holland, Amsterdam, 1969, pp. 68–131.

[23] D. Miller, A. F. Tiu, A proof theory for generic judgments: An extended abstract, in: LICS, IEEE Computer Society, 2003, pp. 118–127.

[24] D. Miller, A. Saurin, From proofs to focused proofs: A modular proof of focalization in linear logic, in: J. Duparc, T. A. Henzinger (Eds.), CSL, Vol. 4646 of Lecture Notes in Computer Science, Springer, 2007, pp. 405–419.

[25] D. S. Scott, Domains for denotational semantics, in: M. Nielsen, E. M. Schmidt (Eds.), ICALP, Vol. 140 of LNCS, Springer, 1982, pp. 577–613.

[26] G. Smolka, A foundation for higher-order concurrent constraint programming, in: J.-P. Jouannaud (Ed.), Proceedings of Constraints in Computational Logics, Vol. 845 of LNCS, Springer-Verlag, 1994, pp. 50–72.

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

[28] N. A. Durgin, P. Lincoln, J. C. Mitchell, Multiset rewriting and the complexity of bounded security protocols, Journal of Computer Security 12 (2) (2004) 247–311.

[29] G. Berry, G. Gonthier, The ESTEREL synchronous programming language: Design, semantics, implementation, Science of Computer Programming 19 (2) (1992) 87–152.

[30] S. Bistarelli, U. Montanari, F. Rossi, Soft concurrent constraint programming, ACM Trans. Comput. Log. 7 (3) (2006) 563–589.

[31] C. Olarte, V. Nigam, E. Pimentel, Dynamic spaces in concurrent constraint programming, Electr. Notes Theor. Comput. Sci. 305 (2014) 103–121.

[32] J. Reed, A hybrid logical framework, Ph.D. thesis, Department of Computer Science, CMU (2009).