Subexponential concurrent constraint programming

Carlos Olarte

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

Vivek Nigam

Universidade Federal da Paraíba. João Pessoa, Brazil.

Elaine Pimentel

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

Abstract

In previous works we have shown that linear logic with subexponentials (SELL), a refinement of linear logic, can be used to specify emergent features of concurrent constraint programming (CCP) languages, such as preferences and spatial, epistemic and temporal modalities. In order to do so, we introduced a number of extensions to SELL, such as subexponential quantifiers for the specification of modalities, and more elaborated subexponential structures for the specification of preferences. These results provided clear proof theoretic foundations to existing systems. This paper goes in the opposite direction, answering positively the question: can the proof theory of linear logic with subexponentials contribute to the development of new CCP languages? We propose a CCP language with the following powerful features: 1) computational spaces where agents can tell and ask preferences (soft-constraints); 2) systems where spatial and temporal modalities can be combined; 3) shared spaces for communication that can be dynamically established; and 4) systems that can dynamically create nested spaces. In order to provide the proof theoretic foundations for such a language, we propose a unified logical framework (SELLS$^{\Cap}$) combining the extensions of linear logic with subexponentials mentioned above, and showing that this new framework has interesting proof theoretical properties such as cut-elimination and a sound and complete focused proof system.

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

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

1. Introduction

Logic and proof theory play an important role in the design of programming languages. In fact, new programming constructs have been proposed by following tight connections between programming languages and proof theory. For example, we investigated recently in [1] a proof theoretic specification of the concurrent constraint programming (CCP [2]) languages introduced in [3] that mention epistemic (`eccp`) and spatial (`sccp`) modalities. We used as underlying logical framework linear logic with subexponentials (SELL) [4, 5], together with new quantifiers on subexponentials: $\Cap$ and $\Cup$, allowing, respectively, the universal and existential quantification of subexponentials. The focusing discipline then enforced that the obtained encodings for `eccp` and `sccp` 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 [6].

The study done in [1] allowed the development of extensions of `eccp` and `sccp` with features not available in [3]. For example, we were able to specify systems with an unbounded number of agents for `eccp` or spaces for `sccp` as well as to specify new constructs that allow the communication of location names [7].

More recently, in [8], we have shown that SELL (without the subexponential quantifiers) can be configured to capture CCP languages manipulating soft-constraints [9]. The underlying (soft) constraint system in these CCP calculi is based on semirings structures and has been used for the specification of systems that mention preferences, e.g., costs, probabilities, levels of uncertainty (fuzzy information), etc. However, moving from hard (crisp) constraints to soft constraints was not followed by a corresponding logical/proof theoretic characterization of these systems. This is unfortunate because one of the key motivations of the original CCP was its tight connection to logic and proof theory which enabled the proposal of more advanced systems such as its linear version `lcc` [10]. Hence, the main contribution of [8] was to recover this connection by studying the proof theory of soft constraint systems in the form of SELL theories.

It is worthy saying that it is not possible to specify in SELL some notions of soft-constraints, namely those based on non-idempotent semirings. For this, we introduced in [8] a new proof system, called SELLS, for which the subexponential promotion rule behaves differently. This new rule is quite interesting since it coincides with the usual one in SELL in the case of idempotent semirings. Moreover, it faithfully captures notions such as probabilities and costs that require non-idempotent semirings as the underlying algebraic structure.

The tight correspondence between soft constraints and proof theory that we found allows us now to extend `eccp` and `sccp` with soft constraints, thus strengthening the main results in [1] and [8]. Hence, SELL can be regarded as a uniform underlying logical framework for a number of programming languages.

Building the foundations of programming languages, such as CCP, on solid proof theory reduces the principles used for designing a programming language to the foundations of logic. The comparison of different programming principles and languages can then be established by soundness and completeness results. In contrast, while model-theoretic approaches have also played an important role in the development of programming languages, including CPP, they allow for many specifications, which are many times hard to be compared and some may even be considered ad-hoc.

This paper continues our research program of using extensions of linear logic with subexponentials to provide solid proof-theoretical foundations to different CCP languages as well as for the development of new programming constructs. Our main goal is to propose a unified general logical framework where many variants of CCP may be specified, including new ones, all with clear proof-theoretical foundations.

We summarize our main contributions below:

- We propose new subexponential quantifiers which are considerably more expressive than the ones introduced in our previous work [1]. Subexponentials are organized into a pre-order specifying the provability relation between them. While in our previous work we allowed only the quantification of subexponentials that are in the ideal of a single subexponential, our new quantifiers allow for the quantification of subexponentials that appear in the ideal of any subexponential of a given non-empty set of subexponentials, or between two subexponentials. We prove that the resulting system, called $\mathrm{SELLS}^{\Cap}$, admits cut-elimination;

- We demonstrate that a number of CCP languages with different modalities can be specified as $\mathrm{SELLS}^{\Cap}$ theories. For that, we define a general language called $\mathcal{M}\mathtt{ccp}$ where the programmer can express, and combine, different modalities. More precisely, we show how the new quantifiers naturally induce new CCP operators which allow for the sharing and exporting of information between processes. For instance, we show how to formally express the situation when some information can be exported from an agent a to another agent b, but it is confined to these agents only. Thus, two agents are able to share private spaces. Moreover, we allow the combination of spatial modalities, as proposed in [3], and preferences (soft-constraints) [11]. This means that agents may not only share constraints, but also preferences, allowing for the specification of systems with spatial modalities constrained to levels of uncertainty.

- Finally, we propose a focused proof system [12] for $\mathrm{SELLS}^{\Cap}$. Focusing is a discipline on proofs introduced originally for linear logic in order to reduce the proof search non-determinism. We show that our focused proof system is complete with respect to $\mathrm{SELLS}^{\Cap}$. Moreover, the adequacy results relating the operational semantics of $\mathcal{M}\mathtt{ccp}$ and derivations in $\mathrm{SELLS}^{\Cap}$ rely on the focused proof system.

The remainder of the paper is organized as follows. In Section 2, we review linear logic with subexponentials and propose $\mathrm{SELLS}^{\Cap}$, which includes the new subexponential quantifiers. In this section we also prove that the system admits cut-elimination. Section 3 proposes a focused proof system for SELLS$^{\Cap}$ and prove its soundness and completeness. We use SELLS$^{\Cap}$ as a logical framework to propose new CCP constructs in Section 4 demonstrating that they increase considerably the expressiveness of existing CCP calculi. Finally, in Section 5, we conclude by commenting on related work and pointing out future work.

It should be noted that a preliminary attempt to using SELL in order to specify space-mobility (see Section 4.4) was presented in [7]. In this paper we give many more examples and explanations. We also refine several technical details and present more detailed proofs. While we add much more proof-theoretical machinery to specify new CCP languages, we leave the explanation of the different concepts and their relation to proof theory to the examples in the paper. The new contributions with respect to [7] are: (1) we show how to combine, in a unique logical framework, spatial modalities and preferences. For that, (2) we develop the (focus) SELLS$^{\Cap}$ system; (3) the new type system for location introduced in Section 2.2 allows us to define in a neater way the generation of new locations to be shared among agents; finally, (4) we develop the theory of agents that can export information to sublocation, a feature considered neither in [3] nor in [7].

2. Modalities in linear logic

In [1] and [8] we presented two linear logic based systems with subexponentials: SELL and SELLS respectively. Both rely on a poset organization of the subexponentials: while SELL requires a simple preorder structure, SELLS asks for a more involved algebraic system – a c-semiring (see Example 2.2).

In this work we will combine both systems, where the underlying algebraic structure is a poset with some extra structure, but not as strong as a c-semiring. In this way, we are able to (1) give the most general possible definition for SELLS; and (2) enable different kinds of modalities in a single logical framework.

2.1. Linear Logic with Subexponentials

SELLS (Linear Logic with Soft SubExponentials) shares with intuitionistic linear logic [13] all its connectives except the exponentials: instead of having a single pair of *exponentials* ! and ?, SELLS may contain as many *subexponentials* [4, 5] as needed (see [14] for a gentle introduction to subexponentials). Figure 1 presents the introduction rules of intuitionistic linear logic without the exponentials.

Contraction and weakening on formulas in linear logic are controlled by using the exponentials, whose inference rules are shown below:

$$\frac{\Gamma, F \longrightarrow G}{\Gamma, !F \longrightarrow G}\ !_L \quad \frac{!\Gamma \longrightarrow G}{!\Gamma \longrightarrow !G}\ !_R \quad \frac{!\Gamma, F \longrightarrow ?G}{!\Gamma, ?F \longrightarrow ?G}\ ?_L \quad \frac{\Gamma \longrightarrow G}{\Gamma \longrightarrow ?G}\ ?_R \quad \frac{\Gamma \longrightarrow G}{\Gamma, !F \longrightarrow G}\ W \quad \frac{\Gamma, !F, !F \longrightarrow G}{\Gamma, !F \longrightarrow G}\ C$$

Notice that we can only introduce a ! on the right or a ? on the left if all formulas in the context are *classical*, that is all formulas on the left-hand-side of the sequent must be marked with a ! and the formula on right-hand-side must

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

$$\frac{\Gamma, F, H \longrightarrow G}{\Gamma, F \otimes H \longrightarrow G}\,\otimes_L \quad \frac{\Gamma_1 \longrightarrow F \quad \Gamma_2 \longrightarrow H}{\Gamma_1, \Gamma_2 \longrightarrow F \otimes H}\,\otimes_R \quad \frac{\Gamma, F_i \longrightarrow G}{\Gamma, F_1 \,\&\, F_2 \longrightarrow G}\,\&_{L_i} \quad \frac{\Gamma \longrightarrow F \quad \Gamma \longrightarrow H}{\Gamma \longrightarrow F \,\&\, H}\,\&_R$$

$$\frac{\Gamma_1 \longrightarrow F \quad \Gamma_2, H \longrightarrow G}{\Gamma_1, \Gamma_2, F \multimap H \longrightarrow G}\,\multimap_L \quad \frac{\Gamma, F \longrightarrow H}{\Gamma \longrightarrow F \multimap H}\,\multimap_R \quad \frac{\Gamma, F \longrightarrow G \quad \Gamma, H \longrightarrow G}{\Gamma, F \oplus H \longrightarrow G}\,\oplus_L \quad \frac{\Gamma \longrightarrow F_i}{\Gamma \longrightarrow F_1 \oplus F_2}\,\oplus_{R_i}$$

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

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

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.

be marked with a ?. The rules $!_R$ and $?_L$ are called *promotion rules*, while the rules $!_L$ and $?_R$ are called *dereliction rules*.

We now substitute the exponentials with a (possibly infinite) set of labeled ones, called *subexponentials*. We start by defining their algebraic structure.

Definition 2.1 ($\times$-poset)**.** *A* partial-order *on a nonempty set P is a binary relation $\leq$ on P that is reflexive, antisymmetric and transitive. The pair $(P, \leq)$ is called a* partially ordered set*, or* poset*. A poset having minimum ($\bot$) and maximum ($\top$) elements is called* bounded*. A* $\times$-poset *$\langle P, \leq, \times\rangle$ is a bounded partial-order together with a binary operation $\times$ (here called product) which is (1) associative; (2) commutative; (3) $\top$ is the neutral element of $\times$, that is, $\forall a \in A, a \times \top = a$; (4) monotone, i.e., $\forall a, b, c, d \in P$ if $a \leq d$ and $b \leq a \times c$, then $b \leq d \times c$; and (5) intensive: $\forall a, b \in P$, $a \times b \leq a$. Moreover, if $glb(a, b)$ exists and $a \times b = glb(a, b)$ for all $a, b \in P$, then the $\times$-poset is called* idempotent.

Observe that $(P, \leq, \times, \top)$ is an abelian ordered monoid, with the extra properties of monotonicity and intensiveness. Note also that $\bot$ is $\times$-absorbing, i.e., $a \times \bot = \bot$. Finally, if P is the real $[0, 1]$ interval, then a $\times$-poset is a t-norm. In this case, the monotonicity guarantees that the degree of *preference* (see [8]) does not decrease if the truth values of the product increase.

Example 2.1. *Every bounded distributive lattice $\langle L, \vee, \wedge, \mathbf{0}, \mathbf{1}\rangle$ is an idempotent $\times$-poset, where $a \leq b$ if and only if $a \vee b = b$ and $\times = \wedge$. In fact:*

- *if $a \leq d$ then $a \vee d = d$. Hence, $d \wedge c = (a \vee d) \wedge c = (a \wedge c) \vee (d \wedge c)$, thus $a \wedge c \leq d \wedge c$;*

- *since $b \vee \mathbf{1} = \mathbf{1}$ and $a \wedge \mathbf{1} = a$, we have that $a = a \wedge (b \vee \mathbf{1}) = (a \wedge b) \vee (a \wedge \mathbf{1}) = (a \wedge b) \vee a$. Hence $a \wedge b \leq a$.*

Example 2.2 (C-semiring [9])**.** *A c-semiring (see Section 4.1 for some examples) is a tuple $\langle \mathcal{A}, +, \times, \bot, \top\rangle$ satisfying: (S1) $\mathcal{A}$ is a set and $\bot, \top \in \mathcal{A}$; (S2) $+$ is a binary, commutative, associative and idempotent operator on $\mathcal{A}$, $\bot$ is its unit element and $\top$ its absorbing element; (S3) $\times$ is a binary, associative* *and commutative operator on $\mathcal{A}$ with unit element $\top$ and absorbing element $\bot$. Moreover, $\times$ distributes over $+$.*

Let $\leq$ be defined as $a \leq b$ iff $a + b = b$. Then, $\langle \mathcal{A}, \leq \rangle$ is a complete lattice where: (S4) $+$ and $\times$ are monotone on $\leq$; (S5) $\times$ is intensive on $\leq$; (S6) $\bot$ (resp. $\top$) is the bottom (resp. top) of $\mathcal{A}$; (S7) $+$ is the lub operator.

If $\times$ is idempotent, then: (S8) $+$ distributes over $\times$; (S9) $\langle \mathcal{A}, \leq \rangle$ is a complete distribute lattice and $\times$ is its glb. A c-semiring is idempotent if its $\times$ operator is idempotent, and non-idempotent otherwise.

Clearly, $\langle \mathcal{A}, \leq, \times \rangle$ is a $\times$-poset and, if $\times$ is idempotent, $\langle \mathcal{A}, \leq, \times \rangle$ is an idempotent $\times$-poset.

Example 2.3. *Let $(P, \leq)$ be a bounded poset and define $\times$ as: $a \times b = (\downarrow a) \cap (\downarrow b)$ where $\downarrow a$ is the ideal of a, that is, $\downarrow a = \{x \in A \mid x \leq a\}$. Observe that the intersection of ideals is an ideal and it is not empty since $\bot \in (\downarrow a)$ for all $a \in A$. Moreover, $a \leq b$ if and only if $(\downarrow a) \subseteq (\downarrow b)$. Hence monotonicity and intensiveness hold trivially and $\langle P, \leq, \times \rangle$ is a $\times$-poset.*

A SELLS$_\Sigma$ system is specified by a *subexponential signature* $\Sigma = \langle A, \preceq, U \rangle$, where A is a set of labels, $\langle A, \preceq, \times_\Sigma \rangle$ is a $\times$-poset having minimum, maximum $\bot, \top \in A$ and a product $\times_\Sigma$, and $U \subseteq A$ specifies which subexponentials allow both weakening and contraction. We shall use $a, a_1, \ldots$ to range over elements in A and we will assume that $\preceq$ is upwardly closed with respect to U, *i.e.*, if $a \in U$ and $a \preceq a_1$, then $a_1 \in U$.

For a given such subexponential signature, SELLS$_\Sigma$ is the system obtained by substituting the linear logic exponential ! by the subexponential $!^a$ for each $a \in A$, and by adding to the rules in Figure 1 the following inference rules:
- for each $a \in A$ (dereliction and the promotion rules):

$$\frac{\Gamma, F \longrightarrow G}{\Gamma, !^a F \longrightarrow G}\ !^a{}_L \qquad \frac{!^{a_1}F_1, \ldots, !^{a_n}F_n \longrightarrow F}{!^{a_1}F_1, \ldots, !^{a_n}F_n \longrightarrow !^a F}\ !^a{}_R, \text{ provided } a \preceq a_1 \times_\Sigma \ldots \times_\Sigma a_n.$$

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

- for each $b \in U$ (structural rules):

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

Observe that provability is preserved *downwards* i.e., if the sequent $\Gamma \longrightarrow !^a P$ is provable in SELLS$_\Sigma$, so is the sequent $\Gamma \longrightarrow !^{a_1} P$ for all $a_1 \preceq a$. We shall elide the signature Σ whenever it is not important or clear from the context.

In [1], we showed that by using different prefixes it is possible to interpret subexponentials in interesting ways, such as temporal units or spatial and epistemic modalities. Moreover, in [8] we showed how to capture the notion of *preferences* (soft-constraints) using subexponentials. In this paper, we will show how to combine these modalities in a single system. In order to do so, we need the notion of *quantification over subexponentials*, to be presented next.

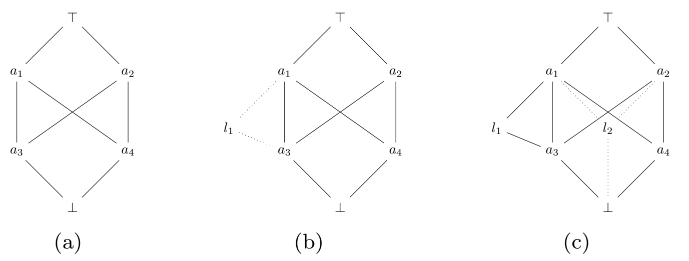

Figure 2: Creating subexponential variables in a $\times$-poset.

2.2. SELLS$^{\Cap}$ system

We will now enhance the notion of quantification over subexponentials presented in [1]. We will call the resulting system SELLS$^{\Cap}$.

The initial subexponential signature of SELLS$^{\Cap}$ is the SELLS signature presented in last section, $\langle A, \preceq, U\rangle$. We will call the elements in A *subexponential constants*. SELLS$^{\Cap}$ will also allow *subexponential variables*. Intuitively, these variables will be introduced by the subexponential quantifiers in a similar fashion as the usual *eigenvariables* in first-order systems. Before presenting the subexponential quantifiers, we will add some machinery and set the notation.

We start by generalizing the quantification presented in [1], adding a broader notion of typing to subexponential constants and variables. We use three kinds of typing: one for subexponential constants, and two for subexponential variables. Here l is a subexponential variable, *i.e.*, $l \notin A$, a a subexponential constant, *i.e.*, $a \in A$, s denotes both subexponential constants and variables and $i \in \{b, u\}$ indicates whether the subexponential is bounded or unbounded:

$$a : \{a\}_i \qquad\qquad l : \{s_1, \ldots, s_n\}_i \qquad \text{and} \qquad l : \{s_1/s_2\}_i,$$

where $n \geq 1$ and $s_2 \prec s_1$. The typing $l : \{s_1, \ldots, s_n\}_i$ specifies that $\bot \prec l$ and the subexponential l is in the ideal of all the subexponentials in $\{s_1, \ldots, s_n\}$, that is, $l \preceq s_j$ for all $1 \leq j \leq n$. The typing $l : \{s_1/s_2\}_i$ specifies that $s_2 \preceq l \preceq s_1$. Observe that here the *sandwich rule* applies, that is, if both s_1 and s_2 are unbounded (resp. bounded), so it will be l, hence $i = u$ (resp. $i = b$). For subexponential constants we just have $a : \{a\}_i$ specifying that a is in its own ideal. Here, $i = u$ if $a \in U$ and $i = b$ otherwise. We note that we could have simply removed the typing of subexponential constants, but the definition of the proof system is considerably simplified by using the more uniform and rather trivial typing $a : \{a\}$.

In the following, we shall omit the subscript i when it can be inferred from the context or it is not relevant. Moreover, we shall use the letters $l, l_1, l_2, \ldots$ for subexponential variables, $a, a_1, a_2, \ldots$ for subexponential constants, $s, s_1, s_2, \ldots, d, d_1, d_2, \ldots$ for both subexponential variables and constants and τ for any of the three typing expressions above.

Example 2.4. *Consider the subexponential signature $\Sigma = \langle A, \preceq, A\rangle$ presented in Figure 2(a), where $\times_\Sigma$ is a product defined as $a_i \times_\Sigma a_j = glb(a_i, a_j)$ if*

$glb(a_i, a_j)$ exists and $a_i \times_\Sigma a_j = \bot$ otherwise. If the subexponential variable $l_1 : \{a_1/a_3\}$ is added, we obtain the Figure 2(b). The $\times$-poset obtained by further adding $l_2 : \{a_1, a_2\}$ is shown in Figure 2(c).

SELLS$^{\Cap}$ sequents have the form $\mathcal{S}; \Gamma \longrightarrow G$, where $\mathcal{S} = \mathcal{A}_\Sigma \cup \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$, with $\{l_1, \ldots, l_n\}$ a disjoint set of subexponential variables and $\mathcal{A}_\Sigma = \{a : \{a\}_i \mid a \in A\}$. Formally, only these subexponential constants and variables may appear free in an index of subexponential bangs and question marks.

Let $S = \{l \mid (l : \tau) \in \mathcal{S}\}$. The *sequent pre-order* $\preceq_S$ is defined in S as the transitive and reflexive closure of the set:

$$\begin{array}{l}\preceq \cup \{(l, \top), (\bot, l) \mid l \in S\} \cup \\ \{(l, s) \mid (l : \{s_1, \ldots, s_n\}), (s : \tau) \in \mathcal{S}, \text{ and } (s_j, s) \text{ for some } 1 \leq j \leq n\} \cup \\ \{(l, s_1), (s_2, l) \mid (l : \{s_1/s_2\}) \in \mathcal{S}\}\end{array}$$

Observe that $\bot, \top$ remain the minimum and maximum elements wrt $\preceq_S$.

The grammar of the formulas of SELLS$^{\Cap}$ extends the formulas of SELLS by adding the subexponential quantifiers as follows:

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

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 $\mathcal{S}$:

$$\dfrac{\mathcal{S}; \Gamma, F[s/l] \longrightarrow G}{\mathcal{S}; \Gamma, \Cap l : \{s_1, \ldots, s_n\}_i.F \longrightarrow G}\ \Cap_{L1}(\star 1) \qquad \dfrac{\mathcal{S}; \Gamma, F[s/l] \longrightarrow G}{\mathcal{S}; \Gamma, \Cap l : \{s_1/s_2\}_i.F \longrightarrow G}\ \Cap_{L2}(\star 2)$$

$$\dfrac{\mathcal{S}; \Gamma \longrightarrow G[s/l]}{\mathcal{S}; \Gamma \longrightarrow \Cup l : \{s_1, \ldots, s_n\}_i.G}\ \Cup_{R1}(\star 1) \qquad \dfrac{\mathcal{S}; \Gamma \longrightarrow G[s/l]}{\mathcal{S}; \Gamma \longrightarrow \Cup l : \{s_1/s_2\}_i.G}\ \Cup_{R2}(\star 2)$$

$$\dfrac{\mathcal{S}, l_e : \tau; \Gamma, F[l_e/l] \longrightarrow G}{\mathcal{S}; \Gamma, \Cup l : \tau.F \longrightarrow G}\ \Cup_L(\star 3) \qquad \dfrac{\mathcal{S}, l_e : \tau; \Gamma \longrightarrow G[l_e/l]}{\mathcal{S}; \Gamma \longrightarrow \Cap l : \tau.G}\ \Cap_R(\star 3)$$

where l_e is fresh, *i.e.*, not appearing in $\mathcal{S}$ in the rules $\Cup_L, \Cap_R$, and the side conditions are defined as follows

($\star$1) $s : \tau \in \mathcal{S}$ is such that $s \preceq_S s_j$ for all $1 \leq j \leq n$ and if $i = b$ then s is bounded otherwise it is unbounded;

($\star$2) $s : \tau \in \mathcal{S}$ is such that $s_1 \preceq_S s \preceq_S s_2$ and if $i = b$ then s is bounded otherwise it is unbounded;

($\star$3) provided the relation $\preceq_{S'}$ is a pre-order, upward closed with respect to the set $U_{\mathcal{S}'}$, where $\mathcal{S}' = \mathcal{S}, l_e : \tau$ and $U_{\mathcal{S}'} = \{s \mid (s : \{\tau\}_u) \in \mathcal{S}'\}$.

In order to complete the poset w.r.t. the $\times_\Sigma$-operator, we define the $\times$ operator using the pre-order $\preceq_S$ for a given set of typed subexponentials $\mathcal{S}$, as:

$$s_1 \times s_2 = \begin{cases} s_1 \times_\Sigma s_2 & \text{if } \{s_1, s_2\} \subseteq A; \\ glb(s_1, s_2) & \text{if } \{s_1, s_2\} \not\subseteq A \text{ and if } glb(s_1, s_2) \text{ exists in } \preceq_S; \\ \bot & \text{if } \{s_1, s_2\} \not\subseteq A \text{ and if } glb(s_1, s_2) \text{ does not exist in } \preceq_S. \end{cases}$$

Observe that, due to the definition of $\preceq_S$, $\bot$ is $\times$-absorbing and $\top$ is the neutral element of $\times$. Moreover, it is trivial to check that $\times$ is associative, commutative, monotone and intensive. Hence, $\langle S, \preceq_S, \times\rangle$ is a $\times$-poset, called the *underlying* SELLS$^{⋒}$ $\times$-*poset*.

The ordering $\preceq_S$ is used in the right-introduction of bangs and the left-introduction of question-marks in a similar way as before in SELLS:

$$\frac{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n \longrightarrow G}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n \longrightarrow !^{s}G}\ !^s{}_R, s \preceq_S s_1 \times \cdots \times s_n$$

$$\frac{\mathcal{S}; !^{s_1}F_1, \ldots !^{s_n}F_n, P \longrightarrow ?^{s_{n+1}}G}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n, ?^{s}P \longrightarrow ?^{s_{n+1}}G}\ ?^s{}_L, s \preceq_S s_1 \times \cdots \times s_n \times s_{n+1}$$

2.3. Cut-Elimination

For proving that SELLS$^{⋒}$ admits the cut rule, we start by stating the straightforward result of admissibility of weakening for unbounded subexponentials.

Lemma 2.1 (Weakening)**.** *Let u be an unbounded subexponential. If the sequent $\mathcal{S}; \Gamma \longrightarrow C$ is provable in SELLS$^{⋒}$ then $\mathcal{S}; \Gamma, !^{u}F \longrightarrow C$ is provable in SELLS$^{⋒}$.*

It is well known that cut-elimination holds for SELL [15, 4]. An important observation for guaranteeing that SELLS$^{⋒}$ have the same property is in order: when substituting a subexponential variable l_e by a subexponential s of the same type, all the relations and properties valid for l_e are "inherited" by s. This intuitive idea is formally described in Appendix B, together with the proof of Theorem 2.2 below.

Theorem 2.2. *The cut rule below is admissible in SELLS$^{⋒}$.*

$$\frac{\mathcal{S}; \Gamma_1 \longrightarrow G \quad \mathcal{S}; \Gamma_2, G \longrightarrow F}{\mathcal{S}; \Gamma_1, \Gamma_2 \longrightarrow F}\ Cut$$

3. SELLFS$^{⋒}$- a Focused Proof System for SELLS$^{⋒}$

Focusing is a discipline on proofs [12] first proposed by Andreoli for Linear Logic [13]. Although initially developed for reducing proof search space, focused proof systems have been successfully used as logical frameworks for specifying deductive systems, such as proof systems [6, 14] and programming languages [5, 1, 16]. Focusing plays an important technical role in this paper as it provides the proof theoretic means to establish the adequacy of the logical specification of the calculus we propose (see Theorem 4.1).

We propose two focused proof systems for SELLS$^{⋒}$, one for when the underlying subexponential $\times$-poset $\langle S, \preceq_S, \times\rangle$ is idempotent, another for when $\langle S, \preceq_S, \times\rangle$ is not idempotent (see Definition 2.1). We prove the completeness of these proof systems with respect to SELLS$^{⋒}$.

The main challenge is to handle the new exponentials described in Section 2.2. As the introduction rules for these connectives apply if the sequent context satisfies specific conditions, we need to take care when developing the focused proof systems. We take the strategy used by Andreoli [12] and define first dyadic versions of SELLS$^{\Cap}$ before introducing the focused proof systems. We prove their soundness and completeness. Then we propose the the focused proof system showing that they are sound and complete following standard techniques [17], omitting most of the details.

3.1. The dyadic system SELLS^d

The dyadic system SELLS^d is given in Figure 3 with the exception of the right bang and left question mark introduction rules and the rules introducing the subexponential quantifiers, which will be introduced later. Its sequents have the following form:[1]

$$\mathcal{S}; \mathcal{K} : \mathcal{L} : \Gamma \longrightarrow C$$

Let $U_{\mathcal{S}} = \{s \mid s : \tau_u \in \mathcal{S}\}$ be the set of unbounded subexponentials in $\mathcal{S}$ and $I_{\mathcal{S}} = \{s \mid s : \tau_i \in \mathcal{S}, i \in \{u, b\}\}$ be the set of all subexponential in $\mathcal{S}$. In the sequent above, Γ is a multiset of linear logic formulas, C is a linear logic formula, $\mathcal{K}$ is a function from $U_{\mathcal{S}}$ to *sets* of linear logic formulas, and $\mathcal{L}$ is a function from $I_{\mathcal{S}} \setminus U_{\mathcal{S}}$ to *multisets* of linear logic formulas. We call $\mathcal{K}$ the unbounded context and $\mathcal{L}$ the linear one. Intuitively, $\mathcal{K}[u] = \{F_1, \ldots, F_n\}$ and $\mathcal{L}[b] = \{F_1, \ldots, F_n\}$ should be interpreted as $!^s F_1, \ldots, !^s F_n$, for $s = u$ or $s = b$, respectively. We will normally elide the typing context $\mathcal{S}$ whenever it is not important.

In order to introduce the proof system for SELLS^d, we need some operations on contexts. Here $\mathcal{B}$ is a set of bounded subexponentials, $\mathcal{U}$ a set of unbounded subexponentials and $\star \in \{\subset, \subseteq, =\}$ is a set comparison operation:

$$\mathcal{L}[\mathcal{B}] = \biguplus_{b \in \mathcal{B}} \mathcal{L}[b]$$
$$\mathcal{K}[\mathcal{U}] = \bigcup_{u \in \mathcal{U}} \mathcal{K}[u]$$
$$(\mathcal{L} +_b F)[b'] = \begin{cases} \mathcal{L}[b'] \uplus \{F\} & \text{if } b' = b \\ \mathcal{L}[b'] & \text{otherwise} \end{cases}$$
$$(\mathcal{K} +_u F)[u'] = \begin{cases} \mathcal{K}[u'] \cup \{F\} & \text{if } u' = u \\ \mathcal{K}[u'] & \text{otherwise} \end{cases}$$
$$(\mathcal{L}_1 \otimes \mathcal{L}_2)[b] = \mathcal{L}_1[b] \uplus \mathcal{L}_2[b] \text{ for all } b \in I \setminus U$$

We will sometimes abuse of the notation and write $\mathcal{L}$ for $\mathcal{L}[I_{\mathcal{S}} \setminus U_{\mathcal{S}}]$ and $\mathcal{K}$ for $\mathcal{K}[U_{\mathcal{S}}]$ for a given typing context $\mathcal{S}$.

Notice that the dyadic system does not contain explicit contraction nor weakening rules. These are incorporated into the introduction rules. For example, in

[1]Instead of using a single context for both bounded and unbounded subexponentials as done in [15], we use two contexts, one for unbounded and another for bounded. This is a difference only in presentation of the system – we will continue calling the system *dyadic*.

the $\otimes_R$ and $\multimap_L$ rules, the unbounded context, $\mathcal{K}$, is copied among the premises, while the bounded context is split among them. Since unbounded formulas are allowed to contract and also weaken, we do not lose provability by doing so. On the other hand, the initial rule, 1_R and as we will see the $!_R$ rules incorporate the weakening rule. In particular, formulas in the unbounded context are weakened.

$$\frac{}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow A}\ I, \text{provided } \{A\} = \mathcal{L} \uplus \Gamma \text{ or } A \in \mathcal{K} \text{ and } (\mathcal{L} \uplus \Gamma) = \emptyset$$

$$\frac{\mathcal{K} : \mathcal{L} : \Gamma, F, G \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, F \otimes G \longrightarrow H}\ \otimes_L \qquad \frac{\mathcal{K} : \mathcal{L}_1 : \Gamma_1 \longrightarrow F \quad \mathcal{K} : \mathcal{L}_2 : \Gamma_2 \longrightarrow G}{\mathcal{K} : \mathcal{L}_1 \otimes \mathcal{L}_2 : \Gamma_1, \Gamma_2 \longrightarrow F \otimes G}\ \otimes_R$$

$$\frac{\mathcal{K} : \mathcal{L} : \Gamma, F_i \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, F_1 \,\&\, F_2 \longrightarrow H}\ \&_{L_i} \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F \qquad \mathcal{K} : \mathcal{L} : \Gamma \longrightarrow G}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F \,\&\, G}\ \&_R$$

$$\frac{\mathcal{K} : \mathcal{L}_1 : \Gamma_1 \longrightarrow F \qquad \mathcal{K} : \mathcal{L}_2 : \Gamma_2, G \longrightarrow H}{\mathcal{K} : \mathcal{L}_1 \otimes \mathcal{L}_2 : \Gamma_1, \Gamma_2, F \multimap G \longrightarrow H}\ \multimap_L \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma, F \longrightarrow G}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F \multimap G}\ \multimap_R$$

$$\frac{\mathcal{K} : \mathcal{L} : \Gamma, F \longrightarrow H \qquad \mathcal{K} : \mathcal{L} : \Gamma, G \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, F \oplus G \longrightarrow H}\ \oplus_L \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F_i}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F_1 \oplus F_2}\ \oplus_{R_i}$$

$$\frac{}{\mathcal{K} : \mathcal{L} : \Gamma, 0 \longrightarrow H}\ 0_L \quad \frac{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, 1 \longrightarrow H}\ 1_L \quad \frac{}{\mathcal{K} : \mathcal{L} : \cdot \longrightarrow 1}\ 1_R, \text{provided, } \mathcal{L} = \emptyset$$

$$\frac{}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow \top}\ \top_R \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma, F[e/x] \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, \exists x.F \longrightarrow H}\ \exists_L \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F[t/x]}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow \exists x.F}\ \exists_R$$

$$\frac{\mathcal{K} : \mathcal{L} : \Gamma, F[t/x] \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, \forall x.F \longrightarrow H}\ \forall_L \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow F[e/x]}{\mathcal{K} : \mathcal{L} : \Gamma \longrightarrow \forall x.F}\ \forall_R$$

$$\frac{\mathcal{K} +_u F : \mathcal{L} : \Gamma \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, !^u F \longrightarrow H}\ !_{L1} \qquad \frac{\mathcal{K} : \mathcal{L} +_b F : \Gamma \longrightarrow H}{\mathcal{K} : \mathcal{L} : \Gamma, !^b F \longrightarrow H}\ !_{L2}$$

$$\frac{\mathcal{K} +_u F : \mathcal{L} : \Gamma, F \longrightarrow H}{\mathcal{K} +_u F : \mathcal{L} : \Gamma \longrightarrow H}\ D_{L1} \qquad \frac{\mathcal{K} : \mathcal{L} : \Gamma, F \longrightarrow H}{\mathcal{K} : \mathcal{L} +_b F : \Gamma \longrightarrow H}\ D_{L2}$$

Figure 3: The fragment of the dyadic system for SELLS without the cut-rule and the right introduction rules for the bang. Here $u \in U$ is an unbounded subexponential and $b \in I \setminus U$ is a bounded subexponential.

The novelty is on the right introduction rule for bang. Let us first define the following two operations on contexts:

$$(\mathcal{K} \geq_u)[u'] = \begin{cases} \mathcal{K}[u'] & \text{if } u' \geq u \\ \emptyset & \text{otherwise} \end{cases}$$
$$\prod(\mathcal{K}[\mathcal{S}]) = \prod_{u \in \mathcal{S}} u^n, \text{where } n = |\mathcal{K}[u]|$$
$$\prod(\mathcal{L}[\mathcal{S}]) = \prod_{b \in \mathcal{S}} b^n, \text{where } n = |\mathcal{L}[b]|$$

Here s^n denotes $\underbrace{s \times \cdots \times s}_{n \text{ times}}$. For example, if $\mathcal{K}[s_1] = \{F_1, F_2\}$ and $\mathcal{K}[s_2] = \{G_1, G_2, G_3\}$, then $\prod(\mathcal{K}[\{s_1, s_2\}]) = s_1 \times s_1 \times s_2 \times s_2 \times s_2$. We write $\prod(\mathcal{K})$ and $\prod(\mathcal{L})$ for $\prod(\mathcal{K}[U])$ and $\prod(\mathcal{L}[I \setminus U])$, respectively. Notice that

The dyadic proof system will have the corresponding promotion rule, $!^s{}_R$ and $!^s{}_{RS}$, depending on whether the underlying $\times$-poset is idempotent or not.

Idempotent $\times$-poset.

$$\frac{\mathcal{S}; \mathcal{K} \geq_s : \mathcal{L} : \cdot \longrightarrow F}{\mathcal{S}; \mathcal{K} : \mathcal{L} : \cdot \longrightarrow !^s F} \; !^s{}_R, \text{provided } \mathcal{L}[s'] = \emptyset \text{ for all } s \not\preceq_{\mathcal{S}} s'$$

$$\frac{\mathcal{S}; \mathcal{K} \geq_s : \mathcal{L} : F \longrightarrow ?^{s'} H}{\mathcal{S}; \mathcal{K} : \mathcal{L} : ?^s F \longrightarrow ?^{s'} H} \; ?^s{}_L, \text{provided } \mathcal{L}[s''] = \emptyset \text{ for all } s \not\preceq_S s'' \text{ and } s \preceq_S s'$$

Non-idempotent $\times$-poset.

$$\frac{\mathcal{S}; \mathcal{K}' : \mathcal{L} : \cdot \longrightarrow F}{\mathcal{S}; \mathcal{K} : \mathcal{L} : \cdot \longrightarrow !^s F} \; !^s{}_{RS}, \text{provided } \mathcal{K}' \subseteq \mathcal{K} \text{ and } s \preceq_S \prod(\mathcal{K}') \times \prod(\mathcal{L})$$

$$\frac{\mathcal{S}; \mathcal{K}' : \mathcal{L} : F \longrightarrow ?^{s'} H}{\mathcal{S}; \mathcal{K} : \mathcal{L} : ?^s F \longrightarrow ?^{s'} H} \; ?^s{}_{LS}, \text{provided } \mathcal{K}' \subseteq \mathcal{K} \text{ and } s \preceq_S \prod(\mathcal{K}') \times \prod(\mathcal{L}) \times s'$$

There is an important difference on proof search between these rules. The first pair of rules, $?^s{}_L, !^s{}_R$, has a *don't care non-determinism*: one simply weakens all formulas that are marked with subexponentials smaller than s. The second pair of rules, $?^s{}_{LS}, !^s{}_{RS}$, has a *don't know non-determinism*: one needs to choose subsets of formulas in the context $\mathcal{K}$ obtaining $\mathcal{K}'$ such that its side-condition is satisfied.

One comment is in order: if $s_j \times s_k = glb(s_j, s_k)$ for all s_j, s_k, then the signature is an idempotent $\times$-poset. Thus, the condition $s \preceq_S s_1 \times \ldots \times s_n$ is equivalent to the condition $s \preceq_S s_i$ for all $i \in 1..n$. Therefore, the two rules above are equivalent in this case.

We shall then call SELLS$^{⋒}$ the system with the rules $!^s{}_{RS}$ and $?^s{}_{LS}$, understanding that they are more general than $!^s{}_R$ and $?^s{}_L$. The presentation of both pairs of rules has proof-theoretical purposes only, and could serve as inspiration for a more efficient implementation.

Finally, by adding the rules for subexponential quantifiers, we obtain SELLS$^{\Cap d}$, a dyadic system for SELLS$^{\Cap}$. Below we show only some of these rules:

$$\frac{\mathcal{S};\mathcal{K}:\mathcal{L}:\Gamma, F[s/l] \longrightarrow G}{\mathcal{S};\mathcal{K}:\mathcal{L}:\Gamma, \Cap l:\{s_1,\ldots,s_n\}_i.F \longrightarrow G}\ \Cap_{L1}(\star 1)$$

$$\frac{\mathcal{S}, l_e:\tau;\mathcal{S};\mathcal{K}:\mathcal{L}:\Gamma \longrightarrow G[l_e/l]}{\mathcal{S};\mathcal{K}:\mathcal{L}:\Gamma \longrightarrow \Cap l:\tau.G}\ \Cap_{R}(\star 3)$$

The other rules and the conditions $(\star 1), (\star 3)$ are similar to the ones shown in Section 2.2.

It is not hard to prove the soundness and completeness of SELLS$^{\Cap d}$ with respect to SELLS$^{\Cap}$. Most of the cases are given in [15], and some cases involving subexponentials are given in Appendix A.

Theorem 3.1. *SELLS$^{\Cap d}$ is sound and complete with respect to SELLS$^{\Cap}$.*

3.2. The focused system SELLFS$^{\Cap}$

The focused proof system without the promotion rules and the rules for the subexponential quantifiers are depicted in Figure 4. The promotion rules for SELLFS$^{\Cap}$ are shown below:

Idempotent $\times$-poset.

$$\frac{\mathcal{S};\mathcal{K}\geq_s:\mathcal{L}:\cdot \longrightarrow F}{\mathcal{S};\mathcal{K}:\mathcal{L}:\cdot \xrightarrow[!^s F]{}}\ !^s{}_R, \text{provided } \mathcal{L}[s'] = \emptyset \text{ for all } s \not\preceq_S s'$$

$$\frac{\mathcal{S};\mathcal{K}\geq_s:\mathcal{L}:F \longrightarrow [?^{s'}H]}{\mathcal{S};\mathcal{K}:\mathcal{L}:\cdot \xrightarrow{?^s F} [?^{s'}H]}\ ?^s{}_L, \text{provided } \mathcal{L}[s''] = \emptyset \text{ for all } s \not\preceq_S s'' \text{ and } s' \preceq_S s$$

Non-Idempotent $\times$-poset.

$$\frac{\mathcal{S};\mathcal{K}':\mathcal{L}:\cdot \longrightarrow F}{\mathcal{S};\mathcal{K}:\mathcal{L}:\cdot \xrightarrow[!^s F]{}}\ !^s{}_{RS}, \text{provided } \mathcal{K}' \subseteq \mathcal{K} \text{ and } s \preceq_S \textstyle\prod(\mathcal{K}') \times \prod(\mathcal{L})$$

$$\frac{\mathcal{S};\mathcal{K}':\mathcal{L}:F \longrightarrow [?^{s'}H]}{\mathcal{S};\mathcal{K}:\mathcal{L}:\cdot \xrightarrow{?^s F} [?^{s'}H]}\ ?^s{}_{LS}, \text{provided } \mathcal{K}' \subseteq \mathcal{K} \text{ and } s \preceq_S \textstyle\prod(\mathcal{K}') \times \prod(\mathcal{L}) \times s'$$

Again, we only consider $!^s{}_{RS}$ and $?^s{}_{LS}$ as part of our system.

In order to introduce the proof system, we need some more terminology. We classify as *negative* all formulas whose main connective is $\&, \multimap, \forall, ?^s$ 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.*, $\forall_R, \otimes_L$.

Negative Phase

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

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

$$\frac{}{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta, 0 \longrightarrow \mathcal{R}}\ 0_L \qquad \frac{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow F \quad [\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow G}{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow F \,\&\, G}\ \&_R$$

$$\frac{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta, F \longrightarrow \mathcal{R} \quad [\mathcal{K} : \mathcal{L} : \Gamma], \Delta, H \longrightarrow \mathcal{R}}{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta, F \oplus H \longrightarrow \mathcal{R}}\ \oplus_L$$

$$\frac{[\mathcal{K} : \mathcal{L} +_b F : \Gamma], \Delta \longrightarrow \mathcal{R}}{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta, !^b F \longrightarrow \mathcal{R}}\ !^b{}_L,\ b \notin U \qquad \frac{[\mathcal{K} +_u F : \mathcal{L} : \Gamma], \Delta \longrightarrow \mathcal{R}}{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta, !^u F \longrightarrow \mathcal{R}}\ !^u{}_L,\ u \in U$$

Positive Phase

$$\frac{[\mathcal{K} : \mathcal{L}_1 : \Gamma_1]-_{F}\!\rightarrow \quad [\mathcal{K} : \mathcal{L}_2 : \Gamma_2]-_{G}\!\rightarrow}{[\mathcal{K} : \mathcal{L}_1 \otimes \mathcal{L}_2 : \Gamma_1, \Gamma_2]-_{F \otimes G}\!\rightarrow}\ \otimes_R \qquad \frac{[\mathcal{K} : \mathcal{L}_1 : \Gamma_1]-_{F}\!\rightarrow \quad [\mathcal{K} : \mathcal{L}_2 : \Gamma_2] \xrightarrow{H} [G]}{[\mathcal{K} : \mathcal{L}_1 \otimes \mathcal{L}_2 : \Gamma_1, \Gamma_2] \xrightarrow{F \multimap H} [G]}\ \multimap_L$$

$$\frac{[\mathcal{K} : \mathcal{L} : \Gamma]-_{G_i}\!\rightarrow}{[\mathcal{K} : \mathcal{L} : \Gamma]-_{G_1 \oplus G_2}\!\rightarrow}\ \oplus_{R_i} \qquad \frac{[\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{F_i} [G]}{[\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{F_1 \& F_2} [G]}\ \&_{L_i} \qquad \frac{}{[\mathcal{K} : \mathcal{L} : \Gamma]-_{1}\!\rightarrow}\ 1_R, \text{provided, } \mathcal{L} = \emptyset$$

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

$$\frac{}{[\mathcal{K} : \mathcal{L} : \Gamma]-_{A}\!\rightarrow}\ I_R, \text{ provided } \{A\} = \mathcal{L} \uplus \Gamma \text{ or } A \in \mathcal{K} \text{ and } (\mathcal{L} \uplus \Gamma) = \emptyset$$

Structural Rules

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

$$\frac{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow [?^b H]}{[\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow ?^b H}\ []^?_R \qquad \frac{[\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{F} [G]}{[\mathcal{K} : \mathcal{L} : \Gamma, F] \longrightarrow [G]}\ D_{L1} \qquad \frac{[\mathcal{K} : \mathcal{L} : \Gamma]-_{G}\!\rightarrow}{[\mathcal{K} : \mathcal{L} : \Gamma] \longrightarrow [G]}\ D_R$$

$$\frac{[\mathcal{K} +_u NA : \mathcal{L} : \Gamma] \xrightarrow{NA} [G]}{[\mathcal{K} +_u NA : \mathcal{L} : \Gamma] \longrightarrow [G]}\ D_{L2} \qquad \frac{[\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{NA} [G]}{[\mathcal{K} : \mathcal{L} +_b NA : \Gamma] \longrightarrow [G]}\ D_{L3}$$

Figure 4: Focused Proof System for Intuitionistic Linear Logic with Subexponentials (SELLFS$^{⋒}$). 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.

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 (looking bottom-up). For example, the $\&_R$ belongs to the negative phase as provability is not lost when applying this rule. A positive rule, on the other hand, is non-invertible in general and therefore provability may be lost. For example, the rule $\otimes_R$ belongs to the positive phase because provability depends on how the linear formulas in $\mathcal{L}_1 \otimes \mathcal{L}_2$ and in Γ_1, Γ_2 are split among the rules premises.

SELLFS$^{\Cap}$ contains four kinds of sequents.

- $[\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow \mathcal{R}$ 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.
- $[\mathcal{K} : \mathcal{L} : \Gamma] \longrightarrow [F]$ is a sequent representing the end of the negative phase.
- $[\mathcal{K} : \mathcal{L} : \Gamma] {-}_{F}{\to}$ is a sequent focused on the right.
- $[\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{F} [H]$ is a sequent focused on the left.

As one can see from inspecting the proof system in Figure 4, 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 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.

Also the rules of the subexponential quantifiers have the same behavior of as the usual first-order quantifier, that is, $\Cap_R$ and $\Cup_L$ belong to the negative phase, while the remaining rules to the positive phase. We show some of these rules:

$$\frac{\mathcal{S}; [\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{F[s/l]} [G]}{\mathcal{S}; [\mathcal{K} : \mathcal{L} : \Gamma] \xrightarrow{\Cap l:\{s_1,\ldots,s_n\}_i.F} [G]} \ \Cap_{L1}(\star 1)$$

$$\frac{\mathcal{S}, l_e : \tau; \mathcal{S}; [\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow G[l_e/l]}{\mathcal{S}; [\mathcal{K} : \mathcal{L} : \Gamma], \Delta \longrightarrow \Cap l : \tau.G} \ \Cap_R(\star 3)$$

Given the dyadic system SELLS$^{\Cap d}$ and Theorem 3.1, the completeness proof for SELLFS$^{\Cap}$ follows the same lines as in the completeness proof given in [17].

Theorem 3.2. *SELLFS$^{\Cap}$ is sound and complete with respect to SELLS$^{\Cap}$.*

4. Modalities in Concurrent Constraint Programming

Concurrent Constraint Programming (CCP) [2] (see a survey in [18]) is a model for concurrency (see e.g., [19]) where processes *interact* with each other by *telling* and *asking* constraints (pieces of information) in a common store of partial information. CCP combines the traditional operational view of process calculi with a *declarative* view based on logic. This salient feature has distinguished CCP from other models of concurrency from its inception: processes can be seen, at the same time, as computing agents and as formulas in a given logic. This allows CCP to benefit from the large set of reasoning techniques of both process calculi and logic. In this section we show that SELLS$^{\Cap}$ provides the proof-theoretical foundations for different CCP-based languages. Hence, SELLS$^{\Cap}$ can be regarded as a uniform underlying logical framework to reason about CCP calculi. More interestingly, we put in the hands of CCP programmers and modelers new programming constructs inspired in SELLS$^{\Cap}$ underlying theory. Our extensions to the language thus adheres to its original conception: a model of concurrency where logic and behavioral techniques coexist coherently.

We start with the definition of Constraint System that makes CCP calculi parametric and hence, versatile to be used in different contexts. Following the developments of SELL, we show how different modalities in the constraint system can be integrated in a coherent way. Next, we introduce the language of processes and we provide several examples of the new features available in the proposed extensions of CCP. Finally, we show that both constraints and processes can be interpreted as formulas in SELLS$^{\Cap}$ where operational steps have a one-to-one correspondence with (focused) proofs in SELLFS$^{\Cap}$.

4.1. Subexponential Constraint System

The type of constraints in CCP is not fixed but parametric in a constraint system (CS). Intuitively, a CS provides a signature from which constraints can be built from basic tokens (e.g., predicate symbols), and two basic operations: conjunction to add new information and variable hiding to define local variables. The CS defines also an *entailment* relation ($\vdash$) specifying inter-dependencies between constraints: $c \vdash d$ means that the information d can be deduced from the information c. Such systems can be formalized as a Scott information system as in [2], or they can be built upon a suitable fragment of logic *e.g.*, as in [10, 20]. For instance, the finite domain constraint system (FD) [21] assumes variables to range over finite domains and, in addition to equality, one may have predicates that restrict the possible values of a variable to some finite set, e.g. $x < 42$. The Herbrand constraint system [22] consists of a first-order language with equality. The entailment relation is the one we expect from equality, e.g., $f(x, y) = f(g(a), z)$ must entail $x = g(a)$ and $y = z$.

Here we shall consider a general notion of constraint system that allows us to capture declaratively different behaviors and modalities in CCP. For instance, the constraint system will allow us to *confine* information to a given *location* or to mark some information with a given *preference*. Locations can be thought of as spaces, distributed agents or even temporal modalities. As for preferences, we can interpret such modalities as probabilities, fuzzy information, costs, etc.

Locations. For the spatial information we shall need a poset S with typical elements $s, s', s_i, \ldots$. Locations can be unrelated or it is possible to define systems where two spaces s and s' belong to a hierarchy where s has the right to *export* (or share) information to s' if the relation $s' \preceq s$ holds in S. Some locations are resource aware, i.e., agents can consume information from them while some others are *unbounded*, i.e., information is persistent on them and they belong to the subset S_U of S. As needed by the subexponential structure in SELLS$^{\Cap}$, the partial order $\preceq$ is assumed to be upwardly closed with respect to S_U , i.e., if $s \in S_U$ and $s \preceq s'$, then $s' \in S_U$.

For a more interesting example, consider a set of agents $\mathfrak{A} = \{\mathfrak{a}_1, ..., \mathfrak{a}_n\}$ and a given poset S as above. We can define a new poset $S(\mathfrak{A})$ where $s'_{\mathfrak{a}_i} \preceq s_{\mathfrak{a}_i}$ iff $s' \preceq s$ in S. That is, $S(\mathfrak{A})$ is a disjoint copy of the structure S for each agent in $\mathfrak{A}$. Hence, $s_{\mathfrak{a}}$ can be interpreted as the spatial location s pertaining to the agent $\mathfrak{a}$ which is unrelated to any other location of a different agent $\mathfrak{b}$.

Preferences. It is well known that crisp (hard) constraints fail to represent accurately situations where soft constraints, *i.e.*, preferences, probabilities, uncertainty or fuzziness, are present. In constraint programming [23], two general frameworks have been proposed to deal with soft constraints: *semiring* based constraints [24] and *valued* constraints [25]. Roughly speaking, in both frameworks an algebraic structure defines the operations needed to *combine* soft constraints and choosing when a constraint (or solution) is *better* than another. In [26], it is shown that both frameworks are equally expressive and they are general enough to represent different kinds of soft constraints including, e.g., fuzzy, probabilistic and weighted constraints. Hence, we shall use c-semiring based constraints in order to integrate preferences into the constraint system.

Recall that a c-semiring is a tuple $\langle \mathcal{A}, +, \times, \bot, \top \rangle$ satisfying the properties in Example 2.2 (see Section 2.1). Elements in the set $\mathcal{A}$ (c-semiring values) are used to denote the *upper bound of preference degrees*, or simply *preference level*, where the "preference" could be a probability, cost, etc. The $\times$ operator is used to combine values while $+$ is used to select which is the "best" value in the sense that $a + a' = a'$ iff $a \leq_{\mathcal{A}} a'$ iff a' is "better" than a.

Instances of c-semirings. The c-semiring $S_c = \langle \{\mathtt{true}, \mathit{false}\}, \vee, \wedge, \mathit{false}, \mathtt{true} \rangle$ models crisp contraints. The **fuzzy** c-semiring $S_F = \langle [0, 1], max, min, 0, 1 \rangle$ allows for fuzzy constraints that have an associate preference level in the real interval $[0, 1]$ where 1 represents the best value. In a **probabilistic** setting [27], a constraint c is annotated with its probability of existence where probabilities are supposed to be independent (*i.e.*, no conditional probabilities). This can be modeled with the c-semiring $S_P = \langle [0, 1], max, \times, 0, 1 \rangle$. In **weighted** constraints there is an accumulate cost that can be computed with the c-semiring $S_w = \langle \mathcal{R}^-, max, +, -\infty, 0 \rangle$, where 0 means no cost.

Now we are ready to formally introduce our constraint system with spatial and preferences modalities. The definition below is based on the idea of constraint systems as a fragment of intuitionistic linear logic in [10]. In fact, for modeling the modalities, we shall use (a fragment of) SELLS$^{\Cap}$ as in [1].

Definition 4.1 (Modal Constraint System)**.** *A modal constraint system (***mcs** *for short) is a tuple $(S, \mathcal{A}, \mathcal{C}, \vdash_\Delta)$ where S is a poset defining* spatial *modalities, $\mathcal{A}$ is a c-semiring with only unbounded elements, $\mathcal{C}$ is a set of formulas (constraints) built from a first-order signature and the grammar*

$$\begin{array}{lcll} PC & := & 1 \mid A \mid PC \otimes PC & \texttt{pre-constraints} \\ C & := & PC \mid C \otimes C \mid \exists \overline{x}.C \mid (\!| PC |\!)_a \mid [C]^s_{s'} & \texttt{constraints} \end{array}$$

where A is an atomic formula, $a \in \mathcal{A}$, $s, s' \in S$ and $s' \preceq s$. We shall use c, c', d, d', etc, to denote elements of $\mathcal{C}$. Moreover, let Δ be a set of non-logical axioms of the form $\forall \overline{x}[c \multimap c']$ where 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 $\mathcal{C}[\![\Delta]\!], \mathcal{C}[\![d]\!] \longrightarrow \mathcal{C}[\![d']\!]$ is provable in SELLS$^{\Cap}$ ($\mathcal{C}[\![\cdot]\!]$ and the SELLS$^{\Cap}$ signature Σ are later introduced in Definition 4.7). We shall omit the "Δ" in $\vdash_\Delta$ when it is unimportant or it can be inferred from the context.*

Let us give some intuition. Pre-constraints (PC) are just atoms or conjunctions of atoms. The constraint 1 corresponds to the empty store, i.e., the initial state of computation. The connective $\otimes$ in $C \otimes C$ allows processes to add more information to the store. The existential quantifier hides variables from constraints. The constraint $(\!| PC |\!)_a$ means that the pre-constraint PC was added to the store with an upper bound preference degree $a \in \mathcal{A}$. Finally, the constraint $[c]^s_{s'}$ means that the information c is *located* and *confined* to the space-location s. Moreover, such information can be *exported* (or moved) until the inner (or *weaker*) location $s' \preceq s$. We shall write $[c]_s$ instead of $[c]^s_s$.

As we shall see later, constraints of the form $(\!| PC |\!)_s$ (resp. $[c]^s_{s'}$) are just formulas of the shape $!^a(F)$ (resp. $!^s ?^{s'} F$) in SELLS$^{\Cap}$, where a is an unbounded subexponential (see the encoding in Definition 4.7). For the moment, we shall continue using the notation in Definition 4.1 which is simpler and more intuitive from a programming language perspective.

Let us show some interesting properties of constraints in a **mcs**.

Proposition 4.1 (Properties of of **mcs**)**.** *Let $(S, \mathcal{A}, \mathcal{C}, \vdash_\Delta)$ be a* **mcs** *and assume a non-logical axiom in Δ of the form $c \otimes d \longrightarrow_\Delta 0$ (0 is the ILL unity denoting falsity) . Then,*

- False Confinement. Let $s, s' \in S$ be two different and possibly related locations:

1. $[0]_s \vdash_\Delta [c]_s$ *(any c can be deduced in the space s if its local store is inconsistent);*
2. $[0]_s \not\vdash_\Delta [0]_{s'}$ *and* $[0]_{s'} \not\vdash_\Delta [0]_s$ *(inconsistency is confined);*
3. $[c]_s \otimes [d]_s \vdash_\Delta [0]_s$ *(if space s contains both c and d, then it becomes inconsistent);*

4. $[c]_s \otimes [d]_{s'} \not\vdash_\Delta [0]_s$ *and* $[c]_s \otimes [d]_{s'} \not\vdash_\Delta [0]_{s'}$ *(false is not deduced if c and d are in different spaces);*
5. $[c]_s \not\vdash_\Delta c$ *(local information is not global).*

- Sharing Information. Assume now that $s'' \preceq s' \preceq s$*:*

1. $[c]^s_{s'} \vdash_\Delta [c]_{s'}$ *(information c can be propagated to the space* s'*);*
2. $[c]^s_{s''} \vdash_\Delta [c]_{s'}$ *(information c can be propagated to the intermediate location in the hierarchy);*
3. $[c]_s \not\vdash_\Delta [c]^s_{s'}$ *(information is confined if sharing is not explicit);*
4. $[c]^s_{s'} \vdash_\Delta [c]_s$ *(information shared to sub-locations also hold in the parent location).*

- Preference Behavior. Assume that $a \leq_{\mathcal{A}} a'$ *and* $a'' \leq_{\mathcal{A}} a \times_{\mathcal{A}} a'$*. Reminding that* a, a', a'' *are unbounded:*

1. $(\!|c|\!)_{a'} \vdash_\Delta (\!|c|\!)_a$ *(if c is added with a higher preference* a'*, then it can be deduced with a lower preference a);*
2. $(\!|c|\!)_a \otimes (\!|c|\!)_a \equiv_\Delta (\!|c|\!)_a$ *(information about preferences is idempotent);*
3. $(\!|c|\!)_a \otimes (\!|d|\!)_{a'} \vdash_\Delta (\!|c \otimes d|\!)_{a''}$ *(*$\vdash_\Delta$ *respects the ordering induced by* $+_{\mathcal{A}}$*);*
4. $(\!|c \otimes d|\!)_{a'} \vdash (\!|c|\!)_a \otimes (\!|d|\!)_{a'}$ *(believing both c and d with a given preference level* a' *is stronger than believing c with a preference level* $a \leq_{\mathcal{A}} a'$*).*

Proof. The proof of each of the above entailments $F \vdash_\Delta G$ is straightforward by proving the sequent $\mathcal{C}[\![\Delta]\!], \mathcal{C}[\![F]\!] \longrightarrow \mathcal{C}[\![G]\!]$ in SELLS$^{\Cap}$ ($\mathcal{C}[\![\cdot]\!]$ is later introduced in Definition 4.7). $\square$

Let us give some examples of instances of constraint systems and the behavior they can model.

Example 4.1 (Linear Constraint Systems)**.** *Linear constraint systems [10], where formulas are built from a fragment of intuitionistic linear logic, allowed the development of CCP calculi where agents can consume information from the store. A inear constraint systems can be specified as a* **mcs** *by considering a preorder* $S = \{l, u\}$ *(linear and unbounded) where* $S_U = \{u\}$ *and* $l \preceq u$*. A linear constraint c is then represented as* $[c]_l$ *and any replicated constraint of the form* $!\,c$ *is represented as* $[c]^u_l$*. Note that in this case, constraints are not marked with the* $(\!|\cdot|\!)_a$ *modality and hence* $\mathcal{A}$ *is irrelevant. Observe that representing the unbound constraint* $!\,c$ *as* $[c]^u_l$ *allows us to copy the information c into the linear context since* $[c]^u_l \vdash [c]_l$ *(see Proposition 4.1).*

Example 4.2 (Soft Constraint System)**.** *Soft constraint systems [11] where plugged into CCP to allow agents to tell/ask preferences. Such systems can be represented as a* **mcs** *by restricting constraints to be built without the constructor* $[c]^s_{s'}$*. Preference reasoning on a constraint* $(\!|c|\!)_a$ *is then possible (see Proposition 4.1). Unlike the constraint system proposed in [8], a* **mcs** *allows us also to have different* beliefs *in different locations. For instance, the store* $[(\!|c|\!)_a]_s \otimes [(\!|c|\!)_{a'}]_{s'}$ *models the situation where c is believed with a preference a (resp.* a'*) in the space s (resp.* s'*).*

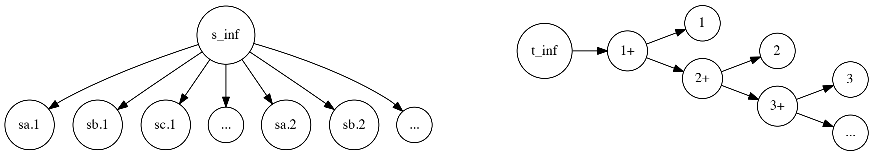

Figure 5: Subexponential structure for spatial and timed modalities. $a \to b$ means $b \preceq a$

Example 4.3 (Spatial Constraint Systems)**.** *Spatial constraint systems, where all information is confined and not shared as in [3], can be specified in a* **mcs** *by disallowing the constructor* $\llparenthesis \cdot \rrparenthesis_a$*. We note that our definition of* **mcs** *is more expressive than the spatial constraint system proposed in [3] since:*

- *Information can be shared in a controlled way thanks to the constructor* $[c]_{s'}^{s}$*. In [3], once a constraint is stored in a given space, it cannot be shared with other locations. In our case,* $[c]_{s'}^{s}$ *means that the information* c *holds in every space* l *such that* $s' \preceq l \preceq s$*. In other words,* c *can be shared according to the hierarchies established by the preorder relation* $\preceq$*.*
- *The* **mcs***, unlike the constraint system in [3], allows for some location to be resource aware (linear). Then, it is possible to define* update *of locations.*

Example 4.4 (Temporal and Spatial Dependencies)**.** *The notion of time was introduced in the context of CCP languages to deal with reactive system (see e.g., [28, 20]). The constraint* $[c]_s$ *can be also interpreted as a temporal modality. For that, consider the preorder in Figure 5. 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 *onward. Hence* $[[c]_2]_{s_a} \otimes [[d]_{3+}]_{s_{a'}}$ *means that* c *holds for agent* a *in time-unit* 2 *while* d *holds for* a' *in all future time-unit* $t \geq 3$*. An interesting application of this constraint system in the modeling of biological systems was recently proposed in [29].*

4.2. The language of CCP processes

In the previous section we gave a general definition of constraint system with modalities. In this section we propose Modal CCP ($\mathcal{M}\mathtt{ccp}$), a CCP language that can manipulate formulas in such constraint system. The main design criteria for $\mathcal{M}\mathtt{ccp}$ are the following:

(i) distributed agents can be defined where local information is private to them. Here the key aspect is to identify agents as unrelated locations (spaces in S). Hence, the information of an agent will be confined to its local store;
(ii) agents can have an internal structure, i.e., its local store can be divided into locations. For that, it suffices to define sublocations for a given agent

in the preorder S. We shall allow unbounded and bounded locations to specify spaces where information can be updated;

(iii) agents are allowed to create, dynamically, new locations. Such locations can be restricted to their own local store or they can be shared with other agents;

(iv) agents are allowed to add preferences to the information posted into their own or shared spaces.

Similar to most processes calculi, the language of processes in $\mathcal{M}$`ccp` features a small number of constructors and it is powerful enough to express interesting behaviors of concurrent and distributed systems. Common to all languages based on CCP, we include constructs to add (*tell*) new information to the store, to hide (local) variables and to compose processes in parallel. Following the developments of `lcc` [10, 30] and `utcc` [31], we allow the quantification of free variables in *ask* processes. Furthermore, as in `lcc`, ask agents consume information when evolving due to the linear nature of the store. Here we notice that, by changing the subexponential structure, we can specify that some stores are persistent while some others are linear. Finally, following the developments of spatial CCP (`sccp`) [3], we allow processes to be confined to a given space (see $[P]_s$ below). However, unlike `sccp`, in $\mathcal{M}$`ccp` it is possible to create and communicate shared spaces of communication between agents. Later we show that this ability is not *ad hoc* since we can give it a declarative meaning thanks to the connectives $\Cup$ and $\Cap$ in SELLS$^{\Cap}$.

Definition 4.2 (Syntax of $\mathcal{M}$`ccp`). *Processes in $\mathcal{M}$`ccp` are built from constraints in the underlying* **mcs** *as follows:*

$$P, Q \quad := \quad \textbf{tell}(c) \mid (\textbf{local}\,\overline{x};\overline{\ell})\,Q \mid (\textbf{abs}\ \overline{x};\overline{\ell};c)\,Q \mid P \parallel Q \mid [P]^{s}_{s'} \mid p(\overline{x})$$

where variables in $\overline{x}$ and spatial (subexponential) typed variables in $\overline{\ell}$ are pairwise distinct. We assume that for each process name, there is a unique process definition of the form $p(\overline{x}) \stackrel{\Delta}{=} P$ where the set of free variables is a subset of $\overline{x}$. Given a set of process definition $\mathcal{D}$ and a process P, a $\mathcal{M}$`ccp` program takes the form $\mathcal{D}.P$.

Let us give some intuition about the processes above. The process $\textbf{tell}(c)$ adds c to the current store d producing the new store $d \otimes c$.

The process $P = (\textbf{local}\,\overline{x};\overline{\ell})\,Q$ creates a new set of variables $\overline{x}$ and declares them to be private to Q. Moreover, the process P creates a set of new locations (spaces) $\overline{\ell}$. The typing information of the variables in $\overline{\ell}$ will determine the kind of location to be created (see Section 2.2). For instance, in the case of $l : \{a_1, a_2\}_b$, the new location l can be used as bounded shared space between the agents (or spaces) a_1 and a_2; in the case of $l : \{a\}_b$ we are just creating a sub-space in a. In order to simplify the notation, we shall omit the subscript "b" in bounded locations. Moreover, if the set $\overline{x}$ is empty, we shall simply write $(\textbf{local}\,\overline{\ell})\,Q$ instead of $(\textbf{local}\,\overline{x};\overline{\ell})\,Q$ when no confusion arises. The same syntactic simplification applies for the set $\overline{\ell}$. Furthermore, instead of $(\textbf{local}\,\{x\};\{\ell\})\,Q$ we shall write $(\textbf{local}\,x;\ell)\,Q$.

The process $P = (\mathbf{abs}\ \overline{x}; \overline{\ell}; c)\, Q$ evolves into $Q[\overline{y}/\overline{x}][\overline{s}/\overline{\ell}]$ if the current store entails $c[\overline{y}/\overline{x}][\overline{s}/\overline{\ell}]$. When either $\overline{x}$ or $\overline{\ell}$ is empty (or a singleton), we use a similar notational convention as we did for the **local** process. Furthermore, when all these sets are empty, we simply write **ask** c **then** Q instead of $(\mathbf{abs}\ \emptyset; \emptyset; c)\, Q$. The **abs** process (which is actually a universally quantified *ask* process) defines a simple and powerful synchronization mechanism based on entailment of constraints: Q is executed only when the information c can be deduced from the store.

Another interesting view of $P = (\mathbf{abs}\ \overline{x}; \overline{\ell}; c)\, Q$ is as a λ-abstraction of the process Q on the variables $\overline{x}$ and the spaces $\overline{\ell}$ under the constraint (or with the guard) c. From a programming language perspective, the variables $\overline{x}$ and $\overline{\ell}$ in $(\mathbf{local}\, \overline{x}; \overline{\ell})\, Q$ can be viewed as the local variables of Q while $\overline{x}$ and $\overline{\ell}$ in $(\mathbf{abs}\ \overline{x}; \overline{\ell}; c)\, Q$ can be viewed as the formal parameters of Q. Following the developments of Universal Timed CCP (`utcc`) [31], we shall show that the interplay of **local** and **abs** processes allows us to communicate share spaces (and variables) among agents.

The parallel composition of P and Q is denoted as $P \parallel Q$.

The processes $[P]^{s}_{s'}$ executes and confines the process P in any space l such that $s' \preceq l \preceq s$. Instead of $[P]^{s}_{s}$ we shall write $[P]_s$.

Finally, given a process definition of the form $p(\overline{x}) \stackrel{\Delta}{=} P$, the agent $p(\overline{y})$ executes the process $P[\overline{y}/\overline{x}]$.

4.3. Operational Semantics

The operational semantics of $\mathcal{M}$`ccp` is given by the transition relation $\gamma \longrightarrow \gamma'$ satisfying the rules on Figure 6. A *configuration* γ takes the form $\langle \overline{x}; \overline{\ell}; \Gamma; c\rangle$ where c is a constraint specifying the current store, Γ is a multiset of processes, $\overline{x}$ is the set of hidden (local) variables of c and Γ and $\overline{\ell}$ is a set of typed locations of the form $l : \tau$ representing the spaces created by processes. The multiset $\Gamma = P_1, P_2, \ldots, P_n$ represents the process $P_1 \parallel P_2 ... \parallel P_n$. We shall indistinguishably use both notations to denote parallel composition of processes.

Processes are quotiented by a structural congruence relation $\cong$ satisfying: (1) $P \cong Q$ if they differ only by a renaming of bound variables (alpha-conversion); (2) $P \parallel Q \cong Q \parallel P$; and (3) $P \parallel (Q \parallel R) \cong (P \parallel Q) \parallel R$. Furthermore, $\Gamma = \{P_1, ..., P_n\} \cong \{P'_1, ..., P'_n\} = \Gamma'$ iff $P_i \cong P'_i$ for all $1 \leq i \leq n$. Finally, $\langle \overline{x}; \overline{\ell}; \Gamma; c\rangle \cong \left\langle \overline{x}'; \overline{\ell}'; \Gamma'; c'\right\rangle$ iff $\overline{x} = \overline{x}'$, $\overline{\ell} = \overline{\ell}'$, $\Gamma \cong \Gamma'$ and $c \equiv_\Delta c'$ (*i.e.*, $c \vdash_\Delta c'$ and $c' \vdash_\Delta c$).

In the following we describe the rules in Figure 6.

Rule $\mathrm{R_T}$ says that the constraint c in $\mathbf{tell}(c)$ is added to the current store. Rule $\mathrm{R_{EQUIV}}$ says that structurally congruent processes have the same transitions. A process $(\mathbf{local}\, \overline{y}; \overline{\ell}_y)\, Q$ adds the local variables $\overline{y}$ (resp. the fresh subexponential variables $\overline{\ell}_y$) to the sets $\overline{x}$ (resp. $\overline{\ell}$) as it is shown in Rule $\mathrm{R_L}$. We shall call the variables in $\overline{\ell}_y$ *spatial variables*. The side condition of this rule simply avoids clash of variables. Notice that such condition can be always fulfilled by using alpha conversion (rule $\mathrm{R_{EQUIV}}$).

If the store d is able to entail $c[\overline{t}/\overline{y}][\overline{\ell}_t/\overline{\ell}_y]$, then the agent $(\textbf{abs}\ \overline{y};\overline{\ell}_y;c)\,Q$ evolves into $Q[\overline{t}/\overline{y}][\overline{\ell}_t/\overline{\ell}_t]$. On doing that, according to the rules of the subexponentials, the constraint c may be consumed. Note that the constraint e in the entailment $d \vdash_\Delta c[\overline{t}/\overline{y}][\overline{\ell}_t/\overline{\ell}_y] \otimes e$ is not necessarily unique. Take for instance an unbounded space l and the entailments $!^l c \vdash_\Delta c \otimes 1$ and $!^l c \vdash_\Delta c \otimes !^l c$. In the first case, $e = 1$ and we have an unwanted weakening of the store, which is not satisfactory since we did not consume the *minimal information* required for the ask agent to proceed. This is avoided in the second entailment, where $e = !^l c$. Moreover, assume now that the current store is $\exists y(c(y))$ – here we use $c(y)$ to explicitly state that $fv(c) = y$. The ask agent $(\textbf{abs}\ x; c(x))\,P$ should be allowed to *open the scope* of the existentially quantified variable y to be able to execute $P[y/x]$. In order to handle these situations, the rule $\mathrm{R_A}$ in Figure 6 states that: (1) the scope of existentially quantified constraints in the store is opened. Note that the premise $\overline{y} \cap fv(X,\Gamma,d)$ guarantees that no clash of variables is produced; and (2), the most general choice (*mgc*) for the residual store is considered to consume the least information required to entail the guard of an ask agent. The *mgc* can be formalized as follows:

Definition 4.3 (Most general choice (*mgc*) [32])**.** *Consider the entailment* $d \vdash_\Delta \exists\overline{y}(e \otimes c[\overline{t}/\overline{x}])$*. Assume also that* $d \vdash_\Delta \exists\overline{y}(e' \otimes c[\overline{t}'/\overline{x}])$ *for an arbitrary* e' *and* $\overline{t}'$*. We say that* e *and* $\overline{t}$ *are the most general choices, notation* $mgc(e,\overline{t})$*, whenever* $e' \vdash_\Delta e$ *implies* $e \vdash_\Delta e'$ *and* $c[\overline{t}/\overline{x}] \vdash_\Delta c[\overline{t}'/\overline{x}]$*.*

Before explaining the rules for $[P]^s_{s'}$ we need some extra definitions.

Definition 4.4. *Let* c *be a constraint and* $\overline{s},\overline{s}'$ *be sequences of spatial locations (elements in* S*). We define* $\nabla^{\overline{s}}_{\overline{s}'}\ c$ *inductively as follows:*

$$\begin{array}{lcllcl}
\nabla^{\overline{s}}_{\overline{s}'}\ PC & = & [PC]^{\overline{s}}_{\overline{s}'} & \nabla^{\overline{s}}_{\overline{s}'}\ (C_1 \otimes C_2) & = & \nabla^{\overline{s}}_{\overline{s}'}\ C_1 \otimes \nabla^{\overline{s}}_{\overline{s}'}\ C_2 \\
\nabla^{\overline{s}}_{\overline{s}'}\ \exists\overline{x}.C & = & \exists\overline{x}.\,\nabla^{\overline{s}}_{\overline{s}'}\ C & \nabla^{\overline{s}}_{\overline{s}'}\ (\!|PC|\!)_a & = & [(\!|PC|\!)_a]^{\overline{s}}_{\overline{s}'} \\
\nabla^{\overline{s}}_{\overline{s}'}\ [C]^l_{l'} & = & \nabla^{\overline{s}.l}_{\overline{s}'.l'}\ C & & &
\end{array}$$

Moreover, we define the projection of c *to the space* s*, notation,* c^s *as the information the space* s *may see or have of* c*, i.e.,* $c^s = \bigotimes\{d \mid c \vdash_\Delta [d]_s\}$*.*

Intuitively, $\nabla^{\overline{s}}_{\overline{s}'}\ C$ confines the information C inside the hierarchy of spaces defined by $\overline{s}$ and $\overline{s}'$. In the case $\nabla^{\overline{s}.l}_{\overline{s}'.l'}\ C$ above, we assume that the locations $\overline{s}.l$ (resp. $\overline{s}'.l'$), representing the space l (resp. l') inside the space $\overline{s}$ (resp. $\overline{s}'$) exists (see Example 4.7). Concerning the projection of the information, if $c = [c_1]_s \otimes c_2$, then the space s *sees* the information c_1.

The rule $\mathrm{R_{SCH}}$ allows the process $[P]^s_{s'}$ to *choose* one possible sub-space l to execute P inside l. This intuitively means that the process P can *move* to any space in the hierarchy of spaces starting in s and ending in s'.

To explain the rule $\mathrm{R_S}$, consider the process $[\textbf{tell}(A)]_s$. What we observe from this process is that the constraint $[A]_s$ is added to the store. This means that the output of $\textbf{tell}(A)$ is confined to the space s. Now consider the process

$$\frac{}{\langle \overline{x}; \overline{\ell}; \mathbf{tell}(c), \Gamma; d\rangle \longrightarrow \langle \overline{x}; \overline{\ell}; \Gamma; c \otimes d\rangle}\ \mathrm{R_T}$$

$$\frac{\left\langle \overline{x}; \overline{\ell}_x; \Gamma; c) \cong (\overline{x}'; \overline{\ell}'_x; \Gamma'; c' \right\rangle \longrightarrow \left\langle \overline{y}'; \overline{\ell}'_y; \Delta'; d' \right\rangle \equiv \langle \overline{y}; \overline{\ell}_y; \Delta; d\rangle}{\langle \overline{x}; \overline{\ell}_x; \Gamma; c) \longrightarrow (\overline{y}; \overline{\ell}_y; \Delta; d\rangle}\ \mathrm{R_{EQUIV}}$$

$$\frac{\overline{y} \cap fv(\overline{x}, \overline{\ell}, d, \Gamma) = \overline{\ell}_y \cap fv(\overline{x}, \overline{\ell}, d, \Gamma)) = \emptyset}{\langle \overline{x}; \overline{\ell}; (\mathbf{local}\, \overline{y}; \overline{\ell}_y)\, P, \Gamma; d\rangle \longrightarrow \langle \overline{x} \cup \overline{y}; \overline{\ell} \cup \overline{\ell}_y; P, \Gamma; d\rangle}\ \mathrm{R_L}$$

$$\frac{d \vdash_\Delta \exists \overline{z}.(c[\overline{t}/\overline{y}][\overline{\ell}_t/\overline{\ell}_y] \otimes e) \quad \star}{\langle \overline{x}; \overline{\ell}; (\mathbf{abs}\ \overline{y}; \overline{\ell}_y; c)\, P, \Gamma; d\rangle \longrightarrow \langle \overline{x} \cup \overline{z}; \overline{\ell}; P[\overline{t}/\overline{y}][\overline{\ell}_t/\overline{\ell}_y], \Gamma; e\rangle}\ \mathrm{R_A}$$

$$\frac{s' \preceq l \preceq s, \quad s \notin S_U, \star\star}{\langle \overline{x}; \overline{\ell}; [P]^s_{s'}, \Gamma; d\rangle \longrightarrow \left\langle \overline{x}'; \overline{\ell}'; [P']_l, \Gamma'; d' \right\rangle}\ \mathrm{R_{SCH}} \qquad \frac{s' \preceq l \preceq s, \quad s \in S_U, \star\star}{\langle \overline{x}; \overline{\ell}; [P]^s_{s'}, \Gamma; d\rangle \longrightarrow \left\langle \overline{x}'; \overline{\ell}'; [P']_l, [P]^s_{s'}, \Gamma'; d' \right\rangle}\ \mathrm{R_{CPY}}$$

$$\frac{\langle \overline{x}; \overline{\ell}; P, \Gamma; d^s\rangle \longrightarrow \left\langle \overline{x}'; \overline{\ell}'; P', \Gamma'; d' \right\rangle}{\langle \overline{x}; \overline{\ell}; [P]_s, \Gamma; d\rangle \longrightarrow \left\langle \overline{x}'; \overline{\ell}'; [P']_s, \Gamma; d \otimes \bigtriangledown^s_s d' \right\rangle}\ \mathrm{R_S} \qquad \frac{p(\overline{x}) \stackrel{\mathsf{def}}{=} P}{(X; p(\overline{y}), \Gamma; d) \longrightarrow (X; P[\overline{y}/\overline{x}], \Gamma; d)}\ \mathrm{R_C}$$

Figure 6: Structural Operational Semantics for $\mathcal{M}$ccp. $fv(\cdot)$ denotes the set of free variables (first-order variables and location variables). In $\mathrm{R_L}$, $fv(\overline{x}, \overline{\ell}, d, \Gamma)$ means $\overline{x} \cup \overline{\ell} \cup fv(d) \cup fv(\Gamma)$. The side condition $\star$ in rule $\mathrm{R_A}$ is $\overline{z} \cap fv(\overline{x}, \Gamma, d) = \emptyset, mgc(e, \overline{t})$, i.e., e is the most general choice (Definition 4.3). The operators $\bigtriangledown^s_s d'$ and d^s in Rule $\mathrm{R_S}$ are in Definition 4.4. The side condition $\star\star$ *is* $\left\langle \overline{x}; \overline{\ell}; [P]_l, \Gamma, d \right\rangle \longrightarrow \left\langle \overline{x}'; \overline{\ell}'; [P']_l, \Gamma', d' \right\rangle$

[**ask** c **then** $Q]_s$. In this case, to decide if Q must be executed, we need to infer whether c can be deduced from the information available at location s. Hence, the premise of Rule $\mathrm{R_S}$ considers only the store d^s. Moreover, all the information produced by Q is confined to the space s ($\bigtriangledown^s_s d'$).

Rule $\mathrm{R_{CPY}}$ is similar to $\mathrm{R_{SCH}}$ but it applies for unbounded locations where a process P can be copied (replicated) as many times as needed.

Finally, rule $\mathrm{R_C}$ simply unfolds the definition of the process name p.

Definition 4.5 (Observables)**.** *Let $\longrightarrow^*$ be the reflexive and transitive closure of $\longrightarrow$ and c be a constraint without occurrences of spatial variables. If $\langle \overline{x}; \overline{\ell}; \Gamma; d\rangle \longrightarrow^* \left\langle \overline{x}'; \overline{\ell}'; \Gamma'; d' \right\rangle$ and $\exists \overline{x}' d' \vdash_\Delta c$ we write $\langle \overline{x}; \overline{\ell}; \Gamma; d\rangle \Downarrow_c$. If $\overline{x} = \overline{\ell} = \emptyset$ and $d = 1$ we simply write $\Gamma \Downarrow_c$. Intuitively, if P is a process then $P \Downarrow_c$ captures the outputs of P (under input 1).*

4.4. Programming in $\mathcal{M}$ccp

In this section we show some examples of distributed and concurrent behaviors that can be modeled in $\mathcal{M}$ccp. We also show how the interplay of **local** and **abs** processes allows us to dynamically create private or shared stores among agents.

Example 4.5 (Local stores)**.** *Let a and a' be bounded subexponentials, representing two different agents. Let also $P =$ **tell**(c), $Q =$ **ask** c **then** **tell**(d) and*

$R = [P]_a \parallel [Q]_{a'}$. It is easy to see that

$$\langle \emptyset; \emptyset; R, 1\rangle \longrightarrow \langle \emptyset; \emptyset; [Q]_{a'}; [c]_a\rangle \not\longrightarrow$$

Intuitively, Q remains blocked since the information c is only available for the agent (space) a.

Now let $R = [P]_a \parallel [Q]_a$. Then, we observe a derivation of the form

$$\langle \emptyset; \emptyset; R, 1\rangle \longrightarrow \langle \emptyset; \emptyset; [Q]_a, [c]_a\rangle \longrightarrow^* \langle \emptyset; \emptyset; \emptyset, [d]_a\rangle$$

This means that Q consumed the information c to add d to its local store.

Finally, consider $R = [[P]_{a'}]_a \parallel [Q]_a$. In this case, we observe a derivation of the shape:

$$\langle \emptyset; \emptyset; R, 1\rangle \longrightarrow \langle \emptyset; \emptyset; [Q]_a; [c]_{a.a'}\rangle \not\longrightarrow$$

As the information c is added to the nested *space a' in a, the process Q cannot deduce c in the space a.*

Example 4.6 (Sharing Information)**.** *Let a and a' be as in the previous example and consider the following processes:*

$$\begin{array}{lcl} R & = & (\mathbf{local}\, l : \{a, a'\})\ (P_A \parallel P_{A'}) \\ P_A & = & \mathbf{tell}([c]_l) \\ P_{A'} & = & \mathbf{ask}\ [c]_l\ \mathbf{then}\ Q \end{array}$$

The process R creates a share space between the agents a (resp. a') and it can move to a configuration of the shape $\langle \emptyset; \emptyset; Q; 1\rangle$ where $P_{A'}$ consumed c in the space l to latter execute Q.

Now consider an unbounded location a and the process

$$P = (\mathbf{local}\, l : \{a\})\, (\mathbf{local}\, l' : \{l\})\ (\mathbf{tell}([c]^a_{l'}) \parallel \mathbf{tell}([d]_a))$$

The process P creates a sub-space l (directly below a) and a sub-space l' of l. We then observe as final configuration

$$\langle \emptyset; l : \{a\}, l' : \{l\}; \emptyset; [c]^a_{l'} \otimes [d]_a\rangle$$

This means that the information c can be deduced in all spaces dominated *by a (i.e., those with type $\{a\}$) which means that c is also available in the spaces l and l'. Moreover, the information d is confined to the top level space a.*

When local spaces are created, one should pay attention to the possibly nested spaces generated by processes of the form $[P]_a$ as shown below.

Example 4.7 (Nested locations)**.** *Consider the following process*

$$P = (\mathbf{local}\, l : \{a\})\ ([[\mathbf{tell}(c)]_l]_a \parallel [\mathbf{tell}(d)]_l)$$

P evolves to a configuration of the shape $\gamma = \langle \emptyset; l : \{a\}; \emptyset; [c]_{a.l} \otimes [d]_l\rangle$. Notice, however, that the constraint c cannot be added to $a.l$ since this location is not

$$
\begin{array}{ll}
 & \langle \emptyset; \emptyset; \mathbf{request}(a,b) \parallel \mathbf{accept}(a,b); 1\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; \mathbf{request}(a,b) \parallel \mathbf{accept}(a,b); 1\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; \mathbf{tell}([\mathtt{com}(x)]_b) \parallel \mathbf{ask}\ [\mathtt{com}(x)]_a\ \mathbf{then}\ (\mathbf{tell}([\mathtt{com}(x)]_l) \parallel P) \parallel \mathbf{accept}(a,b); 1\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; \parallel \mathbf{ask}\ [\mathtt{com}(x)]_a\ \mathbf{then}\ (\mathbf{tell}([\mathtt{com}(x)]_l) \parallel P) \parallel \mathbf{accept}(a,b); [\mathtt{com}(x)]_b\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; \mathbf{ask}\ [\mathtt{com}(x)]_a\ \mathbf{then}\ (\mathbf{tell}([\mathtt{com}(x)]_l) \parallel P) \parallel \\
 & \qquad\qquad (\mathbf{tell}([\mathtt{com}(y)]_a) \parallel (\mathbf{abs}\ k:\{b\}; [\mathtt{com}(y)]_k)\, Q)\, [x/y]\, ; 1\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; \mathbf{ask}\ [\mathtt{com}(x)]_a\ \mathbf{then}\ (\mathbf{tell}([\mathtt{com}(x)]_l) \parallel P) \parallel (\mathbf{abs}\ k:\{b\}; [\mathtt{com}(y)]_k)\, Q; [\mathtt{com}(x)]_a\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; (\mathbf{tell}([\mathtt{com}(x)]_l) \parallel P) \parallel (\mathbf{abs}\ k:\{b\}; [\mathtt{com}(x)]_k)\, Q; 1\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; P \parallel (\mathbf{abs}\ k:\{b\}; [\mathtt{com}(y)]_k)\, Q; [\mathtt{com}(x)]_l\rangle \\
\longrightarrow^* & \langle x; l:\{a,b\}; P \parallel Q\, [l/k]\, ; 1\rangle
\end{array}
$$

Figure 7: Transitions of the system in Example 4.8.

defined in the configuration. The problem is that the process P is intended to add d to the new space l and c to a location that is nested *in a, which cannot be the same location l. Therefore, we cannot apply any rule to the configuration γ above. This problem can be solved by correctly writing P, for instance, as*

$$P = (\mathbf{local}\, l:\{a\})\, (\mathbf{local}\, a.l:\{a\})\ ([[\mathbf{tell}(c)]_l]_a \parallel [\mathbf{tell}(d)]_l)$$

It is worth noticing that $a.l$ is not dominated by l (i.e., $a.l \not\preceq l$). The spaces $a.l$ and l are completely different and information does not flow among them. If one wants to establish a connection between these spaces, it is sufficient to declare $a.l$ of type $\{a,l\}$ or $\{l\}$.

In the following example we show how to create shared spaces of communication as those in Example 4.5, but following a protocol where an agent sends a request and the other needs to accept such request to establish the shared store.

Example 4.8 (Name/Space Mobility)**.** *Name and space mobility is obtained in $\mathcal{M}$*`ccp` *by the interplay of* **abs** *and* **local** *processes. This allows processes to dynamically establish and communicate new shared variables and locations. Hence, we do not change the structure of agents but we* reconfigure *the communication structure of the system. Assume for instance an uninterpreted predicate symbol* $\mathtt{com}(\cdot)$ *and two linear spaces a and b (for Alice and Bob, respectively). Let us define the following shortcuts:*

$$
\begin{array}{lcl}
\mathbf{request}(a,b) & \overset{\text{def}}{=} & (\mathbf{local}\, x, l:\{a,b\})\, (\mathbf{tell}([\mathtt{com}(x)]_b) \parallel \mathbf{ask}\ [\mathtt{com}(x)]_a\ \mathbf{then}\ (\mathbf{tell}([\mathtt{com}(x)]_l) \parallel P)) \\
\mathbf{accept}(a,b) & \overset{\text{def}}{=} & (\mathbf{abs}\ y:b; [\mathtt{com}(y)]_b)\, (\mathbf{tell}([\mathtt{com}(y)]_a) \parallel (\mathbf{abs}\ k:b; [\mathtt{com}(y)]_k)\, Q)
\end{array}
$$

The behavior of the agent A (resp. B) is defined by the process $\mathbf{request}(a,b)$ *(resp.* $\mathbf{accept}(a,b)$*). The transitions for this system are depicted in Figure 7. The process* $\mathbf{request}(a,b)$ *creates a new location l of type $\{a,b\}$ and a fresh variable x. Then it "sends"* $\mathtt{com}(x)$ *to B by adding the constraint* $[\mathtt{com}(x)]_b$*. After that, agent B consumes this information and sends back to A the constraint* $\mathtt{com}(x)$*. Then A sends again the constraint* $\mathtt{com}(x)$ *but using the new established private space l. Due to the* **abs** *process, agent B is able to read* $\mathtt{com}(x)$ *on the space l. In the end, we observe that P and Q may use the new space l as a shared store.*

Before we go any further, let us note that some processes built from Definition 4.2 may not adhere to the design criterion (i) of $\mathcal{M}$`ccp`. For instance, assume that the agent A in the previous example contains a sub-term of the form $(\mathbf{abs}\ l : b; [c]_l)\, P$. In this case, A will query *all the spaces* in the store of B, and it can possibly consume information from it. Hence, agent A was able to directly read the store of another agent. A similar situation occurs if the agent A contains a sub-term of the form $[P]_b$, thus allowing to *execute* the process P in the space of computation of B. On the other side, a sub-term in A of the form $[\mathbf{tell}(c)]_b$ or $\mathbf{tell}([c]_b)$ do not seem to be problematic since it can be understood as an asynchronous communication between A and B.

In order to avoid these undesired behaviors, we can simply impose syntactic restrictions on the processes and constraints agents can tell and ask. For instance, it seems natural to think that agents can only ask constraints in their own hierarchy of spaces. Similar for processes of the form $[P]_s$. More involved mechanisms, such as type systems, can be also considered for this purpose (see [33]). Nevertheless, defining fragments of $\mathcal{M}\mathtt{ccp}$ that may exhibit some particular behaviors is completely orthogonal to our developments and we leave this task as future work. We also note that a similar situation occurs in the specification of security protocols, using, *e.g.*, multiset rewriting languages [34], where nonces are created. The rewrite language allows in principle for the specification of an agent that has access to any generated nonces. This is avoided, however, by using sensible protocol theories and intruder theories.

We finish this section by showing how processes can add information with a given preference.

Example 4.9 (Preferences)**.** *Let us consider the probabilistic c-semiring (see Section 4.1) and two spatial locations s and s'. Consider the following processes*

$$\begin{array}{rcl} P & = & \mathbf{tell}([(\!|c|\!)_{0.5} \otimes (\!|c|\!)_{0.3}]_s) \parallel \mathbf{tell}([(\!|c|\!)_{0.7} \otimes (\!|c|\!)_{0.6}]_{s'}) \parallel [Q]_s \parallel [Q]_{s'} \\ Q & = & \mathbf{ask}\ (\!|c \otimes d|\!)_{0.3}\ \mathbf{then}\ Q' \end{array}$$

The process P adds the same information to s and s' but with different preferences. We note that $(\!|c|\!)_{0.7} \otimes (\!|c|\!)_{0.6} \vdash_\Delta (\!|c \otimes d|\!)_a$ when $a \leq 0.42$ and $(\!|c|\!)_{0.5} \otimes (\!|c|\!)_{0.3} \vdash_\Delta (\!|c \otimes d|\!)_a$ when $a \leq 0.15$. Hence, Q' is only executed in the space s' where the probability of believing c and d is higher.

4.5. Logical Characterization of Processes

In [1] we showed a strong adequacy result, at the level of derivations, between SELL$^{\Cap}$ and different flavors of CCP, namely, epistemic, spatial and timed CCP. Here we extend the encodings presented in [1] to consider the processes $(\mathbf{local}\,\overline{\ell})\, Q$ and $(\mathbf{abs}\ \overline{\ell}; c)\, Q$. As expected, those processes will correspond, respectively, to formulas of the shape $\Cup \ell.F$ and $\Cap \ell.F$ where F corresponds to the encoding of Q. Following also the developments in [1], we shall consider three disjoint copies of the sub-exponential structure: $\mathfrak{c}$ to mark constraints, $\mathfrak{p}$ to mark processes and $\mathfrak{d}$ to mark procedure calls. Intuitively, for all $s, s' \in S$, the subexponentials $\mathfrak{c}(s)$, $\mathfrak{p}(s)$ and $\mathfrak{d}(s)$ are unrelated and they are unbounded if and only if s is unbounded; moreover, if $s' \preceq s$ then $\mathfrak{c}(s) \preceq \mathfrak{c}(s')$ (similarly for $\mathfrak{p}$ and $\mathfrak{d}$).

We begin by building a $\times$-poset (Definition 2.1) from a **mcs**. Then, we encode the stores (constraints) produced by processes.

Definition 4.6 ($\times$-poset from a **mcs**). *Let $(S, \mathcal{A}, \mathcal{C}, \vdash_\Delta)$ be a constraint system. Let us extend S to S' with two distinguished elements $\{\infty, nil\}$ such that ∞ (resp. nil) is the top (resp. bottom) of S' (i.e., S' is a bounded poset) and ∞ is unbounded. We shall define the $\times$-poset $\langle A, \leq, \times\rangle$ where $A = S' \cup \mathcal{A} \cup \{\bot, \top\}$, elements of $\mathcal{A}$ are unbounded and $\leq$ is the least relation containing $\preceq_{S'}$ and $\preceq_{\mathcal{A}}$ such that $\bot \leq s \leq \top$ for all $s \in A$. Moreover, $s \times s' = s \times_{\mathcal{A}} s'$ if $s, s' \in \mathcal{A}$. If $s, s' \in S'$, $s \times s' = glb(s, s')$ if it exists and $s \times s' = nil$ otherwise. In any other case, $s \times s' = \bot$.*

Intuitively, given two preferences (i.e., elements in $\mathcal{A}$), we combine information by using the $\times_{\mathcal{A}}$ operator of the c-semiring $\mathcal{A}$. Given two spatial locations s, s', $s \times s' = s$ iff $s \preceq_S s'$. Finally, $s \times a = \bot$ if $s \in S$ and $a \in \mathcal{A}$.

Definition 4.7 (Representation of Constraints). *Let $(S, \mathcal{A}, \mathcal{C}, \vdash_\Delta)$ be a* **mcs**, *c be a constraint and $\nabla_{\overline{s'}}^{\overline{s}}\, c$ be as in Definition 4.4 where $\overline{s}$,$\overline{s'}$ are sequences of elements in S (i.e., spatial locations). We shall define the encoding $\mathcal{C}[\![c]\!]_{\overline{s}}$ as the SELLS$^{\Cap}$ formula resulting from $\nabla_{\overline{s}}^{\overline{s}}\, c$ by replacing:*

- **Spaces***: $[c]_{\overline{s'}}^{\overline{s}}$ with $!^{\mathsf{c}(\overline{s})}?^{\mathsf{c}(\overline{s'})}c$; and*
- **Preferences***: $[(\!|PC|\!)_a]_{\overline{s'}}^{\overline{s}}$ with $!^{\mathsf{c}(\overline{s})}?^{\mathsf{c}(\overline{s'})}!^{\mathsf{c}(a)}PC$.*

  Moreover, an axiom in Δ of the form $\forall \overline{x}[c \multimap c']$ is encoded as

- **Axioms:** $!^{\mathsf{c}(\infty)}\Cap l : \infty.(\forall \overline{x}.(\mathcal{C}[\![c]\!]_l \multimap \mathcal{C}[\![c]\!]_l))$

We shall use $\mathcal{C}[\![\Delta]\!]$ to denote the encoding of all the axioms in Δ. The subexponential signature Σ is built from $(S, \mathcal{A}, \mathcal{C}, \vdash_\Delta)$ as in Definition 4.6.

Roughly, a formula of the shape $!^{\mathsf{c}(\overline{s})}?^{\mathsf{c}(\overline{s'})}c$ means that c holds in the space $\overline{s}$ and this information is confined up to the subspace $\overline{s}'$ (see properties in Proposition 4.1). Similarly, the subexponential $!^{\mathsf{c}(a)}$ allows us to mark formulas with a given preference $a \in \mathcal{A}$, which is unbounded. Finally, we note that axioms are available in any space in the system i.e., marked with the higher (and unbounded) subexponential ∞.

Next definition gives meaning to $\mathcal{M}\mathtt{ccp}$ processes as SELL$^{\Cap}$ formulas.

Definition 4.8 (Logical view of Processes). *Let P be a process and $\overline{s}$ be a sequence of spatial locations. We define the encoding $\mathcal{P}[\![\cdot]\!]_{\overline{s}}$ as $\mathcal{P}[\![p(\overline{x})]\!]_{\overline{s}} = !^{\mathsf{d}(\overline{s})}p(\overline{x})$ and $\mathcal{P}[\![P]\!]_{\overline{s}} = !^{\mathsf{p}(\overline{s})}\mathcal{P}'[\![P]\!]_{\overline{s}}$ where:*

- $\mathcal{P}'[\![\mathbf{tell}(c)]\!]_{\overline{s}} = \mathcal{C}[\![c]\!]_{\overline{s}}$
- $\mathcal{P}'[\![(\mathbf{abs}\ \overline{x}; \overline{\ell}; c)\ P]\!]_{\overline{s}} = \forall \overline{x}.\Cap \overline{\ell}.\ (\mathcal{C}[\![c]\!]_{\overline{s}} \multimap \mathcal{P}[\![P]\!]_{\overline{s}})$
- $\mathcal{P}'[\![(\mathbf{local}\,\overline{x}; \overline{\ell})\ P]\!]_{\overline{s}} = \exists \overline{x}.\Cup \overline{\ell}.\mathcal{P}[\![P]\!]_{\overline{s}}$

- $\mathcal{P}'[\![P_1, ..., P_n]\!]_{\overline{s}} = \mathcal{P}[\![P_1]\!]_{\overline{s}} \otimes ... \otimes \mathcal{P}[\![P_n]\!]_{\overline{s}}$
- $\mathcal{P}'[\![[P]^{s_1}_{s_2}]\!]_{\overline{s}} = \Cap l : s_1/s_2.\mathcal{P}'[\![[P]_l]\!]_{\overline{s}}$ *if* P *is an* **abs** *process and* $\mathcal{P}'[\![[P]^{s_1}_{s_2}]\!]_{\overline{s}} = \Cap l : s_1/s_2.\mathcal{P}[\![[P]_l]\!]_{\overline{s}}$
- $\mathcal{P}[\![[P]_{s'}]\!]_{\overline{s}} = \mathcal{P}'[\![P]\!]_{\overline{s}.s'}$ *otherwise.*

Moreover, a process definition $p(\overline{x}) \stackrel{\Delta}{=} P$ *is encoded as:*

$$!^{\mathfrak{d}(\infty)} \Cap l : \infty.\forall \overline{x}.(!^{\mathfrak{d}(l)} p(\overline{x}) \multimap \mathcal{P}[\![P]\!]_l)$$

We use $\mathcal{P}[\![\Upsilon]\!]$ *to denote the encoding of the process definitions in the set* Υ*.*

Let us give some intuition. The encoding of any process is a formula of the shape $!^{\mathfrak{p}(l)}F$. This means that every process is marked with a subexponential of the type $\mathfrak{p}(\cdot)$. As usual, *ask* agents are mapped as formulas of the shape $F \multimap G$. Here we use universal quantification on variables and locations to accurately represent the behavior of **abs** processes. For the **local** process, as expected, we use existential quantification (on variables and locations). Parallel composition is identified with conjunction of formulas. The call to a procedure in a hierarchy of spaces $\overline{s}$ is simply a formula of the shape $!^{\mathfrak{d}(\overline{s})}p(\overline{x})$ and the formula $!^{\mathfrak{d}(\infty)} \Cap l : \infty.\forall \overline{x}.(!^{\mathfrak{d}(l)} p(\overline{x}) \multimap \mathcal{P}[\![P]\!]_l)$, encoding the process definition, is able to unfold the body $\mathcal{P}[\![P]\!]_l$.

The most interesting cases are those involving the process $Q = [P]^s_{s'}$. Remember that, operationally, Q must choose a location l in the hierarchy $s' \prec s$ to execute P. The universal quantifier $\Cap l : s/s'$ allows us to do that. We note that the side condition $\star\star$ in Rules $\mathrm{R_{SCH}}$ and $\mathrm{R_{CPY}}$ (Figure 6) requires that the process P may exhibit one transition. Then, special attention must be paid in the encoding $\mathcal{P}[\![Q]\!]_{\overline{s}}$ when P is an ask agent (which is the only process that blocks in CCP). To better illustrate this situation, let $P = $ **ask** c **then** R and consider a focus derivation in SELL$^{\Cap}$ where we decide to focus on

$$\mathcal{P}[\![Q]\!]_{\overline{s}} = !^{\mathfrak{p}(\overline{s})} \left(\Cap l : \{s/s'\}.\mathcal{P}'[\![\textbf{ask } c \textbf{ then } R]\!]_{\overline{s}.l}\right)$$

Hence, the focusing persists on the quantifier $\Cap l$ and later on the the formula $\mathcal{C}[\![c]\!]_{\overline{s}.l} \multimap \mathcal{P}[\![R]\!]_{\overline{s}.l}$ which is also positive. This means that c must be "immediately" deduced from the context (see proof of Theorem 4.1). Note that, in the encoding $\mathcal{P}'[\![[P]^s_{s'}]\!]_{\overline{s}}$, we use again the encoding $\mathcal{P}'[\![\cdot]\!]$ instead of $\mathcal{P}[\![\cdot]\!]$ for the process $[P]_l$. Otherwise, we would obtain a formula of the shape $!^{\mathfrak{p}(\overline{s})}(\mathcal{C}[\![c]\!]_{\overline{s}} \multimap \mathcal{P}[\![R]\!]_{\overline{s}})$ that introduces the exponential "$!^{\mathfrak{p}(\overline{s})}$" and then focusing will be lost.

Finally, the last rule in the above definition allows us to observe the execution of P when the sub-location l is chosen.

Theorem 4.1 (Adequacy)**.** *Let* P *be a* $\mathcal{M}\mathtt{ccp}$ *process,* $(S, \mathcal{A}, \mathcal{C}, \vdash_\Delta)$ *be an constraint system,* Ψ *be a set of process definitions, and* $\mathcal{C}[\![c]\!]$, $\mathcal{P}[\![P]\!]$ *be as in Definitions 4.8 and 4.7. Then* $P \Downarrow_c$ *iff* $!^{\mathfrak{c}(\infty)}[\![\Delta]\!], !^{\mathfrak{p}(\infty)}[\![\Psi]\!], \mathcal{P}[\![P]\!]_{nil} \longrightarrow \mathcal{C}[\![c]\!]_{nil} \otimes \top$.[2]

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

Proof. The proof follows the proof technique in [1] and relies on completeness of the focusing strategy. Assume a $\mathcal{M}\texttt{ccp}$ configuration $\langle \overline{x}; \overline{\ell}; \Gamma, d \rangle$ which is encoded by a sequent of the form:

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

The shape of the above sequent can be obtained by 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

$$[\mathcal{C}_U, \mathcal{D}_U, \mathcal{P}_U : \mathcal{C}_L, \mathcal{D}_L, \mathcal{P}_L : \cdot] \longrightarrow [G]$$

where the contexts $\mathcal{K}$ and $\mathcal{L}$ are split into three contexts each: $\mathcal{C}_U, \mathcal{D}_U$ and $\mathcal{P}_U$, and $\mathcal{C}_L, \mathcal{D}_L$ and $\mathcal{P}_L$, containing all formulas marked, respectively, with bangs of the $\mathfrak{c}, \mathfrak{d}$ and $\mathfrak{p}$ types.

Let us consider the case of the ask agent in [1]. We know that

$$\mathcal{P}[\![\textbf{ask } c \textbf{ then } P]\!]_\ell = !^{\mathfrak{p}(\ell)}(\mathcal{C}[\![c]\!]_\ell \multimap \mathcal{P}[\![P]\!]_\ell)$$

is in the context. We show the derivation obtained by focusing on this formula when $\mathfrak{p}(\ell)$ is unbounded and $(\mathcal{C}[\![c]\!]_\ell \multimap \mathcal{P}[\![P]\!]_\ell) \in \mathcal{P}_U[\mathfrak{p}(\ell)]$. The case when it is bounded is similar, but where the modified context is the $\mathcal{P}_L$.

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

where $\mathcal{L} = \mathcal{C}_L, \mathcal{D}_L, \mathcal{P}_L$. Notice that all formulas of the bounded context $\mathcal{L}$ are moved to the right premise. This is because $\mathcal{C}[\![c]\!]_\ell$ contains only positive formulas, and therefore, it will be totally decomposed resulting on a positive trunk with sequents of the form $[\mathcal{C}_U, \mathcal{D}_U, \mathcal{P}_U : \cdot] -_{\bigtriangledown_{\mathfrak{c}(\ell_i)} A}\!\!\rightarrow$. Hence the sequents obtained in π_1 will necessarily end with derivations of the form:

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

The important thing to notice is that the contexts $\mathcal{D}_U$ and $\mathcal{P}_U$ are necessarily weakened in the 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 $\mathcal{C}_{atom}$ in $\mathcal{C}$ and the theory Δ. That is, $\mathcal{C}_{atom} \vdash_\Delta A$. Finally, observe that formulas in $\mathcal{C}_{atom}$ are constraints, coming from *tells*. 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.

Now consider the formula $\mathcal{C}[\![(\mathbf{abs}\,\overline{x};\overline{\ell};c)\,]\!]_P = !^{\mathfrak{p}(\overline{s})}\left(\forall\overline{x}.\Cap\overline{\ell}.\,(\mathcal{C}[\![c]\!]_{\overline{s}} \multimap \mathcal{P}[\![P]\!]_{\overline{s}})\right)$. We note that $\forall$ and $\Cap$ must be introduced in a positive phase (just like the implication for the ask agent). Hence, what we observe is that in a single phase, the terms $\overline{t}$ and the locations $\overline{\ell}_t$ must be chosen in such a way that the formula $c[\overline{t}/\overline{x}][\overline{\ell}_t/\overline{\ell}]$ must be provable "immediately" for the constrains already in the context. This also corresponds exactly to the operational behavior.

Now consider to focus in the formula $\mathcal{P}[\![[P]^s_{s'}]\!]_{\overline{s}} = !^{\mathfrak{p}(\overline{s})}\,(\Cap l : s/s'.\mathcal{P}'[\![[P]_l]\!]_{\overline{s}})$. Focusing on this formula results necessarily in the following derivation, where $(\Cap l : s/s'.\mathcal{P}'[\![[P]_l]\!]_{\overline{s}}) \in \mathcal{P}_U[\mathfrak{p}(\ell)]$:

$$\cfrac{\cfrac{\overset{\pi}{\mathcal{S};[\mathcal{C}_U,\mathcal{D}_U,\mathcal{P}_U : \mathcal{L}] \xrightarrow{\mathcal{P}'[\![[P]_{s''}]\!]_{\overline{s}}} [G]}}{\mathcal{S};[\mathcal{C}_U,\mathcal{D}_U,\mathcal{P}_U : \mathcal{L}] \xrightarrow{\Cap l:s/s'.\mathcal{P}'[\![[P]_l]\!]_{\overline{s}}} [G]}\ \Cap_L}{\mathcal{S};[\mathcal{C}_U,\mathcal{D}_U,\mathcal{P}_U : \mathcal{L}] \longrightarrow [G]}\ D$$

where $s'' : s/s' \in \mathcal{S}$. Here we consider two cases.

- If P is of the shape **ask** c **then** Q, then the formula $\mathcal{P}'[\![P]\!]_{s''}$ is a positive formula and focusing cannot be lost. Then, the guard c must be immediately proved from the context to later introduce the encoding of Q in the context $\overline{s}.s'$. Similarly for the case when P is of the shape $(\mathbf{abs}\,\overline{x};\overline{\ell};c)\,Q$.

- If P is not an ask agent, the formula $\mathcal{P}'[\![P]\!]_{s''}$ is of the shape $!^{\mathfrak{p}(\overline{s}.s'')}F$. Then, focusing is lost in π and the encoding of P is stored in the context $\overline{s}.s''$ as required.

□

5. Concluding Remarks

In this paper we proposed a new proof system, called SELLS$^\Cap$, which includes novel subexponential quantifiers for linear logic with subexponentials. We show that not only a wide range of existing CCP languages can be specified in SELL$^\Cap$, as done in our previous works [1, 8], but that SELLS$^\Cap$ provide a logical framework for the development of new CCP languages with clear proof theoretic foundations. In particular, we have proposed a CCP calculus that combines and extends features from spatial CCP [3] and soft-constraints [11], allowing the dynamic creation of new spaces and the sharing of information. In order to prove these results, we have proposed a focused proof system for SELLS$^\Cap$ and proved its soundness and completeness. Our encodings of CCP processes and constraint stores have a strong adequacy, meaning that there are tight connections between focused derivations and CCP transitions. This paper, thus, continues the CCP tradition where logic and proof theory plays an important role in the specification of CCP languages.

Related Work. The first CCP language featuring soft constraints was proposed in [11]. There, c-semiring based constraints, seen as functions mapping variable assignments into c-semiring values, are lifted to a higher-order semiring where constraints can be combined and compared. In such formalization, an entailment relation à la Saraswat [2] can be defined only if the $\times_{\mathcal{A}}$ operator is idempotent (see [11, Def. 3.8, Th. 3.9]). In particular, given a set of constraints C, if $\times_{\mathcal{A}}$ is non-idempotent, $C \vdash d$ does not imply that $C \sqcup d \equiv C$. In our system, if $C \longrightarrow (\!|d|\!)_a$ then $(\bigotimes C \otimes (\!|d|\!)_a) \equiv (\bigotimes C)$ (regardless the idempotency of $\times$). Hence, our logical characterization of soft constraints as formulas in SELLS$^{⋒}$ follows closely the idea of monotonic store in CCP.

A model-based (semantic) characterization of soft constraints based on c-semirings is given in [35]. To the best of our knowledge, ours is the first proof-theoretic characterization of such systems [8]. However, the use of more involved orders for subexponentials is not completely new. They were used recently in different contexts, such as in Bounded Linear Logic [36] and in programming languages [37].

The logical framework literature has specified a number of distributed systems. For example, [16] proposes a concurrent logical framework based on intuitionistic linear logic (without subexponentials). It does not seem possible to capture the spatial properties (Proposition 4.1) in a declarative fashion in such a framework. The use of subexponentials and how they are organized is needed.

Finally, the use of more elaborate subexponential signature seems close to the work done by the Hybrid Logic literature [38]. This framework is similar to SELL as it also combines the use of standard logic (first-order logic) with modal operators. It is not clear, however, how Hybrid Logic compares with SELLS$^{⋒}$. In particular, [38] does not specify the types of systems that we specify, which include spatial and preferences as well as information sharing.

We are currently investigating methods for verifying systems specified in $\mathcal{M}\mathtt{ccp}$ which mention spatial properties, *e.g.*, the Airport Security problem [39]. We believe that linear logic together with the strong levels of adequacy may help us develop more general techniques for verifying CCP programs.

References

[1] 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 LNCS, Springer, 2013, pp. 410–424.

[2] 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.

[3] 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 LNCS, Springer, 2012, pp. 317–332.

[4] 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 LNCS, Springer, 1993, pp. 159–171.

[5] 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.

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

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

[8] E. Pimentel, C. Olarte, V. Nigam, A proof theoretic study of soft concurrent constraint programming, TPLP 14 (4-5) (2014) 649–663.

[9] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint logic programming: syntax and semantics, ACM Trans. Program. Lang. Syst. 23 (1) (2001) 1–29.

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

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

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

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

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

[15] V. Nigam, Exploiting non-canonicity in the Sequent Calculus, Ph.D. thesis, Ecole Polytechnique (Sep. 2009).

[16] I. Cervesato, F. Pfenning, D. Walker, K. Watkins, A concurrent logical framework II: Examples and applications, Tech. Rep. CMU-CS-02-102, Carnegie Mellon University, revised, May 2003 (2003).

[17] 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 LNCS, Springer, 2007, pp. 405–419.

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

[19] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, Parts I and II, Inf. Comput. 100 (1) (1992) 1–40.

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

[21] P. V. Hentenryck, V. A. Saraswat, Y. Deville, Design, implementation, and evaluation of the constraint language cc(fd), Journal of Logic Programming 37 (1-3) (1998) 139–164.

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

[23] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Vol. 2 of Foundations of Artificial Intelligence, Elsevier, 2006.

[24] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, J. ACM 44 (2) (1997) 201–236.

[25] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: IJCAI (1), Morgan Kaufmann, 1995, pp. 631–639.

[26] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, H. Fargier, Semiring-based csps and valued csps: Frameworks, properties, and comparison, Constraints 4 (3) (1999) 199–240.

[27] H. Fargier, J. Lang, Uncertainty in constraint satisfaction problems: a probalistic approach, in: M. Clarke, R. Kruse, S. Moral (Eds.), ECSQARU, Vol. 747 of Lecture Notes in Computer Science, Springer, 1993, pp. 97–104.

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

[29] D. Chiarugi, D. Hemith, M. Falaschi, C. Olarte, Verification of spatial and temporal modalities in biochemical systems, in: Proc. of the Fifth International Workshop on Static Analysis and Systems Biology, 2014.

[30] R. Haemmerlé, F. Fages, S. Soliman, Closures and modules within linear logic concurrent constraint programming, in: V. Arvind, S. Prasad (Eds.), FSTTCS, Vol. 4855 of LNCS, Springer, 2007, pp. 544–556.

[31] C. Olarte, F. D. Valencia, Universal concurrent constraint programing: symbolic semantics and applications to security, in: R. L. Wainwright, H. Haddad (Eds.), SAC, ACM, 2008, pp. 145–150.

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

[33] T. T. Hildebrandt, H. A. López, Types for secure pattern matching with local knowledge in universal concurrent constraint programming, in: P. M. Hill, D. S. Warren (Eds.), ICLP, Vol. 5649 of Lecture Notes in Computer Science, Springer, 2009, pp. 417–431.

[34] 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.

[35] N. Wilson, A logic of soft constraints based on partially ordered preferences, J. Heuristics 12 (4-5) (2006) 241–262.

[36] D. R. Ghica, A. Smith, From bounded affine types to automatic timing analysis, CoRR abs/1307.2473.

[37] A. Brunel, M. Gaboardi, D. Mazza, S. Zdancewic, A core quantitative coeffect calculus, in: Z. Shao (Ed.), ESOP, Vol. 8410 of Lecture Notes in Computer Science, Springer, 2014, pp. 351–370.

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

[39] B. Schneier, Schneier on security, Wiley, 2008.

[40] A. S. Troelstra, Lectures on linear logic, CSLI lecture notes, Center for the Study of Language and Information, Stanford, CA, 1992.
URL `http://opac.inria.fr/record=b1089180`

Appendix A. Soundness and completeness for the dyadic system

Theorem3.1. $\text{SELLS}^{\Cap d}$ is sound and complete with respect to $\text{SELLS}^{\Cap}$.

Proof. We only show the cases involving the promotion rules, which are new with respect to the cases shown in [15]. The proof by induction on the height of proofs.

Let $!^s\{F_1, \ldots, F_n\}$ denote the collection of formulas $!^sF_1, \ldots, !^sF_n$ and let Λ_U be the formulas $!^{u_1}\mathcal{K}[u_1], \ldots, !^{u_n}\mathcal{K}[u_n]$ for each $s \preceq_S u_i$ and Λ'_U be the set $!^{u'_1}\mathcal{K}[u'_1], \ldots, !^{u'_m}\mathcal{K}[u'_m]$ for each $s \not\preceq_S u_i$.

Consider the derivation in $\text{SELLS}^{\Cap d}$:

$$\dfrac{\overset{\Xi}{\mathcal{K} : \mathcal{L}_1 : \Gamma_1 \longrightarrow F} \quad \overset{\Xi_2}{\mathcal{K} : \mathcal{L}_2 : \Gamma_2 \longrightarrow G}}{\mathcal{K} : \mathcal{L}_1 \otimes \mathcal{L}_2 : \Gamma_1, \Gamma_2 \longrightarrow F \otimes G}\ \otimes_R$$

$$\begin{array}{c}
\Lambda_U, !^{b_1}\mathcal{L}_1[b_1], \ldots, !^{b_m}\mathcal{L}_1[b_m], \Gamma_1 \longrightarrow F \quad \Lambda_U, !^{b_1}\mathcal{L}_2[b_1], \ldots, !^{b_m}\mathcal{L}_2[b_m], \Gamma_2 \longrightarrow F \\
\hline
\Lambda_U, \Lambda_U, !^{b_1}(\mathcal{L}_1[b_1] \uplus \mathcal{L}_2[b_1]), \ldots, !^{b_m}(\mathcal{L}_1[b_m] \uplus \mathcal{L}_2[b_m]), \Gamma_1, \Gamma_2 \longrightarrow F \otimes G \\
\hline\hline
\Lambda_U, !^{b_1}(\mathcal{L}_1[b_1] \uplus \mathcal{L}_2[b_1]), \ldots, !^{b_m}(\mathcal{L}_1[b_m] \uplus \mathcal{L}_2[b_m]), \Gamma_1, \Gamma_2 \longrightarrow F \otimes G
\end{array}
\begin{array}{l} \otimes_R \\ C_L \end{array}$$

From the inductive hypothesis, we have that the premises of this derivation are provable if and only if their corresponding sequents in the derivation above in the dyadic system are provable, finishing the proof for this case. For soundness, we also use the Lemma 2.1 for handling the implicit contractions in the derivation of the dyadic system.

Now consider the derivation in $\text{SELLS}^{\Cap d}$:

$$\dfrac{\overset{\Xi}{\mathcal{K} \geq_s : \mathcal{L} : \cdot \longrightarrow F}}{\mathcal{K} : \mathcal{L} : \cdot \longrightarrow !^sF}\ !^s{}_R$$

This proof can be obtained in $\text{SELLS}^{\Cap}$ by simply weakening the formulas marked with $!^u$ such that $s \not\preceq_S u$, *i.e.*, the formulas in Λ'_U:

$$\begin{array}{c}
\overset{\Xi'}{\Lambda_U, !^{b_1}\mathcal{L}[b_1], \ldots, !^{b_p}\mathcal{L}[b_p] \longrightarrow F} \\
\hline
\Lambda_U, !^{b_1}\mathcal{L}[b_1], \ldots, !^{b_p}\mathcal{L}[b_p] \longrightarrow !^sF \\
\hline\hline
\Lambda_U, \Lambda'_U, !^{b_1}\mathcal{L}[b_1], \ldots, !^{b_p}\mathcal{L}[b_p] \longrightarrow !^sF
\end{array}\ W_L$$

where Ξ' is obtained from Ξ by using the inductive hypothesis. Notice that from the side condition of the rule $!^s{}_R$, $\mathcal{L}[b] = \emptyset$ for all $s \not\preceq_S b$.

The case for $!^s{}_{RS}$ is similar:

$$\dfrac{\overset{\Xi}{\mathcal{K}' : \mathcal{L} : \cdot \longrightarrow F}}{\mathcal{K} : \mathcal{L} : \cdot \longrightarrow !^sF}\ !^s{}_{RS}$$

This proof can be obtained in SELLS$^{\Cap}$ by simply weakening the formulas that do not appear in the premise of the $!^s{}_{RS}$ rule. Let Λ_U be the formulas $!^{u_1}\mathcal{K}'[u_1], \ldots, !^{u_n}\mathcal{K}'[u_n]$ and Λ'_U be the formulas $!^{u_1}\mathcal{K}[u_1] \setminus \mathcal{K}'[u_1], \ldots, !^{u_n}\mathcal{K}[u_n] \setminus \mathcal{K}'[u_n]$:

$$\dfrac{\dfrac{\begin{array}{c}\Xi' \\ \Lambda_U, !^{b_1}\mathcal{L}[b_1], \ldots, !^{b_p}\mathcal{L}[b_p] \longrightarrow F\end{array}}{\Lambda_U, !^{b_1}\mathcal{L}[b_1], \ldots, !^{b_p}\mathcal{L}[b_p] \longrightarrow !^a F}}{\Lambda_U, \Lambda'_U, !^{b_1}\mathcal{L}[b_1], \ldots, !^{b_p}\mathcal{L}[b_p] \longrightarrow !^s F}\ W_L$$

where Ξ' is obtained from Ξ by using the inductive hypothesis. $\square$

Appendix B. Proof of Theorem 2.2

The following remarks make formal the following intuitive idea: when substituting a subexponential variable l_e by a subexponential s of the same type, all the relations and properties valid for l_e are "inherited" by s.

Remark Appendix B.1. *In rules $\Cup_L$ and $\Cap_R$, the premise signature $\mathcal{S}' = \{\mathcal{S}, l_e : \{s_1, \ldots, s_n\}\}$ is such that, by construction, there is no subexponential $s' \neq \bot$ such that $s' \prec_{S'} l_e$ or $l_e \prec_{S'} s' \prec_{S'} s_j$, for any $1 \leq j \leq n$.*

Let $s \neq l_e$ be a subexponential satisfying the condition $(\star 1)$. We claim that if $l_e \preceq_{S'} glb(d_1, d_2)$, with $d_1, d_2 \neq l_e$, then $s_j \preceq_{S'} glb(d_1, d_2)$ for some $1 \leq j \leq m$ (that is, $l_e \prec_{S'} glb(d_1, d_2)$). In fact, if $n = 1$ then the result is trivial and if $n > 1$, $glb(d_1, d_2) \neq l_e$, since s and l_e are not related and there exists $1 \leq j, k \leq n$ such that $s_j \preceq_{S'} d_1$ and $s_k \preceq_{S'} d_2$, that is, s is also a lower bound of d_1, d_2 and the result holds. In other words, if $l_e \preceq_{S'} glb(d_1, d_2)$, then $s \preceq_{S'} glb(d_1, d_2)$, since $s \preceq_{S'} s_j$ for all $1 \leq j \leq n$. This base case shows that l_e can always *be substituted by s when no new relations on l_e have been created.*

Remark Appendix B.2. *Let $l_e, s, \mathcal{S}'$ be as in Remark Appendix B.1, and $l : \{s_j/l_e\}$, that is, l is created between l_e and s_j, with $l_e \prec_{S''} s_j$, where $\{\mathcal{S}', l : \{s_j/l_e\}\} \subseteq \mathcal{S}''$. Hence, when substituting l_e by s, the type of l will turn to be $\{s_j/s\}$. Observe that, since $s \preceq_{S''} s_j$, cycles could be created with this substitution. This is ruled out by $(\star 3)$ as explained in Remark Appendix B.4. Inductively, if $l : \{s''/s'\}$ is such that $l_e \preceq_{S''} s' \preceq_{S''} s'' \preceq_{S''} s_j$ then, when substituting l_e by s, the type of l will be well formed, i.e. $s \preceq_{S''} s' \preceq_{S''} s'' \preceq_{S''} s_j$ with $s \prec_{S''} s_j$.*

Remark Appendix B.3. *Let $l_e, s, \mathcal{S}'$ be as in Remark Appendix B.1 and $\mathcal{S}' \subseteq \mathcal{S}''$. Suppose $d = glb(l_e, d_1)$, with $d \neq \bot$. Hence $d \preceq_{S''} l_e$. There are two possibilities: either $d = l_e$ or d is a subexponential variable of type $\{s'_1, \ldots, s'_m\}$ with $s'_j \preceq_{S''} l_e$ for some $1 \leq j \leq m$, or of type s''/s' with $s'' \preceq_{S''} l_e$. In any case, when substituting l_e by s, the type of d will continue being well formed: either $\{s'_1, \ldots, s'_m\}$ with $s'_j \preceq_{S''} s$ for some $1 \leq j \leq m$, or s''/s' with $s'' \preceq_{S''} s$. Hence, $glb(l_e, d_1)[s/l_e] = d$.*

Remark Appendix B.4. *Finally, the side condition ($\star 3$) is necessary for cut-elimination of SELLS$^{\Cap}$, which requires the relation $\preceq_S$ to be a pre-order. In fact, consider for example the formula $\Cap l : a/b.\Cap l' : b/l.F$. Once we introduce the first quantifier $\Cap l : a/b$, we create a fresh name l_e with typing $l_e : a/b$, which means that $b \preceq_{S'} l_e \preceq_{S'} a$, with $\mathcal{S}' = \mathcal{S} \cup l_e : a/b$. If we introduce the second quantifier, $\Cap l' : b/l_e$, we create another fresh subexponential l'_e, with typing $l'_e : b/l_e$. This means that $l_e \preceq_{S''} l'_e \preceq_{S''} b$ with $\mathcal{S}'' = \mathcal{S}' \cup l'_e : b/l_e$, obtaining thus a cycle. Notice that checking that the relation is a pre-order can be done in polynomial time with respect to the number of elements of the pre-order. Thus it is possible to check whether the rule is an instance of $\Cup_L$ or $\Cap_R$ in polynomial time.*

We observe that similar conclusions in Remarks Appendix B.1, Appendix B.2 and Appendix B.3 can be proved for l_e of type $\{s''/s'\}$.

We can now prove cut-elimination to SELLS$^{\Cap}$.

Theorem Appendix B.5. *The cut rule below is admissible in SELLS$^{\Cap}$.*

$$\frac{\mathcal{S}; \Gamma_1 \longrightarrow G \quad \mathcal{S}; \Gamma_2, G \longrightarrow F}{\mathcal{S}; \Gamma_1, \Gamma_2 \longrightarrow F}\ Cut$$

Proof. The proof follows the usual Gentzen cut-elimination procedure. We will fill in the details involving the introduction rules for the bang, as these are new. The remaining cases are similar to the cut-elimination proof for SELL$^{\Cap}$, see [1].

Permutation Lemmas. The first step is to show that any proof with cuts can be transformed into a proof of the same end-sequent but with only principal cuts. This is done by showing that the *Cut* rule permutes over the other rules, when the cut formula is not principal in one of the premises. In the case of the promotion rule, if $s : \tau \in \mathcal{S}$

$$\frac{\dfrac{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n \longrightarrow G}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n \longrightarrow !^{s}G}\ !^{s}{}_{R_S} \quad \dfrac{\mathcal{S}; !^{d_1}G_1, \ldots, !^{d_m}G_m, !^{s}G \longrightarrow F}{\mathcal{S}; !^{d_1}G_1, \ldots, !^{d_m}G_m, !^{s}G \longrightarrow !^{d}F}\ !^{d}{}_{R_S}}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n, !^{d_1}G_1, \ldots, !^{d_m}G_m \longrightarrow !^{d}F}\ Cut \quad \rightsquigarrow$$

$$\frac{\dfrac{\dfrac{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n \longrightarrow G}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n \longrightarrow !^{s}G}\ !^{s}{}_{R_S} \quad \mathcal{S}; !^{d_1}G_1, \ldots, !^{d_m}G_m, !^{s}G \longrightarrow F}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n, !^{d_1}G_1, \ldots, !^{d_m}G_m \longrightarrow F}\ Cut}{\mathcal{S}; !^{s_1}F_1, \ldots, !^{s_n}F_n, !^{d_1}G_1, \ldots, !^{d_m}G_m \longrightarrow !^{d}F}\ !^{d}{}_{R_S}$$

Note that the derivation above is possible since, from the left premise of the first derivation, $s \preceq_S s_1 \times \cdots \times s_n$ and, from the right premise of the same derivation, $d \preceq_S s \times d_1 \times \cdots \times d_m$. Thus by monotonicity, we have that $d \preceq s_1 \times \cdots \times s_n \times d_1 \times \cdots \times d_m$ and hence the last $!^{d}$ can be introduced.

Reduction to Atomic Cuts. The second step consists of exchanging non-atomic principal cuts into smaller ones, until getting to atomic cuts. The cases involving the quantifiers and/or bang introduction rules are:

- $⋒_{L1} + ⋒_R$. The reduction follows the same idea as for the first-order quantifiers. The deduction

$$\dfrac{\dfrac{\overset{\Xi}{\mathcal{S}, l_e : \{s_1, \ldots, s_n\}_i; \Gamma \longrightarrow F[l_e/l]}}{\mathcal{S}; \Gamma \longrightarrow ⋒ l : \{s_1, \ldots, s_n\}_i.F}\, ⋒_R \quad \dfrac{\overset{\Xi'}{\mathcal{S}; \Gamma, F[s/l] \longrightarrow G}}{\mathcal{S}; \Gamma, ⋒ l : \{s_1, \ldots, s_n\}_i.F \longrightarrow G}\, ⋒_{L1}}{\mathcal{S}; \Gamma \longrightarrow G}\, Cut$$

is replaced by

$$\dfrac{\overset{\Xi[s/l_e]}{\mathcal{S}; \Gamma \longrightarrow F[s/l]} \quad \overset{\Xi'}{\mathcal{S}; \Gamma, F[s/l] \longrightarrow G}}{\mathcal{S}; \Gamma \longrightarrow G}\, Cut$$

As pointed out in [4], for cut-elimination, one needs to be careful with the structural properties of subexponentials. We avoid such problems since, by conditions $(\star 1), (\star 2)$ and $(\star 3)$, l, s and l_e are either all bounded or all unbounded.

Moreover, we can show by induction that the object $\Xi[s/l_e]$ is indeed a SELLS$^{⋒}$ proof. The only interesting cases are when a $!^{s'}$ is introduced on the right and a $?^{s'}$ is introduced on the left, somewhere in Ξ. We show only the former, as the latter follows similarly.

Assume that the formula $!^{s'} H$ is introduced. Then the context is a set of the form $\{!^{d_1} H_1, \ldots, !^{d_m} H_m\}$ with $s' \preceq_{S'} d_1 \times \ldots \times d_m$ for some $\mathcal{S} \subseteq \mathcal{S}'$. Let $d_1 = a_1, \ldots, d_k = a_k \in A$ and $d_{k+1} = l_{k+1}, \ldots, d_m = l_m \notin A$. Hence $d = d_1 \times \ldots \times d_m = glb\{a, l_{k+1}, \ldots, l_m\}$ where $a = a_1 \times_\Sigma \ldots \times_\Sigma a_k \in A$. Thus we have to show that $s' \preceq_{S'} d$ is invariant under substitution, that is, $s'[s/l_e] \preceq_{S'} d[s/l_e]$. There are two subcases to consider:

  - Suppose $s' = l_e$. If $k = m$ (that is, there are no subexponential variables in the context) or if $l_j \in S, \forall\, k+1 \leq j \leq m$ (that is, no new subexponential variables are created), then $\mathcal{S} = \mathcal{S}'$ and $s \preceq_S d[s/l_e]$ by Remark Appendix B.1.

    If $l_j \notin \mathcal{S}$, for some $k+1 \leq j \leq m$, it means that l_j was created after l_e. By intensiveness, $l_e \preceq_S l_j$ which implies that l_j has the shape $l_j : s''/s'''$. By condition $(\star 3)$, it must be the case that $l_e \preceq_{S'} s''' \prec_{S'} s''$. Hence the result follows by Remark Appendix B.2.
  - Suppose $s' \neq l_e$. If $d_j \neq l_e$ for all $k+1 \leq j \leq m$, then the result follows trivially. On the other hand, if $d_j = l_e$ for some j then the result follows by Remark Appendix B.3.

- Promotion + dereliction

$$\dfrac{\dfrac{\mathcal{S}; \Gamma \longrightarrow G}{\mathcal{S}; \Gamma \longrightarrow !^s G}\, !^s{}_{R_S} \quad \dfrac{\mathcal{S}; \Delta, G \longrightarrow F}{\mathcal{S}; \Delta, !^s G \longrightarrow F}\, !^s{}_L}{\mathcal{S}; \Gamma, \Delta \longrightarrow F}\, Cut \rightsquigarrow \dfrac{\mathcal{S}; \Gamma \longrightarrow G \quad \mathcal{S}; \Delta, G \longrightarrow F}{\mathcal{S}; \Gamma, \Delta \longrightarrow F}\, Cut$$

- Promotion + weakening

$$\dfrac{\dfrac{\mathcal{S};\Gamma \longrightarrow G}{\mathcal{S};\Gamma \longrightarrow !^s G}\ !^s{}_{R_S} \qquad \dfrac{\Delta \longrightarrow F}{\mathcal{S};\Delta, !^s G \longrightarrow F}\ !^s{}_L}{\mathcal{S};\Gamma,\Delta \longrightarrow F}\ Cut \qquad \rightsquigarrow \qquad \dfrac{\mathcal{S};\Delta \longrightarrow F}{\mathcal{S};\Gamma,\Delta \longrightarrow F}\ W$$

  We can weaken Γ since applying the $!^s{}_{R_S}$ rule in the left premise forces Γ to have the shape $!^{s_1}F_1,\ldots,!^{s_n}F_n$, with $s \preceq_S s_1 \times \ldots \times s_n$. On the other hand, from the right-premise, s is unbounded, *i.e.*, formulas of the form $!^s F$ are allowed to contract and weaken. Since "being unbounded" is upwardly closed with respect to $\preceq_S$, we also have $s_1,\ldots,s_n$ unbounded. Thus $!^{s_1}F_1,\ldots,!^{s_n}F_n$ can also be weakened by Lemma 2.1.

- Promotion + contraction

$$\dfrac{\dfrac{\mathcal{S};\Gamma \longrightarrow G}{\mathcal{S};\Gamma \longrightarrow !^s G}\ !^s{}_{R_S} \qquad \dfrac{\mathcal{S};\Delta, !^s G, !^s G \longrightarrow F}{\mathcal{S};\Delta, !^s G \longrightarrow F}\ !^s{}_L}{\Gamma,\Delta \longrightarrow F}\ Cut \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\dfrac{\mathcal{S};\Gamma \longrightarrow G}{\mathcal{S};\Gamma \longrightarrow !^s G}\ !^s{}_{R_S} \qquad \dfrac{\dfrac{\mathcal{S};\Gamma \longrightarrow G}{\mathcal{S};\Gamma \longrightarrow !^s G}\ !^s{}_{R_S} \qquad \mathcal{S};\Delta, !^s G, !^s G \longrightarrow F}{\mathcal{S};\Delta,\Gamma, !^s G \longrightarrow F}\ Cut}{\mathcal{S};\Gamma,\Gamma,\Delta \longrightarrow F}\ Cut}{\mathcal{S};\Gamma,\Delta \longrightarrow F}\ C$$

Reduction of Atomic Cuts. The step to eliminate atomic cuts by permuting them upwards follows the same steps as in the cut-elimination procedure for SELL$^{\Cap}$.

Finally, it is also easy to check that the usual termination arguments used in Gentzen's cut-elimination also work here (see [40]). $\square$