Dynamic Spaces in Concurrent Constraint Programming

Carlos Olarte [1]

Departamento de Electrónica y Ciencias de la Computación
Pontificia Universidad Javeriana-Cali, Colombia
Escola de Ciências e Tecnologia
Universidade Federal do Rio Grande do Norte
Natal, Brazil

Vivek Nigam[2]

Departamento de Informática
Universidade Federal da Paraíba
João Pessoa, Brazil

Elaine Pimentel[3]

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

Abstract

Concurrent constraint programming (CCP) is a declarative model for concurrency where agents interact with each other by posting and asking constraints (formulas in logic) in a shared store of partial information. With the advent of emergent applications as security protocols, social networks and cloud computing, the CCP model has been extended in different directions to faithfully model such systems as follows: (1) It has been shown that a name-passing discipline, where agents can communicate local names, can be described through the interplay of local ($\exists$) processes along with universally ($\forall$) quantified asks. This strategy has been used, for instance, to model the generation and communication of fresh values (nonces) in mobile reactive systems as security protocols; and (2) the underlying constraint system in CCP has been enhanced with local stores for the specification of distributed spaces. Then, agents are allowed to share some information with others but keep some facts for themselves. Recently, we have shown that local stores can be neatly represented in CCP by considering a constraint system where constraints are built from a fragment of linear logic with subexponentials (SELL$^{\Cap}$). In this paper, we explore the use of existential ($\Cup$) and universal ($\Cap$) quantification over subexponentials in SELL$^{\Cap}$ in order to endow CCP with the ability to communicate location (space) names. The resulting CCP language we obtain is a model of distributed computation where it is possible to dynamically establish new shared spaces for communication. We thus extend the sort of mobility achieved in (1) –for variables – to dynamically change the shared spaces among agents – (2) above. Finally, we argue that the new CCP language can be used in the specification of service oriented computing systems.

Keywords: Concurrent Constraint Programming, Linear Logic, Subexponentials, Mobility, Distributed Spaces.

[1] Email: carlos.olarte@gmail.com
[2] Email: vivek.nigam@gmail.com
[3] Email: elaine@mat.ufmg.br

1 Introduction

The specification of modern concurrent systems often requires to reason using different sorts of modalities, such as time, space, or even the epistemic state of agents. Logic and proof theory have often inspired the design of many of these formalisms. For example, Saraswat and Rinard proposed in [25] Concurrent Constraint Programming (CCP) [23,26] which is a model for concurrency that combines the traditional operational view of process calculi [14] with a declarative view based on logic. Agents in CCP *interact* with each other by *telling* and *asking* information represented as *constraints* (formulas in logic) to a global store. Later, Fages *et al.* in [6] proposed Linear Concurrent Constraint Programming (`lcc`), inspired on linear logic [8] and linear logic programming [11], allowing the use of linear constraints, that is, constraints that once used by an agent are removed from the global store. On the other hand, Saraswat *et al.* proposed Timed CCP (`tcc`) [24], which is an extension of CCP with time modalities. More recently, Knight *et al.* [12] proposed another CCP-based language with spatial (`sccp`) and epistemic (`eccp`) modalities (see in [20] a survey of the state of the art in CCP).

On the other hand, we have showed in [18] that spatial, temporal and epistemic modalities can be *uniformly* specified in a single logical framework called SELL$^{\Cap}$. The proof system SELL$^{\Cap}$ is an extension of *SELL* (intuitionistic linear logic with subexponentials) with a pair of quantifiers over subexponentials, namely, $\Cap$ (universal) and $\Cup$ (existential). These quantifiers allowed the encoding of an existing number of CCP languages, and that does not seem to be possible with other logical frameworks, such as [28,3]. In fact, the view of subexponentials as "locations" greatly enhance the power of the logical framework that is attached to them, in this case linear logic. Moreover, the ability to *reason about locations* seems to be the key for capturing different behaviors in CCP. In particular, the above mentioned quantifiers enable the use of an *arbitrary number of subexponentials*, that plays an important role in the encodings described in [18]. For instance, they are used in the modeling of the unbounded nesting of modalities, which is a common feature of systems involving spatial and epistemic information.

Another important feature of subexponentials is that they can be organized into a pre-order, which specifies the provability relation among them. With the use of quantifiers together with an adequate pre-order over subexponentials, it is possible to specify *declaratively* the rules in which agents can manipulate information: the boundaries are naturally implied by the pre-order of subexponentials.

The fact is that the results in [18] opened a number of possibilities for the specification and verification of systems that mention modalities. For instance, while in [12] one assumes a finite number of agents, it seems possible to extend these systems in order to handle an infinite number of them via quantification on locations. Moreover, linearity of constraints can be straightforwardly included to these systems to represent, for instance, agents that can *update/change* the content of the distributed spaces. Also, by changing the underlying subexponential structure, different modalities can be put in the hands of the modelers and programmers. And most importantly, now it is possible to use all the linear logic meta-theory for reasoning about such systems.

All in all, we can summarize the work done in [18] by

$$\{\mathtt{l},\mathtt{e},\mathtt{s},\mathtt{t},\mathtt{ck}\} - CCP \implies \text{SELL}^{\Cap}$$

where $\{\mathtt{l}, \mathtt{e}, \mathtt{s}, \mathtt{t}, \mathtt{ck}\}$-CCP stands for linear, epistemic, spatial, timed, and common knowledge modalities in CCP. The present work moves in the opposite direction:

$$\text{SELL}^{\Cap} \implies ? - CCP$$

That is, we can ask ourselves what kind of interesting features can be specified in CCP if we enrich the underlying theory of this model with the subexponential discipline.

There are at least two ways of proceeding in this direction. One aspect that could be explored is that of proposing richer subexponential signatures in SELL$^{\Cap}$, hence having different computational behaviors. For example, it seems that if the initial signature is the $[0,1]$ interval, then the correspondent calculus has a probabilistic flavor. Hence one could think of more elaborated topological spaces as signatures, like Hilbert spaces for example, achieving in the other side interesting concurrent systems.

In the present work, though, we will explore another possible aspect driven by SELL$^{\Cap}$: *reconfigurability* of the communication structure, aka *mobility*. In fact, instead of looking to the initial subexponential signature, we will focus our attention on the *quantification over subexponentials*. That is, we will show how the new subexponentials, created by existential quantification ($\Cup$), can be used as private locations that can be communicated and shared to other agents via the universal quantification ($\Cap$). This is reminiscent of the name-passing discipline (for first-order variables) proposed by Olarte and Valencia in Universal Timed CCP (utcc) [21], where local ($\exists$) processes are used to create nonces (fresh values) that can be accessed through universally quantified asks ($\forall$).

The rest of the paper is organized as follows. In Section 2 we recall the framework of SELL$^{\Cap}$ first proposed in [18]. Then, we extend the linear constraint system in [6] to consider formulas in a fragment of SELL$^{\Cap}$. We shall show that processes manipulating such constraints are able to represent interesting behavior in concurrent and distributed systems. We rely on the results in [18] to show that the operational semantics of the CCP language here proposed has a strong adequacy (at the level of derivations) with proofs in SELL$^{\Cap}$. Section 4 concludes the paper.

2 Linear Logic and Subexponential Quantifiers

We shall now review some basic proof theory of Girard's intuitionistic linear logic (ILL) [8] with subexponentials [4]. ILL's connectives are the conjunctions $\otimes$ and $\&$; the disjunction $\oplus$; the implication $\multimap$; the (first-order) quantifiers $\forall$ and $\exists$; the exponentials $!, ?$;[4] and the units $1, \top$ and 0.

Due to the exponential !, we can distinguish in linear logic two kinds of formulas in the left context: the linear ones whose main connective is not a ! and the unbounded ones whose main connective is a !. It turns out that the exponentials are not canonical with respect to the logical equivalence relation. In fact, if, for any reason, we decide to define a

[4] Although ? is not part of ILL, we can add a linear version of it, not allowing it to be contracted or weakened.

blue and red conjunctions ($\wedge^b$ and $\wedge^r$ respectively) with the standard rules:

$$\frac{\Gamma, A, B \longrightarrow C}{\Gamma, A \wedge^b B \longrightarrow C}\ \wedge^b L \qquad \frac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow B}{\Gamma \longrightarrow A \wedge^b B}\ \wedge^b R$$

$$\frac{\Gamma, A, B \longrightarrow C}{\Gamma, A \wedge^r B \longrightarrow C}\ \wedge^r L \qquad \frac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow B}{\Gamma \longrightarrow A \wedge^r B}\ \wedge^r R$$

then it is easy to show that, for any formulas A and B, $A \wedge^b B \equiv A \wedge^r B$. This means that all the symbols for intuitionistic conjunction belong to the same equivalence class. Hence, we can choose to use as the conjunction's *canonical* form any particular color, and provability is not affected by this choice.

However, the same behavior does not hold with the linear logic exponentials. In fact, suppose we have red $!^r$ and blue $!^b$ exponentials with the standard linear logic rules:

$$\frac{!^r \Gamma \longrightarrow F}{!^r \Gamma \longrightarrow !^r F}\ !^r R \qquad \frac{\Gamma, F \longrightarrow C}{\Gamma, !^r F \longrightarrow C}\ !^r L \qquad \frac{!^b \Gamma \longrightarrow F}{!^b \Gamma \longrightarrow !^b F}\ !^b R \qquad \frac{\Gamma, F \longrightarrow C}{\Gamma, !^b F \longrightarrow C}\ !^b L$$

In this case, we cannot show that $!^r F \equiv !^b F$. This thus opens the possibility of defining classes of exponentials.

Formally, intuitionistic linear logic with subexponentials (*SELL*) shares with ILL all connectives and their inference rules, except the exponentials: instead of having a single pair of exponentials ! and ?, *SELL* may contain as many *labelled* exponentials, $!^l, ?^l$, as needed. These are called *subexponentials* [4]. The *subexponential signature* $\Sigma = \langle I, \preceq, U \rangle$ is built from a set of labels I, $U \subseteq I$ is a set specifying which subexponentials allow weakening and contraction on the left side of a sequent, and $\preceq$ is the pre-order among the elements of I. We assume that U is closed wrt $\preceq$, *i.e.*, if $a \in U$ and $a \preceq b$, then $b \in U$.

The exponentials introduction rules are as follows. For each $a \in I$, we add the introduction rules corresponding to dereliction and promotion, where we state explicitly the first-order signature $\mathcal{L}$ of the terms of the language:

$$\frac{\mathcal{L}; \Gamma, F \longrightarrow G}{\mathcal{L}; \Gamma, !^a F \longrightarrow G}\ !^a{}_L \quad \text{and} \quad \frac{\mathcal{L}; !^{x_1} F_1, \ldots !^{x_n} F_n \longrightarrow G}{\mathcal{L}; !^{x_1} F_1, \ldots !^{x_n} F_n \longrightarrow !^a G}\ !^a{}_R$$

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

Observe that this means that provability is preserved *downwards*: if a formula $!^a P$ is provable from a set of hypothesis, so it is $!^b P$, for $b \preceq a$.

Furthermore, for all $a \in U$, we add the structural rules:

$$\frac{\mathcal{L}; \Gamma, !^a F, !^a F \longrightarrow G}{\mathcal{L}; \Gamma, !^a F \longrightarrow G}\ C \quad \text{and} \quad \frac{\mathcal{L}; \Gamma \longrightarrow G}{\mathcal{L}; \Gamma, !^a F \longrightarrow G}\ W$$

That is, we are also free to specify which indices are *unbounded* (those appearing in the set U), and which indices are *linear* or *bounded*. See the companion technical report of [18] at the authors' web page for a focused intuitionistic version of the SELL$^{\Cap}$ system.

It is known that subexponentials greatly increase the expressiveness of the system when compared to linear logic. For instance, subexponentials can be used to represent contexts of proof systems [19], to mark the epistemic state of agents [16], or to specify locations in sequential computations [17].

The key difference to standard presentations of linear logic is that while linear logic has only seven logically distinct prefixes of bangs and question-marks, *SELL* allows for an unbounded number of such prefixes, *e.g.*, $!^i$, or $!^i?^j$. As showed in [18], by using different prefixes, we are able to interpret subexponentials in more creative ways, such as temporal units or spatial and epistemic modalities in distributed systems. For instance, $!^i P$ specifies that the process P is located at space i. Moreover, since $!^i P \longrightarrow P$, the information of P can be propagated outside the space i. On the other side, $!^i?^i P$ specifies that P is located at i but its information is confined to the space i.

The interpretation of the aforementioned modalities in *SELL* relies on the ability to quantify on subexponentials. For that, we introduced in [18] the system SELL$^{\Cap}$ containing two novel connectives: universal ($\Cap$) and existential ($\Cup$) quantifiers over *subexponentials*.

2.1 Subexponential Quantifiers

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

For subexponential quantification, we will be interested in determining whether a subexponential b belongs to the ideal $\downarrow a$ of a given subexponential a. This is formally achieved by adding a typing information to subexponentials. Given the signature $\Sigma = \langle I, \preceq, F, U\rangle$, the judgment $s : a$ is true whenever $s \preceq a$. Thus we obtain the set $\mathcal{A}_\Sigma = \{s : a \mid s, a \in I, s \preceq a\}$ of typed *subexponential constants*.

As with the universal quantifier $\forall$, which introduces *eigenvariables* to the signature, the universal quantification for subexponentials $\Cap$ introduces *subexponential variables* $l : a$, where a is a subexponential constant, *i.e.*, $a \in I$. Thus, SELL$^{\Cap}$ sequents have the form $\mathcal{A}; \mathcal{L}; \Gamma \longrightarrow G$, where $\mathcal{A} = \mathcal{A}_\Sigma \cup \{l_1 : a_1, \ldots, l_n : a_n\}$, and $\{l_1, \ldots, l_n\}$ is a disjoint set of subexponential variables and $\{a_1, \ldots, a_n\} \subseteq I$ are subexponential constants. Formally, only these subexponential constants and variables may appear free as an index of subexponential bangs and question marks.

The introduction rules for the subexponential quantifiers look similar to those introducing the first-order quantifiers, but instead of manipulating the context $\mathcal{L}$, they manipulate the context $\mathcal{A}$, where l_e is fresh, *i.e.*, not appearing in $\mathcal{A}$ nor $\mathcal{L}$ and $l : a \in \mathcal{A}$:

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

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

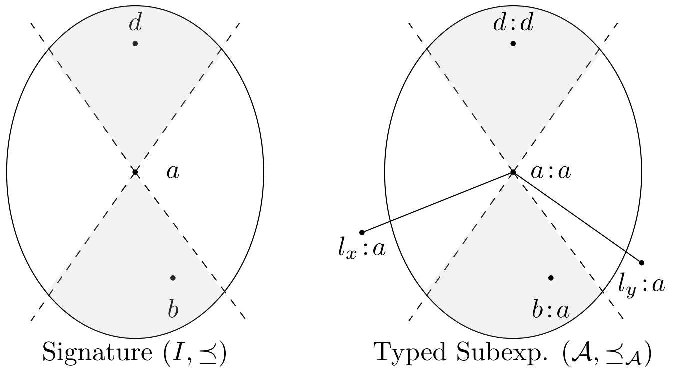

Fig. 1. $a, b, d \in I$ and $\mathfrak{f}(b : a)$ represents any subexponential constant in the ideal of a and $\mathfrak{f}(b : a) \preceq_{\mathcal{A}} \mathfrak{f}(a : a)$. l_x, l_y are subexponential variables of type a. Note that $\mathfrak{f}(l_x : a) \preceq_{\mathcal{A}} \mathfrak{f}(a : a) \preceq_{\mathcal{A}} \mathfrak{f}(d : d)$. Note also that $\mathfrak{f}(l_x : a) \not\preceq_{\mathcal{A}} \mathfrak{f}(l_y : a)$, i.e, variables, even of the same type, are unrelated w.r.t $\preceq_{\mathcal{A}}$.

Intuitively, subexponential variables play a similar role as eigenvariables. The generic variable $l_i : a_i$ represents *any subexponential* that is in the ideal of the subexponential constant a_i. This is formalized by constructing a pre-order, called *sequent pre-order*, written $\preceq_{\mathcal{A}}$. This pre-order is formally used in the side condition of the promotion rule and is defined on subexponentials obtained from applying a family $\mathfrak{f}_i \in F$ to an element of I. Formally, it is the *transitive* and *reflexive closure* of the sets below.

$$\{\mathfrak{f}(s_i : a) \preceq_{\mathcal{A}} \mathfrak{f}(s_j : b) \mid \mathfrak{f} \in F, s_i, s_j \in I \text{ and } s_i \preceq s_j\} \ \cup$$
$$\{\mathfrak{f}(l : a) \preceq_{\mathcal{A}} \mathfrak{f}(s : b) \mid \mathfrak{f} \in F, l \notin I, s \in I \text{ and } a \preceq s\}$$

The first component of this set specifies that families preserve the pre-order $\preceq$ in Σ only involving subexponential constants; thus $\preceq_{\mathcal{A}}$ is a conservative extension of $\preceq$. The second component is the interesting one, which relates subexponential obtained from variables and subexponentials obtained from constants: $l : a$ means that l belongs to the ideal of a and if $a \preceq s$, then $\mathfrak{f}(l : a) \preceq_{\mathcal{A}} \mathfrak{f}(s : s')$. Notice that $\mathfrak{f}(l_i : a_i)$ and $\mathfrak{f}(l_j : a_j)$ are unrelated for two different subexponentials variables l_i and l_j (see Figure 1).

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

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

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

with the side condition that for all $1 \leq i \leq n+1$, $\mathfrak{f}(l : a) \preceq_{\mathcal{A}} \mathfrak{f}(l_i : a_i)$, where $\preceq_{\mathcal{A}}$ is the sequent pre-order constructed from the signatures Σ and $\mathcal{A}$, as described above.

Notice that bangs and question marks use families, while quantifiers use only constants and variables (i.e., typed subexponentials). This interplay allows us to bind formulas with different families as in the formula $\Cap l : a.[!^{\mathfrak{f}(l:a)}P \otimes !^{\mathfrak{g}(l:a)}P']$.

As pointed out in [4], for cut-elimination, one needs to be careful with the structural properties of subexponentials. For subexponential variables, we define $\mathfrak{f}(l_i : a)$ to be always bounded, while for subexponential constants, it is similar as before: if $\mathfrak{f}(s : a) \in U$, then structural rules can be applied.

In [18] we proved that cut-elimination is admissible for the SELL$^{\Cap}$ system.

Theorem 2.1 *For any signature Σ, the proof system SELL$^{\Cap}$ admits cut-elimination.*

In the remainder of the paper, we shall simply write $!^{\mathfrak{f}(l)}$ instead of $!^{\mathfrak{f}(l\,:\,a)}$ when the type ": a" can be inferred from the context as in $\Cap l : a.(!^{\mathfrak{f}(l:a)}F)$. Similarly for "?." We shall also write $!^s$ and $?^s$ when the family and the type are not important or can be inferred from the context. Moreover, given a sequence of subexponentials of the form $\overline{s} = s_1.\cdots.s_n$, we shall write $!^{\overline{s}}$ to mean $!^{s_1}\cdots!^{s_n}$. Similarly for $?^{\overline{s}}$.

3 SELL$^{\Cap}$ as Constraint System

Concurrent Constraint Programming (CCP) [25,23,26] is a model for concurrency that combines the traditional operational view of process calculi with a *declarative* view based on logic (see a survey in [20]). This allows CCP to benefit from the large set of reasoning techniques of both process calculi and logic. Processes in CCP *interact* with each other by *telling* and *asking* constraints (pieces of information) in a common store of partial information. The type of constraints processes may act on is not fixed but parametric in a constraint system. Such systems can be formalized as a Scott information system as in [26], or they be can specified as formulas in a suitable fragment of logic *e.g.*, as in [27,15]. Here we build on the ideas of specifying constraint systems as formulas in Girard's linear logic as in linear CCP (`lcc`) [6]. More precisely, we allow constraints to be formulas in a fragment of SELL$^{\Cap}$. As we shall show later, this gives rise to a more powerful CCP language that is able to capture, declaratively, distributed spaces. Furthermore, the new language offers primitives for a name passing discipline (i.e., mobility) upon both, local names (variables) and locations (subexponential indexes). The latter allows agents to define shared spaces of communication.

Let us start by defining the fragment of SELL$^{\Cap}$ that will serve as the basis to the constraint system.

Definition 3.1 [SELL$^{\Cap}$-Constraint System] A subexponential constraint system (`scs` for short) is a tuple $(\Sigma, C, \vdash_\Delta)$ where Σ is a subexponential signature where all subexponentials are unrelated except for a distinguished subexponential l_∞, which is the top element of the poset Σ. C is a set of formulas (constraints) built from a first-order signature and the grammar

$$F := 1 \mid A \mid F \otimes F \mid \exists \overline{x}.F \mid !^s ?^s F$$

where A is an atomic formula. We shall use c, c', d, d', etc, to denote elements of C. Moreover, let Δ be a set of non-logical axioms of the form $\forall \overline{x}[c \multimap c']$ 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 $C[\![\Delta]\!], d \longrightarrow d'$ is probable in SELL$^{\Cap}$ ($C[\![\Delta]\!]$ is later stated in Definition 3.8). We shall omit the "Δ" in $\vdash_\Delta$ when it is unimportant or it can be inferred from the context.

Let us give some intuitions on the above fragment of SELL$^{\Cap}$. The connective 1 corresponds to the empty store, i.e., the initial state of computation. The connective $\otimes$ allows processes to add more information to the store. The existential quantifier hides variables from constraints. The formula $!^{\mathfrak{f}_j(s_j:a)}?^{\mathfrak{f}_j(s_j:a)}c$ specifies that the constraint c is in the space-location s_j of the agent $\mathfrak{f}_j$ and this information is confined to that space (see Proposition 3.2 below). Finally, $C[\![\Delta]\!]$ intuitively means that the axioms defined in Δ are available in all the spaces of the system.

Notation 3.1 *We shall use both $[c]_s$ and $\bigtriangledown_s c$ to denote the constraint $!^s?^s c$ for an esthetic reason: the first notation will be used when the constraints are inside processes, while the second when they are in the store.*

An interesting behavior of the formula $[c]_s$ is that it defines spaces where even inconsistent information can be confined.

Proposition 3.2 (False confinement) *Let $(\Sigma, C_s, \vdash_{\Delta_s})$ be a* `scs` *and assume that the following sequent is provable $c \otimes d \longrightarrow_{\Delta} 0$. Then,*

(i) $[0]_s \longrightarrow [c]_s$ *(any c can be deduced in the space s if its local store is inconsistent);*

(ii) $[0]_s \not\longrightarrow [0]_{s'}$ *if $s \neq s'$ (inconsistency is confined);*

(iii) $C[\![\Delta]\!], [c]_s, [d]_s \longrightarrow [0]_s$ *(if space s contains both c and d, then it becomes inconsistent);*

(iv) $C[\![\Delta]\!], [c]_s, [d]_{s'} \not\longrightarrow [0]_s$ *if $s \neq s'$ (false is not deduced if c and d are in different spaces);*

(v) $[c]_s \not\longrightarrow c$ *(local information is not global).*

3.1 The language of Processes

In this section we propose `dccp`, a CCP-based language able to manipulate constraints built from a subexponential constraint system. The main design criteria for this language 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 family names in the subexponential signature. For that, notice that subexponentials of two different families are unrelated and then, 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, we shall identify such locations as different subexponential indices in the signature as we did in [18]. Unlike `sccp` [12], we shall allow unbounded and linear locations to specify spaces where information can be updated;

(iii) agents can exchange local names, i.e., it is possible to reconfigure the communication structure of the system. This is achieved by the interplay of local processes and universally quantified asks as in Universal Timed CCP (`utcc`) [21] ; and

(iv) agents can create new sub-spaces (local stores) and communicate them to other agents, thus defining new shared spaces for communication.

Similar to most processes calculi, the language of processes of `dccp` 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` [6,10] and `utcc` [21], 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 on some others are linear. Finally, following the developments of spatial CCP (sccp) [12], we allow processes to be confined to a given space (see $[P]_s$ below). However, unlike sccp, in dccp 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 SELL$^{\Cap}$.

Definition 3.3 [Syntax of dccp] Processes in dccp are built from constraints in the underlying subexponential constraint system as follows:

$$P, Q := \textbf{tell}(c) \mid (\textbf{local}\, \overline{x}; \overline{l})\, Q \mid (\textbf{abs}\, \overline{x}; \overline{l}; c)\, Q \mid P \parallel Q \mid [P]_{\mathfrak{f}(l)} \mid p(\overline{x})$$

where variables in $\overline{x}$ and subexponential indexes in $\overline{l}$ are pairwise distinct. We assume that for each agent in the system there is a unique family name $\mathfrak{f}$ and the behavior of such agent is defined as $Agent\ \mathfrak{f} \stackrel{\mathtt{def}}{=} P$. Moreover, 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 $\overline{x}$.

Let us give some intuitions 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 $(\textbf{local}\, \overline{x}; \overline{l})\, Q$ creates a new set of variables $\overline{x}$ and a set of new subexponential indexes and declares them to be private to Q. When any of those sets is empty, as in $(\textbf{local}\, \overline{x}; \emptyset)\, Q$, we simply write $(\textbf{local}\, \overline{x})\, Q$ when no confusion arises. Furthermore, instead of $(\textbf{local}\, \{x\}; \{l\})\, Q$ we write $(\textbf{local}\, x; l)\, Q$.

The process $(\textbf{abs}\ \overline{x}; \overline{l}; c)\, Q$ evolves into $Q[\overline{y}, \overline{s}/\overline{x}, \overline{l}]$ if the (distributed) store entails $c[\overline{y}, \overline{s}/\overline{x}, \overline{l}]$. When this happens, the constraint c is consumed. When either $\overline{x}$ or $\overline{l}$ 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 $(\textbf{abs}\, \emptyset; \emptyset; c)\, Q$. The **abs** constructor can be then used as a synchronization mechanism based on entailment of constraints.

The parallel composition of P and Q is denoted as $P \parallel Q$. The processes $[P]_l$ executes and confines the process P in the space l. Finally, given a process definition of the form $p(\overline{x}) \stackrel{\Delta}{=} P$, the agent $p(\overline{y})$ executes the process $P[\overline{y}/\overline{x}]$.

Before we go any further, let us note that some processes built from Definition 3.3 may not adhere to the design criteria (i) of the language. For instance, assume the following agent definition

$$Agent\ \mathfrak{f} \stackrel{\mathtt{def}}{=} (\textbf{abs}\ l; [c]_{\mathfrak{g}(l)})\, P$$

In this case, $\mathfrak{f}$ will query *all the spaces* in the store of $\mathfrak{g}$, and it can possibly consume information from it. Hence, agent $\mathfrak{f}$ was able to directly read the store of another agent. Now consider the definition

$$Agent\ \mathfrak{f} \stackrel{\mathtt{def}}{=} [P]_{\mathfrak{g}(l)}$$

Here, $\mathfrak{f}$ is able to *execute* the process P in the space of computation of $\mathfrak{g}$. In order to avoid this undesired behaviors, we need to impose syntactic restrictions on the processes and constraints agents can tell and ask:

Definition 3.4 [Well-formed agents] Let $\mathfrak{f}$ and $\mathfrak{g}$ be two different agent names. We say that the agent definition $Agent\ \mathfrak{f} \stackrel{\mathtt{def}}{=} P$ is well-formed if:

$$\frac{\text{all formula in } c \text{ is indexed by the family } \mathfrak{f}_i}{\cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, \textbf{tell}(c); d_i \right\rangle \cdots \longrightarrow \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i; d_i \otimes c \right\rangle \cdots} \ \mathrm{R}_{\mathrm{TELL}}$$

$$\frac{}{\cdots\cdots \mathfrak{f}_j : \left\langle \overline{x}_j; \bar{l}_j; \Gamma_j; d_j \right\rangle \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, \textbf{tell}([A]_{\mathfrak{f}_j(l)}); d_i \right\rangle \cdots \longrightarrow \cdots \mathfrak{f}_j : \left\langle \overline{x}_j; \bar{l}_j; \Gamma_j; d_j \otimes \nabla_{\mathfrak{f}_j(l)} A \right\rangle \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i; d_i \right\rangle \cdots} \ \mathrm{R}_{\mathrm{TDIS}}$$

$$\frac{\forall j \in 1..n, \overline{x}_j \cap \overline{y} = fv(d_j) \cap \overline{y} = fv(\Gamma_j) \cap \overline{y} = \emptyset}{\cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, (\textbf{local}\, \overline{y}; \bar{l})\, Q; d_i \right\rangle \cdots \longrightarrow \cdots \mathfrak{f}_i : \left\langle \overline{x}_i \cup \overline{y}; \bar{l}_i; \Gamma_i, (\textbf{local}\, \bar{l})\, Q; d_i \right\rangle \cdots} \ \mathrm{R}_{\mathrm{L}}$$

$$\frac{\bar{l}_i \cap \bar{l} = \emptyset}{\begin{array}{l} \mathfrak{f}_1 : \left\langle \overline{x}_1; \bar{l}_1; \Gamma_1; d_1 \right\rangle, \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, (\textbf{local}\, \overline{y}; \bar{l})\, Q; d_i \right\rangle, \cdots \mathfrak{f}_n : \left\langle \overline{x}_n; \bar{l}_n; \Gamma_n; d_n \right\rangle \qquad \longrightarrow \\ \mathfrak{f}_1 : \left\langle \overline{x}_1; \bar{l}_1 \cup \bar{l}; \Gamma_1; d_1 \right\rangle, \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i \cup \bar{l}; \Gamma_i, (\textbf{local}\, \overline{y})\, Q; d_i \right\rangle, \cdots \mathfrak{f}_n : \left\langle \overline{x}_n; \bar{l}_n \cup \bar{l}; \Gamma_n; d_n \right\rangle \end{array}} \ \mathrm{R}_{\mathrm{LL}}$$

$$\frac{d_i \vdash c[\overline{y}'/\overline{y}][\bar{l}'/\bar{l}] \otimes e}{\cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, (\textbf{abs}\, \overline{y}; \bar{l}; c)\, Q; d_i \right\rangle \cdots \longrightarrow \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i Q[\overline{y}'/\overline{y}][\bar{l}'/\bar{l}]; e \right\rangle \cdots} \ \mathrm{R}_{\mathrm{A}}$$

$$\frac{\cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, P; d_i \right\rangle \longrightarrow \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, P'; d_i \right\rangle \cdots}{\cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, Q; d_i \right\rangle \longrightarrow \cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, Q'; d_i \right\rangle \cdots} \ \mathrm{R}_{\mathrm{STR}}, \quad \text{if P} \equiv \text{Q and P}' \equiv \text{Q}'$$

$$\frac{\left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, P; d_i \right\rangle \longrightarrow \left\langle \overline{x}'_i; \bar{l}'_i; \Gamma'_i, P'; d'_i \right\rangle}{\cdots \mathfrak{f}_i : \left\langle \overline{x}_i; \bar{l}_i; \Gamma_i, [P]_{\mathfrak{f}_i(l)}; \nabla_{\mathfrak{f}_i(l)} d_i \otimes \nabla_{\mathfrak{f}_i(x)} d_j \right\rangle \cdots \longrightarrow \cdots \mathfrak{f}_i : \left\langle \overline{x}'_i; \bar{l}'_i; \Gamma'_i, [P']_l; \nabla_{\mathfrak{f}_i(l)} d'_i \otimes \nabla_{\mathfrak{f}_i(x)} d_j \right\rangle \cdots} \ \mathrm{R}_{\mathrm{S}}$$

$$\frac{p(\overline{x}) \overset{\Delta}{=} P}{\cdots \mathfrak{f}_i : \langle \overline{x}_i; \bar{l}_i; \Gamma, p(\overline{y}); d_i \rangle \cdots \longrightarrow \cdots \mathfrak{f}_i : \langle \overline{x}_i; \bar{l}_i; \Gamma P[\overline{y}/\overline{x}]; d_i \rangle \cdots} \ \mathrm{R}_{\mathrm{C}}$$

Fig. 2. Structural Operational Semantics. $fv(\cdot)$ denotes the set of free variables. In R_{A}, the constraint e is the most general constraint to avoid weakening the store (see [9]). .

(i) All process subterm of the form $[Q]_s$ in P is of the shape $[Q]_{\mathfrak{f}(l)}$.

(ii) Constraints of the form $[c]_{\mathfrak{g}(l)}$ only appear in the scope of a tell agent.

Restriction (i) prevents agents to execute processes in the space of other agents. Restriction (2) disallows agents able to read from the store of another agent. Moreover, as we shall see, processes of the form $\textbf{tell}([c]_{\mathfrak{g}(l)})$ in the agent $\mathfrak{f}$ will be interpreted as an asynchronous communication from $\mathfrak{f}$ to $\mathfrak{g}$.

Notation 3.2 *Assume the agent definition Agent $\mathfrak{f} \overset{\mathtt{def}}{=} P$. As a consequence of the previous restrictions, we shall omit the "$\mathfrak{f}$" in a subterm $[Q]_{\mathfrak{f}(l)}$ in P. Similarly, we shall omit the "$\mathfrak{f}$" in the ask agents defined in P. Moreover, we shall understand* $\textbf{tell}([c]_l)$ *as* $\textbf{tell}([c]_{\mathfrak{f}(l)})$.

3.2 Operational Semantics

The operational semantics of dccp is given by the transition relation $\gamma \longrightarrow \gamma'$ satisfying the rules on Figure 2. A *configuration* γ is a set of tuples of the form

$$\mathfrak{f}_1 : \left\langle \overline{x}_1; \bar{l}_1; \Gamma_1; c_1 \right\rangle, ..., \mathfrak{f}_n : \left\langle \overline{x}_n; \bar{l}_n; \Gamma_n; c_n \right\rangle$$

where c_i is a constraint specifying the store of agent $\mathfrak{f}_i$, Γ is a multiset of processes (the behavior of $\mathfrak{f}_i$), and $\overline{x}_i$, $\bar{l}_i$ are the set of hidden (local) variables and spaces of c_i and Γ_i. 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) renaming of bound variables; (2) $P \parallel Q \cong Q \parallel P$: and (3) $P \parallel (Q \parallel R) \cong (P \parallel Q) \parallel R$; (4) $(\textbf{local}\, \emptyset; \emptyset)\, Q \cong Q$; (5) $\textbf{tell}(c \otimes d) \cong \textbf{tell}(c) \parallel \textbf{tell}(d)$.

Let $\longrightarrow^*$ be the reflexive and transitive closure of $\longrightarrow$. If

$$\mathfrak{f}_1 : \left\langle \overline{x}_1; \overline{l}_1; \Gamma_1; c_1 \right\rangle, ..., \mathfrak{f}_n : \left\langle \overline{x}_n; \overline{l}_n; \Gamma_n; c_n \right\rangle \longrightarrow^* \mathfrak{f}_1 : \left\langle \overline{x}'_1; \overline{l}'_1; \Gamma'_1; c'_1 \right\rangle, ..., \mathfrak{f}_n : \left\langle \overline{x}'_n; \overline{l}'_n; \Gamma'_n; c'_n \right\rangle$$

and the sequent $!^{l_\infty} \Cap l : \infty.\mathcal{C}[\![\Delta]\!]_l, \exists \overline{x}'_i. \Cup \overline{l}'_i.c'_i \longrightarrow d$ is provable, we write $\mathfrak{f}_i : \left\langle \overline{x}_i; \overline{l}_i; \Gamma_i; c_i \right\rangle \Downarrow_d$. If $\overline{x}_i = \overline{l}_i = \emptyset$ and $c_i = 1$ we simply write $\mathfrak{f}_i \Downarrow_d$. Intuitively, for an agent definition of the form $Agent\ \mathfrak{f}_i \stackrel{\text{def}}{=} P$, the set $\{d \in \mathcal{C} \mid \mathfrak{f}_i \Downarrow_d\}$ captures the outputs of P under input 1. The formula $\mathcal{C}[\![\Delta]\!]$ will be clarified in Definition 3.8.

Given a set of agent declarations of the form $Agent\ \mathfrak{f}_1 \stackrel{\text{def}}{=} P_1, \cdots, Agent\ \mathfrak{f}_n \stackrel{\text{def}}{=} P_n$ we shall consider the configuration $\mathfrak{f}_1 : \langle \emptyset; \emptyset; P_1; 1 \rangle, ..., \mathfrak{f}_n : \langle \emptyset; \emptyset; P_n; 1 \rangle$ as the initial state of the system.

Now we give some intuitions about the operational rules:

- $\mathrm{R}_{\mathrm{TELL}}$: If all formula in c is indexed by the family $\mathfrak{f}_i$, the process $\textbf{tell}(c)$ in the agent $\mathfrak{f}_i$ is able to add c to its local store. Notice that a process of the form $\textbf{tell}([c]_{\mathfrak{f}_j(l)} \otimes [d]_{\mathfrak{f}_i(l')})$, via rule (5) of the structural congruence, can be decomposed into $\textbf{tell}([c]_{\mathfrak{f}_j(l)}) \parallel \textbf{tell}([d]_{\mathfrak{f}_i(l')})$ to fulfill the side condition in this rule.
- If an agent $\mathfrak{f}_i$ is willing to communicate the atomic formula A to another agent $\mathfrak{f}_j$, it can asynchronously post the constraint $\bigtriangledown_{\mathfrak{f}_j(l)} A$. Rule $\mathrm{R}_{\mathrm{TDIS}}$ says that constraints A is "communicated" and added to the store of the agent $\mathfrak{f}_j$.
- A process $(\textbf{local}\, \overline{y}; \overline{l})\, Q$ adds the local variables $\overline{y}$ (resp. the fresh subexponential variables $\overline{l}$) to the sets $\overline{x}_i$ (resp. $\overline{l}_i$) as it is shown in Rule R_{L} (resp. R_{LL}). We recall that the left introduction rule of $\Cup$ creates the new location and makes it available to all the families in the system (see [18]). Then, Rule R_{LL} adds $\overline{l}$ to all the configurations of the agents in the system (see Example 3.7).
- If the local store d_i of the agent $\mathfrak{f}_i$ is able to entail $c[\overline{y}'/\overline{y}][\overline{l}'\overline{l}]$, then the agent $(\textbf{abs}\, \overline{y}; \overline{l}; c)\, Q$ evolves into $Q[\overline{y}'/\overline{y}][\overline{l}'/\overline{l}]$ and consumes the constraint c. Note that the constraint e in the entailment $d \vdash c[\overline{y}'/\overline{y}][\overline{l}'/\overline{l}] \otimes e$ is not necessarily unique. Take for instance the entailment $!c \vdash c \otimes 1$ and $!c \vdash c \otimes (!c \otimes 1)$. In the first case, we have an unwanted weakening of the store. To avoid this problem, we require e to be the most general choice as in [9]. We recall that the agent $\mathfrak{f}$ is able to query only its own local store since constraints in the guard c must be marked only with subexponentials of the family $\mathfrak{f}$.
- Rule $\mathrm{R}_{\mathrm{STR}}$ says that congruent processes have the same reductions.
- To explain the rule R_{S}, consider the process $[\textbf{tell}(c)]_l$. What we observe from this process is that the constraint $[c]_l$ is added to the store. This means that the output of $\textbf{tell}(c)$ is confined to the space l. Now consider the process $[\textbf{ask}\ c\ \textbf{then}\ Q]_l$. In this case, to decide if Q must be executed, we need to infer whether c can be deduced from the information in location l. Hence, the premise of Rule R_{S} considers only the store $\bigtriangledown_{\mathfrak{f}_i(l)} d_i$. Moreover, the new store in that location, i.e., d'_i is again placed at location l as shown in the conclusion of the rule.
- Rule R_{C} simply unfolds the definition of the process name p.

3.3 *Programming in* dccp

In this section we show some examples of distributed and concurrent behaviors that can be modeled in dccp. We also show how the interplay of **local** and **abs** processes allows us to dynamically create share and private stores among agents. When needed, to avoid formulas in the store not preceded by any subexponential, we assume that each agent is defined as $Agent\ \mathfrak{f} \overset{\texttt{def}}{=} [P]_{\mathfrak{f}(\texttt{out})}$ where out is a subexponential marking the "outermost" space in the store of the agent.

Example 3.5 [Local stores] Let a and b be unbounded subexponentials, $P = \textbf{tell}(c)$, $Q = \textbf{ask}\ c\ \textbf{then}\ \textbf{tell}(d)$ and $R = [P]_a \parallel [Q]_b$. Let the agent $\mathfrak{f}$ be defined as $Agent\ \mathfrak{f} \overset{\texttt{def}}{=} [R]_{\texttt{out}}$. Here we observe the following:

$$\begin{aligned} & \mathfrak{f} : \langle \emptyset; \emptyset; [R]_{\texttt{out}}; 1\rangle \\ \longrightarrow^* \ & \mathfrak{f} : \left\langle \emptyset; \emptyset; [[Q]_b]_{\texttt{out}}; \textstyle\bigtriangledown_{\mathfrak{f}(\texttt{out})} \bigtriangledown_{\mathfrak{f}(a)} c\right\rangle \not\longrightarrow \end{aligned}$$

Then, Q remains blocked since the information c is only available on the space of a. Note also that the sequent $!^{\mathfrak{f}(\texttt{out})}?^{\mathfrak{f}(\texttt{out})}!^{\mathfrak{f}(a)}?^{\mathfrak{f}(a)}c \longrightarrow !^{\mathfrak{f}(\texttt{out})}?^{\mathfrak{f}(\texttt{out})}c$ is not provable, i.e., information c is confined to the inner space a in $\mathfrak{f}$.

Now let $R = [P]_a \parallel [Q]_a$. Then, we observe as a final store the constraint:

$$\bigtriangledown_{\mathfrak{f}(\texttt{out})} \bigtriangledown_{\mathfrak{f}(a)} c \otimes \bigtriangledown_{\mathfrak{f}(\texttt{out})} \bigtriangledown_{\mathfrak{f}(a)} d$$

This means that Q is able to entail the guard c in the space a to later add d to the store.

Finally, consider $R = [[P]_a]_b \parallel [Q]_a$. In this case, P will eventually add the constraint $F = \bigtriangledown_{\mathfrak{f}(\texttt{out})} \bigtriangledown_{\mathfrak{f}(b)} \bigtriangledown_{\mathfrak{f}(a)} c$. Since the sequent $F \longrightarrow \bigtriangledown_{\mathfrak{f}(\texttt{out})} \bigtriangledown_{\mathfrak{f}(a)} c$ is not probable, Q remains blocked. This intuitively means that the space that b confers to a may behave differently (*i.e.*, it contains different information) from the own space of a.

Example 3.6 [Name/Link Mobility] Name mobility is obtained by the interplay of **abs** and **local** processes when variables are considered. Assume for instance an uninterpreted predicate symbol $\texttt{com}(\cdot)$ and let $P = (\textbf{local}\ x)\,(\textbf{tell}(\texttt{com}(x)) \parallel P')$ and $Q = (\textbf{abs}\ y; \texttt{com}(y))\ Q'$ and $R = P \parallel Q$. Then the agent $Agent\ \mathfrak{f} \overset{\texttt{def}}{=} [R]_{\texttt{out}}$ behaves as follows:

$$\begin{aligned} & \mathfrak{f} : \langle \emptyset; \emptyset; [R]_{\texttt{out}}; 1\rangle \\ \longrightarrow^* \ & \mathfrak{f} : \langle x; \emptyset; [\textbf{tell}(\texttt{com}(x)) \parallel P' \parallel Q]_{\texttt{out}}; 1\rangle \\ \longrightarrow^* \ & \mathfrak{f} : \left\langle x; \emptyset; [P' \parallel Q]_{\texttt{out}}; \textstyle\bigtriangledown_{\mathfrak{f}(\texttt{out})} \texttt{com}(x)\right\rangle \\ \longrightarrow^* \ & \mathfrak{f} : \langle x; \emptyset; [P' \parallel Q'[x/y]]_{\texttt{out}}; 1\rangle \end{aligned}$$

Here we assume that out is linear and then, the constraint $\bigtriangledown_{\mathfrak{f}(\texttt{out})} \texttt{com}(x)$ is consumed when the **abs** process evolves. Note that P' and Q' share the link (variable) created by P and all the information posted on that variable may be seen by both processes.

Example 3.7 [Structured comm. patterns] Processes in dccp can exchange locations to define shared spaces of information. This mobility is also akin to the π-calculus: processes

$$
\begin{aligned}
& \mathfrak{f} : \langle \emptyset; \emptyset; \textbf{request}(\mathfrak{g},\mathfrak{f}); 1\rangle, \mathfrak{g} : \langle \emptyset; \emptyset; \textbf{accept}(\mathfrak{g},\mathfrak{f}); 1\rangle \\
\longrightarrow^{*}\ & \mathfrak{f} : \langle x; l; \textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(a)}) \parallel \textbf{ask}\ [\mathtt{com}(x)]_a\ \textbf{then}\ (\textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(l)}) \parallel P); 1\rangle, \\
& \mathfrak{g} : \langle x; l; \textbf{accept}(\mathfrak{g},\mathfrak{f}); 1\rangle \\
\longrightarrow^{*}\ & \mathfrak{f} : \langle x; l; \textbf{ask}\ [\mathtt{com}(x)]_a\ \textbf{then}\ (\textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(l)}) \parallel P); 1\rangle, \\
& \mathfrak{g} : \left\langle x; l; \textbf{accept}(\mathfrak{g},\mathfrak{f}); \nabla_{\mathfrak{g}(a)}\, \mathtt{com}(x)\right\rangle \\
\longrightarrow^{*}\ & \mathfrak{f} : \langle x; l; \textbf{ask}\ [\mathtt{com}(x)]_a\ \textbf{then}\ (\textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(l)}) \parallel P); 1\rangle, \\
& \mathfrak{g} : \langle x; l; \textbf{tell}([\mathtt{com}(x)]_{\mathfrak{f}(a)}) \parallel (\textbf{abs}\ k; [\mathtt{com}(x)]_k)\ Q; 1\rangle \\
\longrightarrow^{*}\ & \mathfrak{f} : \left\langle x; l; \textbf{ask}\ [\mathtt{com}(x)]_a\ \textbf{then}\ (\textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(l)}) \parallel P); \nabla_{\mathfrak{f}(a)}\, \mathtt{com}(x)\right\rangle, \\
& \mathfrak{g} : \langle x; l; (\textbf{abs}\ k; [\mathtt{com}(x)]_k)\ Q; 1\rangle \\
\longrightarrow^{*}\ & \mathfrak{f} : \langle x; l; P; 1\rangle, \mathfrak{g} : \left\langle x; l; (\textbf{abs}\ k; [\mathtt{com}(x)]_k)\ Q; \nabla_{\mathfrak{g}(l)}\, \mathtt{com}(x)\right\rangle \\
\longrightarrow^{*}\ & \mathfrak{f} : \langle x; l; P; 1\rangle, \mathfrak{g} : \langle x; l; Q[l/k]; 1\rangle
\end{aligned}
$$

Fig. 3. Transitions of the system in Example 3.7.

do not move but links (location variables in our case) do [14]. So, we do not change the structure of agents but we *reconfigure* the communication structure of the system.

Consider a signature with a linear subexponential a and assume two agents $\mathfrak{f}$ and $\mathfrak{g}$. Let us define the following shortcuts:

$$
\begin{aligned}
\textbf{request}(\mathfrak{g},\mathfrak{f}) &\overset{\mathtt{def}}{=} (\textbf{local}\ x, l)\,(\textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(a)}) \parallel \textbf{ask}\ [\mathtt{com}(x)]_a\ \textbf{then}\ (\textbf{tell}([\mathtt{com}(x)]_{\mathfrak{g}(l)}) \parallel P)) \\
\textbf{accept}(\mathfrak{g},\mathfrak{f}) &\overset{\mathtt{def}}{=} (\textbf{abs}\ y; [\mathtt{com}(y)]_a)\,(\textbf{tell}([\mathtt{com}(y)]_{\mathfrak{f}(a)}) \parallel (\textbf{abs}\ k; [\mathtt{com}(y)]_k)\ Q)
\end{aligned}
$$

Furthermore, let *Agent* $\mathfrak{f} \overset{\mathtt{def}}{=} \textbf{request}(\mathfrak{g},\mathfrak{f})$ and *Agent* $\mathfrak{g} \overset{\mathtt{def}}{=} \textbf{accept}(\mathfrak{g},\mathfrak{f})$. The transitions for this system are depicted in Figure 3. The process $\textbf{request}(\mathfrak{g},\mathfrak{f})$ creates a new location l and a fresh variable x. Then it sends $\mathtt{com}(x)$ to $\mathfrak{g}$ through the "public space" a. Agent $\mathfrak{g}$ consumes this information and sends back to $\mathfrak{f}$ the constraint $\mathtt{com}(x)$. Latter $\mathfrak{f}$ sends again the constraint $\mathtt{com}(x)$ but using the new established private space l. Due to the **abs** process, agent $\mathfrak{g}$ is able to read $\mathtt{com}(x)$ on l.

Some remarks are in order in this example: Note that in SELL$^{\Cap}$, the existential quantification on locations ($\Cup$) makes available the new location to all the families in the signature. Then, in the first derivation above, we observe that l is also part of the signature of agent $\mathfrak{g}$. In an distributed implementation of the calculus, however, it would be necessary to broadcast an announcement to all the agents that the new location was created. A similar problem arises with the creation of variable x. In the sequent calculus the eignevariable is added and it is "visible" to all the formulas through universal quantification. Again, in an implementation of the calculus, it would be necessary to notify the other sites (or at least the site $\mathfrak{g}$) the creation of the variable.

3.4 *Logical Characterization of Processes*

In [18] 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 [18] to consider the process $(\textbf{local}\ l)\ Q$ and $(\textbf{abs}\ l; c)\ Q$. As expected, those processes will correspond, respectively, to formulas of the shape $\Cup l.F$ and $\Cap l.F$ where F corresponds to the encoding of Q.

We shall use sequences of the form $a.b.c$ to denote the space $[[[]_c]_b]_a$ and remember that all subexponentials but ∞ are unrelated and $s \preceq \infty$ for all $s \in I$. A constraint of the form $\bigtriangledown_a \bigtriangledown_b \bigtriangledown_c d$ will be represented in the logical view of processes as $\bigtriangledown_{a.b.c} d$, thus allowing us to quantify over such prefixes (or boxes) by using a single quantifier.

We begin by encoding the stores (constraints) produced by processes as follows:

Definition 3.8 [Representation of Constraints] Let c be a constraint. We define the encoding $\mathcal{C}[\![c]\!]_l$ inductively as follows:

$$\begin{array}{lll} \mathcal{C}[\![A]\!]_{\overline{s}} = \bigtriangledown_{\overline{s}} A & \qquad & \mathcal{C}[\![c \otimes c']\!]_{\overline{s}} = \mathcal{C}[\![c]\!]_{\overline{s}} \otimes \mathcal{C}[\![c']\!]_{\overline{s}} \\ \mathcal{C}[\![\exists \overline{x}(c)]\!]_{\overline{s}} = \exists \overline{x}(\mathcal{C}[\![c]\!]_{\overline{s}}) & & \mathcal{C}[\![[c]_a]\!]_{\overline{s}} = \mathcal{C}[\![c]\!]_{\overline{s}.a} \end{array}$$

where A is an atomic formula or the unit 1.

Note that the axioms Δ of the constraint system must be available to all the spaces and agents of the system. Then, for each agent $\mathfrak{f}$, we consider the following universally quantified formula:

$$!^{\mathfrak{f}(\infty)} \Cap l : \infty.(\forall \overline{x}.(\mathcal{C}[\![c]\!]_{\mathfrak{f}(l)} \multimap \mathcal{C}[\![c]\!]_{\mathfrak{f}(l)}))$$

We shall use $\mathcal{C}[\![\Delta]\!]$ to denote the encoding of all the axioms in Δ.

Definition 3.9 [Logical view of Processes] Consider the following definition $Agent\ \mathfrak{f} \stackrel{\text{def}}{=} P$. We shall encode P as the formula $\mathcal{P}[\![P]\!]_{\mathfrak{f}(\mathtt{out})}$ where:

$$\begin{array}{lll} \mathcal{P}[\![\mathbf{tell}(c)]\!]_{\overline{s}} = \mathcal{C}[\![c]\!]_{\overline{s}} & \qquad & \mathcal{P}[\![(\mathbf{abs}\ \overline{x}; \overline{l}; c)\ Q]\!]_{\overline{s}} = \forall \overline{x}. \Cap . \overline{l}(\mathcal{C}[\![c]\!]_{\overline{s}} \multimap \mathcal{P}[\![Q]\!]_{\overline{s}}) \\ \mathcal{P}[\![(\mathbf{local}\ \overline{x}; \overline{l})\ Q]\!]_{\overline{s}} = \exists \overline{x}. \Cup \overline{l}.(\mathcal{P}[\![Q]\!]_{\overline{s}}) & & \mathcal{P}[\![P \parallel Q]\!]_{\overline{s}} = \mathcal{P}[\![P]\!]_{\overline{s}} \otimes [\![Q]\!]_{\overline{s}} \\ \mathcal{P}[\![[P]_a]\!]_{\overline{s}} = \mathcal{P}[\![P]\!]_{\overline{s}.a} & & \mathcal{P}[\![p(\overline{x})]\!]_{\overline{s}} = \mathcal{C}[\![p(\overline{x})]\!]_{\overline{s}} \end{array}$$

We assume that for all process definition $p(\overline{x}) \stackrel{\Delta}{=} P$, and for all agent $\mathfrak{f}$, the following formula is available:

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

Note that the above universal quantification allows us to unfold the definition P in all location where $p(\overline{x})$ is invoked. We shall use $\mathcal{P}[\![\Upsilon]\!]$ to denote the encoding of all the process definitions in the set Υ.

Following the proof technique in [18], we can show the following adequacy result.

Theorem 3.10 (Adequacy) *Let Agent* $\mathfrak{f}_1 \stackrel{\text{def}}{=} P_1, ..., Agent\ \mathfrak{f}_n \stackrel{\text{def}}{=} P_n$ *be a set of* dccp *agents declarations. Let* $(\mathcal{C}, \vdash_\Delta)$ *be a* scs *and let* $\mathcal{C}[\![\cdot]\!]_{\overline{l}}$ *and* $\mathcal{P}[\![\cdot]\!]_{\overline{l}}$ *be as in Definitions 3.8 and 3.9 respectively. Then for all agent* $\mathfrak{f}_i$ [5]

$$\mathfrak{f}_i \Downarrow_c \textit{ iff } \mathcal{C}[\![\Delta]\!]_{\mathtt{out}}, \mathcal{P}[\![\Upsilon]\!]_{\mathtt{out}}, \mathcal{P}[\![P_1]\!]_{\mathfrak{f}_1(\mathtt{out})}, ..., \mathcal{P}[\![P_n]\!]_{\mathfrak{f}_n(\mathtt{out})} \longrightarrow \mathcal{C}[\![c]\!]_{\mathfrak{f}_i(\mathtt{out})} \otimes \top$$

[5] The top erases the formulas corresponding to blocked processes.

4 Concluding Remarks

Related Work. Process calculi such as the π-calculus [14] allow for the specification of mobile systems, i.e., systems where agents can communicate their local names. Unlike the π-calculus (that is based on point-to-point communication), interaction in CCP is asynchronous as communication takes place thorough the shared store. In the CCP model it is possible to specify mobility in the sense of reconfiguration of the communication structure of the program. This is done by using logical variables that represent communication channels and unification to bind messages to channels [25,23]. However, since logical variables can be bound to a value only once, if two messages are sent through the same channel, then they must be equal to avoid an inconsistent store. This problem is often solved by the use of variables as streams and by relying on a communication protocol [13].

An alternative way of endowing CCP with a name-passing discipline, is to tailor π-style communication mechanisms. The cc-pi calculus [2] results from adding to the CCP model synchronous communication operands from the π-calculus. cc-pi provides a treatment of names in terms of restriction and structural axioms closer to nominal calculi than to variables with existential quantification. A distributed and probabilistic extension of CCP with networks of computational nodes, each of them with their own local store, is proposed in [1]. Nodes can *send* and *receive*, through communication channels, constraints, agents (processes) and channels themselves. In [22] CCP is endowed with *send* and *receive* primitives to allow asynchronous message-based communication. Moreover, the work in [7] defines a model of *process mobility* for CCP where localities (or sites) allow agents to have their own local store. Sites are organized in a hierarchical way and then, it is possible for an agent to have sub-agents. The reader may also refer to [5] that proposes the π^+-calculus, that extends the π-calculus with a constraint store. The language proposed here (`dccp`) offers also a model of distributed computation in CCP but, unlike the above mentioned works, we preserve the declarative reading of processes as formulas in logic (Theorem 3.10).

Universal Timed CCP (`utcc`) was proposed in [21] as an orthogonal extension of timed CCP [24] for the specification of mobile reactive systems as security protocols. Basically, `utcc` uses the interplay of **local** and **abs** processes, as described in Example 3.6. The sort of mobility that we can model in `dccp` is close to that of `utcc`. Here the reconfiguration of the communication structure is achieved by means of *logical quantification*.

Future work We plan to explore the combination of timed modalities along with spatial and epistemic ones. For that, we can rely on the encoding of such modalities in SELL$^{\Cap}$ described in [18].

From the logical point of view, we are currently exploring whether a quantification on families in the sequent calculus allows us to dynamic create new agents. This new sort of quantification is also required if we consider an infinite number of agents. To see this, note that the axioms ($\mathcal{C}[\![\Delta]\!]$) and process definitions ($\mathcal{P}[\![\Upsilon]\!]$) must be available to all the locations and agents in the system. Furthermore, we plan to explore the possibility of defining higher order CCP calculi where processes can be communicated among agents. For that, we could mark processes (not only constraints) with subexponentials and we can still give a declarative interpretation of processes as formulae in SELL$^{\Cap}$.

Acknowledgments: Olarte, Nigam and Pimentel were supported by CNPq. Pimentel was supported by FAPEMIG. This work has been (partially) supported by Colciencias (Colombia), and by Digiteo and DGAR (École Polytechnique) funds for visitors.

References

[1] Luca Bortolussi and Herbert Wiklicky. A distributed and probabilistic concurrent constraint programming language. In Maurizio Gabbrielli and Gopal Gupta, editors, *ICLP*, volume 3668 of *LNCS*, pages 143–158. Springer, 2005.

[2] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.

[3] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Carnegie Mellon University, 2003. Revised, May 2003.

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

[5] Juan Francisco Díaz, Camilo Rueda, and Frank D. Valencia. Pi+- calculus: A calculus for concurrent processes with constraints. *CLEI Electron. J.*, 1(2), 1998.

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

[7] David Gilbert and Catuscia Palamidessi. Concurrent constraint programming with process mobility. In Lloyd et al, editor, *Computational Logic*, volume 1861 of *LNCS*, pages 463–477. Springer, 2000.

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

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

[10] Rémy Haemmerlé, François Fages, and Sylvain Soliman. Closures and modules within linear logic concurrent constraint programming. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS*, volume 4855 of *LNCS*, pages 544–556. Springer, 2007.

[11] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

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

[13] Cosimo Laneve and Ugo Montanari. Mobility in the cc-paradigm. In Ivan M. Havel and Václav Koubek, editors, *MFCS*, volume 629 of *LNCS*, pages 336–345. Springer, 1992.

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

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

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

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

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

[19] Vivek Nigam, Elaine Pimentel, and Giselle Reis. Specifying proof systems in linear logic with subexponentials. *Electr. Notes Theor. Comput. Sci.*, 269:109–123, 2011.

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

[21] Carlos Olarte and Frank D. Valencia. Universal concurrent constraint programing: symbolic semantics and applications to security. In Roger L. Wainwright and Hisham Haddad, editors, *SAC*, pages 145–150. ACM, 2008.

[22] Jean-Hugues Réty. Distributed concurrent constraint programming. *Fundam. Inform.*, 34(3):323–346, 1998.

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

[24] Vijay A. Saraswat, Radha Jagadeesan, and Vineet Gupta. Timed default concurrent constraint programming. *J. Symb. Comput.*, 22(5/6):475–520, 1996.

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

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

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

[28] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Carnegie Mellon University, 2003. Revised, May 2003.