Open Bisimulation for the
Concurrent Constraint Pi-calculus[*]

Maria Grazia Buscemi[1] and Ugo Montanari[2]

[1] IMT Lucca Institute for Advanced Studies, Italy
m.buscemi@imtlucca.it

[2] Dipartimento di Informatica, University of Pisa, Italy
ugo@di.unipi.it

Abstract. The concurrent constraint pi-calculus (cc-pi-calculus) has been introduced as a model for concluding Service Level Agreements. The cc-pi calculus combines the synchronous communication paradigm of process calculi with the constraint handling mechanism of concurrent constraint programming. While in the original presentation of the calculus a reduction semantics has been proposed, in this work we investigate the abstract semantics of cc-pi processes. First, we define a labelled transition system of the calculus and a notion of open bisimilarity *à la* pi-calculus that is proved to be a congruence. Next, we give a symbolic characterisation of bisimulation and we prove that the two semantics coincide. Essentially, two processes are open bisimilar if they have the same stores of constraints - this can be statically checked - and if their moves can be mutually simulated. A key idea of the symbolic transition system is to have 'contextual' labels, i.e. labels specifying that a process can evolve only in presence of certain constraints. Finally, we show that the polyadic Explicit Fusions calculus introduced by Gardner and Wischik can be translated into monadic cc-pi and that such a transition preserves open bisimilarity. The mapping exploits fusions and tuple unifications as constraints.

1 Introduction

Service Oriented Computing is an emerging paradigm that builds upon the notion of services as interoperable elements that can be described, published, searched and composed. Services may expose both functional properties (i.e. what they do) and non-functional properties (i.e. the way they are supplied). A Service Level Agreement (SLA) is a contract between two parties, usually a service provider and a customer, that records non-functional properties about a service like performance, availability, and cost.

The concurrent constraint pi-calculus (cc-pi calculus) [5] is a model of SLA negotiations that combines two main programming paradigm: name-passing calculi (see e.g. [9]) and concurrent constraint programming [14, 13]. On the one side, cc-pi inherits from the Pi-F calculus [16] a symmetric, synchronous mechanism of interaction between senders and receivers, where the sent name is 'fused' (i.e. identified) to the received name and such *explicit fusion* allows to use interchangeably the two names.

On the other side, cc-pi generalises explicit fusions to be arbitrary constraints and introduces primitives for creating, removing and making logical checks on constraints. For instance, a cc-pi process $P = c \,|\, \text{tell } c'.Q$ can place a constraint c' corresponding to a certain SLA parameter and then evolve to the parallel composition of $c \otimes c'$ and Q, if $c \otimes c'$ is *consistent*. A process $P = c \,|\, \text{ask } c'.Q$ makes a transition to Q if the constraint c' is *entailed* by c. As another example, a process $P = (x = v \otimes v = y) \,|\, \overline{x}\langle z \rangle.P' \,|\, y\langle w \rangle.Q'$, with $\overline{x}\langle z \rangle$ an output action, $y\langle w \rangle$ an input action and $(x = v \otimes v = y)$ a combination of constraints, can make a synchronisation because the identification of the names x and y is entailed by the constraints in parallel. Moreover, such an interaction yields the name fusion $z = w$, which is consistent with the other constraints. In fact, we can think about this synchronisation as a simultaneous execution of an $\text{ask } x = y$ action and a $\text{tell } z = w$ action. From this viewpoint, cc-pi calculus combines primitives borrowed from different paradigms in a coherent way. Another feature of the cc-pi calculus is to include a restriction operation (x) *à la* pi-calculus that allows for local stores of constraints. Synchronisations may have the effect of combining local stores of interacting processes into a global store.

The constraint systems adopted in cc-pi rely on *named c-semirings*, i.e. c-semirings [3] enriched with a notion of *support* to express the relevant names of a constraint. These structures can specify networks of constraints for defining constraint satisfaction problems and to model fuzzy or probabilistic values, as well as Herbrand unifications.

A main contribution of this work is to characterise cc-pi processes that have the same behaviours. Not surprisingly, as a notion of behavioural equivalence we take *bisimulation*, which is a key idea in the context of process calculi. Roughly, two processes are bisimilar if they are able to match each other's moves. A desirable property of behavioural equivalences is that two processes are equivalent in all contexts. Indeed, this feature allows for compositional reasoning about complex interactive systems. Nevertheless, the universal quantification over all contexts makes this definition of a little use in practice. Open bisimulation [12] has been introduced on the pi-calculus as a behavioural equivalence that has a coinductive definition and is guaranteed to be a congruence. In fact, the term 'open' is meant to emphasise that in this bisimulation names can be identified at any time and, so, the relation is preserved by name substitutions.

In this paper we show that open bisimulation naturally transfers to cc-pi processes and it is still a congruence. In our setting, constraints running in parallel with a process have an effect on the names of that process as they can allow or disallow transitions. Hence, the parallel contexts consisting of constraints are as discriminating as arbitrary contexts and the natural adaptation of open bisimilarity to cc-pi is to replace substitutions with constraints in parallel. For instance, the process $\overline{x}\langle z \rangle.\mathbf{0} \,|\, y\langle w \rangle.\mathbf{0}$ that tries to synchronise on channels with different names and the inert process $\mathbf{0}$ are not bisimilar since, in the context $x = y \,|\, \_$, the first one can make a move while the second one is stuck. Beside the dynamic behaviour of processes, the present open bisimilarity takes into account the knowledge exposed by a process to its environment, that is expressed by the store of constraints of the process. This notion reminds the definition of *static equivalence* that has been defined for the applied pi-calculus [1] and it generalises to constraints a similar concept used for open bisimulation in Pi-F [15]. Note that checking whether two processes have the same static behaviour can be performed at compile-

time. As an example, consider the processes $c \mid \mathbf{0}$ and $\mathbf{0}$. They are both inert but, in the context $\texttt{ask } c \mid \_$, the first process can make a transition while the second one cannot.

Checking open bisimilarity is hard since it involves a universal quantification over constraints. We provide an efficient version of open bisimulation and we prove that the two notions coincide. The main idea behind *symbolic bisimulations* [8, 4, 12] is to define specialised transition systems, whose labels specify the minimum conditions that must hold in order for a transition to take place. We adapt this concept to our framework by defining a transition system whose labels represent the 'least restrictive' constraints that allow process moves. We exploit the division operator over c-semiring values [2], which is well defined under mild assumptions. According to the symbolic semantics, e.g., the process $c \mid \texttt{ask } d.\mathbf{0}$ can make a transition labelled by a constraint $c' = d \div c$, which is the weakest constraint such that the combination $c \otimes c'$ entails d.

The results of this work generalise to constraints analogous achievements proved for the Pi-F calculus [15]. To highlight this connection, we translate polyadic Pi-F calculus into monadic cc-pi and we prove that such a translation is fully abstract with respect to open bisimulation. This amounts to say that open bisimilarity over the instance of cc-pi corresponding to Pi-F coincides with the analogous equivalence over Pi-F processes. Note that the encoding of polyadicity exploits the fact that name tuples can be expressed as Herbrand constraints and that tuple matching corresponds to term unification.

Diaz Frias *et al.* [6] introduce the pi$^+$-calculus, an extension of the pi-calculus with constraint agents that can perform $\texttt{tell}$ and $\texttt{ask}$ actions, and they define a barbed bisimulation for the calculus. In contrast to our model, the constraint systems are first-order theories rather than algebraic structures and they do not support local stores. Gilbert and Palamidessi [7] address the interplay between mobility and constraints. Unlike our approach, they enrich concurrent constraint programming with the notion of localities and process migration rather than adding a channel-based communication mechanism *à la* pi-calculus.

2 Named constraints

Let $\mathcal{N}$ be an infinite, countable set of *names* and let $x, y, z \dots$ range over names. We define *(name) fusions* as total equivalence relations on $\mathcal{N}$ with only finitely many non-singular equivalence classes. By $x{=}y$ we denote the fusion with a unique non-singular equivalence class containing x and y. bA *substitution* is a function $\sigma : \mathcal{N} \to \mathcal{N}$. We denote by $[y/x]$ the substitution that maps x into y. A *permutation* ρ is a bijective substitution. The *kernel* $K(\rho)$ of a permutation ρ is the set of names that are changed by ρ. A *permutation algebra A* is defined by a carrier set and by a function defining how states are transformed by the finite-kernel permutations. In our case, A characterises the set of 'relevant' names of each element c of the c-semiring as the support $\texttt{supp}(c)$ in A.

We now recall basic concepts about semirings and c-semirings. We refer to [11, 3, 2] for a more detailed treatment. A *commutative semiring* is a tuple $\langle A, \oplus, \otimes, 0, 1 \rangle$ such that: (i) A is a set and $0, 1 \in A$, and $\oplus, \otimes : A \times A \to A$ are binary operators making the triples $\langle A, \otimes, 1 \rangle$ and $\langle A, \oplus, 0 \rangle$ commutative monoids (semigroups with identity), satisfying the following axioms.

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \ \forall a, b, c \in A \qquad a \otimes 0 = 0 \ \forall a \in A$$

A *constraint semiring (c-semiring)* $\langle A, \oplus, \otimes, 0, 1 \rangle$ is a commutative semiring such that $\oplus$ is idempotent and such that $a \oplus 1 = 1$ for all $a \in A$ (i.e. with top element). Typical examples are the c-semiring for classical constraint satisfaction problems $\langle \{\mathsf{False}, \mathsf{True}\}, \vee, \wedge, \mathsf{False}, \mathsf{True} \rangle$, the c-semiring for fuzzy constraint satisfaction problems $\langle [0,1], \mathsf{max}, \mathsf{min}, 0, 1 \rangle$, and the c-semiring of weighted constraint satisfaction problems $\langle \mathbb{R}^+ \cup \{+\infty\}, \mathsf{min}, +, +\infty, 0 \rangle$. Note that the Cartesian product of two c-semirings is a c-semiring, hence this framework is also suited to model multicriteria optimization.

Commutative semirings with top element are also known in the literature as *absorptive*. Absorptiveness implies that the sum operator is idempotent. Semirings that satisfy this last property are often called *tropical*. Hence, c-semirings are tropical semirings with top element. Next, we briefly overview some classical notions and results on absorptive and tropical semirings that we rephrase for c-semirings.

Let $\preceq$ be a relation over A such that $a \preceq b$ iff $a \oplus b = b$. This relation gives us a way to compare semiring values and constraints. Assume a c-semiring $\mathcal{C} = \langle A, \oplus, \otimes, 0, 1 \rangle$. Then: (i) $\preceq$ is a partial order; (ii) $\oplus$ and $\otimes$ are monotone on $\preceq$; (iii) $a \otimes b \preceq a, b$, for all a, b; (iv) 0 is its minimum and 1 its maximum; and (v) for all $a, b \in A$, $a \oplus b$ is the least upper bound of a and b. Moreover, if $\otimes$ is idempotent, $a \otimes b$ is the greatest lower bound of a and b. $\mathcal{C}$ is *invertible* if there exists an element $c \in A$ such that $b \otimes c = a$ for all elements $a, b \in A$ such that $a \preceq b$; $\mathcal{C}$ is *complete* if it is closed with respect to infinite sums, and the distributivity law holds also for an infinite number of summands. It can be proved that if $\mathcal{C}$ is complete then the set $\{x \in A \mid b \otimes x \preceq a\}$ admits a maximum for all elements $a, b \in A$, denoted $a \div b$. Note that the idempotency of $\otimes$ implies that the invertibility property holds. However, for the purpose of this paper, we simply require invertibility and completeness while not imposing idempotency of $\otimes$.

2.1 Named c-semirings

A named c-semiring is a complete and invertible c-semiring enriched with a notion of name fusions, a permutation algebra A and a hiding operator $(\nu x.)$ that makes a name x local in c. Note that in certain named c-semirings the hiding operator coincides with the homologous operator $\exists x$ defined in $\mathtt{cc}$ programming. Formally, a *named c-semiring* $\mathcal{C} = \langle C, \oplus, \otimes, \nu x., \rho, 0, 1 \rangle$ is a tuple where: (i) $x{=}y \in C$ for all x and y in $\mathcal{N}$; (ii) $\langle C, \oplus, \otimes, 0, 1 \rangle$ is a complete and invertible c-semiring; (iii) $\langle C, \rho \rangle$ is a finite-support permutation algebra such that every permutation ρ distributes over $\otimes$ and $\oplus$ and is inactive on 0 and 1 ; (iv) $\forall x$, $\nu x. : C \to C$ is a unary operation; (v) for all $c, d \in C$ and for all ρ the following axioms hold.

$$
\begin{array}{llll}
x{=}y \otimes c & = & x{=}y \otimes [y/x]\,c & \qquad \rho\,(\nu x.c) = \nu x.\,(\rho\,c)\ \text{if}\ x \notin K(\rho) \\
\nu x.\,1 = 1 & & \nu x.\,\nu y.\,c = \nu y.\,\nu x.\,c & \qquad \nu x.\,c = \nu y.\,[y/x]\,c \quad \text{if}\ y \notin \mathsf{supp}(c) \\
\nu x.\,(c \otimes d) = c \otimes \nu x.d & \text{if}\ x \notin \mathsf{supp}(c) & & \nu x.\,(c \oplus d) = c \oplus \nu x.d \quad \text{if}\ x \notin \mathsf{supp}(c)
\end{array}
$$

The top left axiom above accounts for combining fusions and generic elements of c-semirings According to the top right axiom, the order of ρ and ν can be changed if x is not affected by ρ. The remaining axioms rule how the ν operation interacts with the operations of the c-semiring and they are inspired by the analogous structural congruence axioms for restriction in process calculi. Note that the notion of support $\mathsf{supp}(c)$ associated with permutation algebras recalls the concept of free names in process calculi.

Given $\mathcal{C} = \langle A, \oplus, \otimes, \rho, \vee x., 0, 1 \rangle$, a *(named) constraint* c is an element of A. For $C \subseteq A$, C is *consistent* if $(\otimes C) \neq 0$; moreover, for $c \in A$, C *entails* c if $(\otimes C) \leq c$.

Herbrand constraints A *Herbrand constraint system* can be defined by considering a signature Σ along with an equational theory $=_E$ on the term algebra $T_\Sigma(\mathcal{N})$ plus the additional rules:

$$\text{(SUB-TERM)} \quad \frac{f(t_1,\ldots,t_n) =_E f(t_1',\ldots,t_n')}{t_i =_E t_i'} \quad i = 1,\ldots,n \qquad \text{(REPLACE)} \quad \frac{x =_E t \quad t_1 =_E t_2}{[t/x]t_1 =_E [t/x]t_2}$$

and with the restrictions that $x \neq_E t(x)$ and $f(t_1,\ldots,t_n) \neq_E g(t_1,\ldots,t_m)$, where $t(x)$ is any term different than x which contains x and $f \neq g$. Axiom (SUB-TERM) above allows to reduce the unification of two terms to the unification of their sub-terms provided that the outer function symbols are the same. Axiom (REPLACE) reduces the unification of two terms containing a term t such that $t =_E x$ to the unification of the terms with x in place of t. The restrictions prevents from unifying, respectively, a variable with terms containing that variable, and two terms containing a distinct outer function symbols.

We let $\mathcal{C}_H$ be the tuple $\mathcal{C}_H = \langle C, \oplus, \otimes, \vee x., \rho, 0, 1 \rangle$ where: (i) C is the set of the above-defined equational theories plus a bottom element $\perp$; (ii) $E_1 \oplus E_2 = E_1 \cap E_2$; (iii) $E_1 \otimes E_2$ is the unification of E_1 and E_2, i.e. it is the smallest equational theory largest than or equal to $E_1 \cup E_2$, if it exists, otherwise $\perp$; (iv) $\vee x. E = E \cap \bar{E}$, where $t_1 =_{\bar{E}} t_2$ if $t_1 =_E t_2$ or x does not occur in t_1, t_2; (v) $\rho t_1 =_{\rho E} \rho t_2$ if $t_1 =_E t_2$; (vi) $0 = \perp$ and $1 = \{(t,t) \mid t \in T_\Sigma(\mathcal{N})\}$. $\mathcal{C}_H$ can be proved to be a named c-semiring with idempotent product $\otimes$.

Example 1 (pairs, tuples). Pairs of names can be expressed as elements of the named c-semiring $\mathcal{C}_H$ by assuming two sorts, names and lists, and by taking the signature $\Sigma = \{ (\_, \_), \mathsf{nil} \}$, where nil is a constant of sort '$\rightarrow$ lists' and $(\_, \_)$ is a binary operation of sort 'names $\times$ lists $\rightarrow$ lists'. A tuple of arity n can be defined as $\langle x_1, \ldots, x_n \rangle = (x_1, (x_2, (\ldots (x_{n-1}, \mathsf{nil}))\ldots))$. Notice that, for instance, the unification of two theories of different arities $\langle x_1, x_2 \rangle$ and $\langle y_1, y_2, y_3 \rangle$ reduces to unifying the subterms nil and (y_3, nil) hence leading to $\perp$, since the outer functions are distinct. On the other hand, the unification of $\langle x_1, x_2, x_3 \rangle$ and $\langle y_1, y_2, y_3 \rangle$ yields the identification of the components $x_i = y_i$.

Soft constraints Given a domain D of interpretation for the set of names $\mathcal{N}$ and a c-semiring $S = \langle A, \oplus, \otimes, 0, 1 \rangle$, a *soft* constraint c can be represented as a function $c = (\mathcal{N} \rightarrow D) \rightarrow A$ associating to each variable assignment $\eta = \mathcal{N} \rightarrow D$ (i.e. instantiation of the variables occurring in it) a value in A, which can be interpreted e.g. as a set of preference values or costs. Soft constraints can be combined by means of the operators of S. Assume $\mathcal{C}_{\mathsf{soft}}$ is the tuple $\mathcal{C}_{\mathsf{soft}} = \langle C, \oplus', \otimes', \vee x., \rho, 0', 1' \rangle$ such that: (i) C is the set of all soft constraints over $\mathcal{N}$, D and S; (ii) name equalities $x=y$ are defined as $(x = y)\eta = 1$ if $\eta(x) = \eta(y)$, $(x = y)\eta = 0$ otherwise; (iii) $(c_1 \oplus' c_2)\eta = c_1\eta \oplus c_2\eta$; (iv) $(c_1 \otimes' c_2)\eta = c_1\eta \otimes c_2\eta$; (v) $(\vee x.c)\eta = \sum_{d \in D} (c\eta\{d/x\})$, where $\sum_{d \in D}$ denotes the c-semiring sum operator and the assignment $\eta\{d/x\}$ is defined, as usual, as $\eta\{d/x\}(y) = d$ if $x = y$, $\eta(y)$ otherwise; (vi) $(\rho c)\eta = c\bar{\eta}$ with $\bar{\eta}(x) = \eta(\rho(x))$; (vii) $0'\eta = 0$ and

$1'\eta = 1$ for all η. It is possible to prove that $\mathcal{C}_{\text{soft}}$ is indeed a named c-semiring and that the product $\otimes'$ is idempotent provided that $\otimes$ is idempotent. Remark that for $S = \langle \{\mathsf{False}, \mathsf{True}\}, \vee, \wedge, \mathsf{False}, \mathsf{True} \rangle$, the named constraints of $\mathcal{C}_{\text{soft}}$ leads to solutions consisting of the set of tuples of legal domain values.

3 The cc-pi calculus

The concurrent constraint calculus (cc-pi calculus) features symmetric non-binding input and output actions like in Pi-F calculus along with primitives for constraint handling. The syntax and reduction semantics of the calculus are defined in Fig. 1. Unlike the original presentation of cc-pi [5], here we give a monadic version of the calculus. Moreover, we disregard the `check` and `retract` operators. In fact, `check` is irrelevant for the purpose of this paper but it could be easily included; by contrast, adding `retract` would not be trivial since it would require dealing with a more complex constraint theory. The cc-pi calculus is parametric with respect to named constraints. We let $c, d, e \ldots$ range over constraints of an arbitrary named c-semiring $\mathcal{C}$. The notions of *bound names*, *free names*, and α-conversion of a process are as usual apart that the occurrence of the name y in a process with an input prefix $x\langle y\rangle.U$ is free and that the set of free names is extended to constraints by adding the following clauses:

$$\mathrm{fn}(\pi.U) = \mathrm{supp}(c) \cup \mathrm{fn}(U) \ \text{ if } \pi = \mathtt{tell}\ c, \mathtt{ask}\ c \qquad \mathrm{fn}(c) = \mathrm{supp}(c)$$

The last three structural axioms in Fig. 1 state the correspondence between parallel composition and semiring product, restriction and constraint hiding, the inert process **0** and the top element of c-semiring 1, respectively. Using structural congruence, every process P can be rewritten into the *normal form* $P \equiv (\widetilde{x})\,(c\,|\,U)$, where c is a constraint, U can only contain restrictions under prefixes, i.e. $U \not\equiv (\widetilde{y})U'$, and if $x_i \in \mathrm{supp}(c)$ then $x_i \in \mathrm{fn}(U)$. Roughly, the rules move each name $x \notin \mathrm{fn}(U)$ close to c and then apply νx. to c.

The idea behind the reduction relation is to proceed as follows. First, to put processes into the normal form by applying the rule for structural congruent processes. Next, applying the rules for dealing with primitives on constraints or for synchronising processes. Afterward, closing with respect to summation, parallel composition of unconstrained processes, and restriction. For instance, the parallel composition $x = z\,|\,\overline{x}\langle y\rangle.U\,|\,z\langle w\rangle.V$ can evolve to $x = z \otimes y = w\,|\,U\,|\,V$ since the equality of the names x and z is entailed by the constraint $x = z$ and the store $x = z \otimes y = w$ is consistent. Remark that it is legal to treat name equalities as constraints c over $\mathcal{C}$ because, by definition, named c-semirings contain fusions. Note also that the rule for parallel composition intentionally allows to add only unconstrained process. For this reason, several rules like those for τ's and summation must include the constraint c in parallel.

4 A labelled semantics for cc-pi

We now come to the first contribution of this work. We propose a labelled semantics that coincides with the reduction semantics when restricting to closed processes. We start by introducing the notion of *store* of constraints of processes, that represents the static knowledge exposed by a process to its environment. Roughly, $\mathrm{store}(P)$ is the constraint

The syntax of *prefixes* π, *unconstrained processes* U and *constrained processes* P is:

$$
\begin{array}{lllllllllll}
\pi & ::= & \tau & | & \overline{x}\langle y\rangle & | & x\langle y\rangle & | & \texttt{tell } c & | & \texttt{ask } c \\
U & ::= & \mathbf{0} & | & U|U & | & \sum_i \pi_i.U_i & | & (x)U & | & I(\widetilde{y}) \\
P & ::= & U & | & c & | & P|P & | & (x)P
\end{array}
$$

The *structural congruence*, $\equiv$, is the smallest congruence over processes closed with respect to α-conversion and satisfying the following axioms:

$$P|Q \equiv Q|P \quad (P|Q)|R \equiv P|(Q|R) \quad (x)(y)P \equiv (y)(x)P \quad P|(x)Q \equiv (x)(P|Q) \ \ if \ x \notin \mathrm{fn}(P)$$

$$I(\widetilde{y}) \equiv [\widetilde{y}/\widetilde{x}]U \quad if \ I(\widetilde{x}) \stackrel{\mathrm{def}}{=} U \qquad c|d \equiv c \otimes d \qquad (x)c \equiv \nu x.c \qquad \mathbf{0} \equiv 1$$

The *reduction relation* over processes $\mapsto$ is the smallest relation satisfying the following rules:

$$c\,|\,\tau.U \mapsto c\,|\,U \qquad c\,|\,\texttt{tell } d.U \mapsto c \otimes d\,|\,U \quad if \ c \otimes d \neq 0 \qquad c\,|\,\texttt{ask } d.U \mapsto c\,|\,U \ if \ c \preceq d$$

$$c\,|\,(\overline{x}\langle y\rangle.U + \sum \pi_i.U_i)\,|\,(z\langle w\rangle.V + \sum \pi'_j.V_j) \longrightarrow c \otimes (y = w)\,|\,U\,|\,V \quad if \ c \otimes (y=w) \neq 0 \wedge c \preceq x = z$$

$$\frac{c\,|\,\pi_i.U_i \mapsto P}{c\,|\,\sum \pi_i.U_i \mapsto P} \qquad \frac{P \mapsto P'}{P|U \mapsto P'|U} \qquad \frac{P \mapsto P'}{(x)P \mapsto (x)P'} \qquad \frac{P \equiv P' \quad P' \mapsto Q' \quad Q' \equiv Q}{P \mapsto Q}$$

Fig. 1. The cc-pi calculus

which is obtained by replacing each unconstrained process occurring in P with $\mathbf{0}$ and by applying the structural axioms on constraints to compute the resulting constraint. More formally, for P a process, $\mathsf{store}(P)$ is inductively defined as follows:

$$\mathsf{store}(c) = c \quad \mathsf{store}(P|Q) = \mathsf{store}(P) \otimes \mathsf{store}(Q) \quad \mathsf{store}(U) = 1 \quad \mathsf{store}((x)P) = \nu x.\,\mathsf{store}(P)$$

For example, if $P = (x)(y = x\,|\,x = z\,|\,c(x,v)\,|\,(w)y\langle w\rangle.U)$, $\mathsf{store}(P) = y = z \otimes c(y,v)$. Note that the concept of store is close to the notion of 'frame' given in applied pi-calculus [1] and it generalises to constraints the equivalence relation that characterises the explicit fusions of a process in Pi-F calculus [15].

Assume a set of actions $\mathcal{A} = \{\tau, \overline{x}\langle y\rangle, x\langle y\rangle, \overline{x}(z), x(z)\}$, where τ is a silent action, $\overline{x}\langle y\rangle$ and $x\langle y\rangle$ are *free actions*, and $\overline{x}(z)$ and $x(z)$ are *bound actions*. We let α, β range over $\mathcal{A}$. The labelled transition semantics of processes is the smallest relation $P \stackrel{\alpha}{\rightarrow} Q$, defined by the rules in Fig. 2 plus a rule (OPEN-O) for output and the symmetric counterpart of rule (COMM). The operational rules that deal with τ-transitions are analogous to the reduction rules given in the previous section. The additional rules are standard apart that the usual side condition $x \neq z$ of the rule (OPEN-I) is replaced by the condition that $x = z$ cannot be entailed by the store of constraints of the process. By $\stackrel{\tau}{\rightarrow}^{\star}$ we refer to a sequence of transitions $\stackrel{\tau}{\rightarrow}$.

Proposition 1. *Let P be a process. $P \mapsto Q$ iff $P \stackrel{\tau}{\rightarrow} Q$.*

4.1 Example: modelling Service Level Agreements

We now consider a variant of an example introduced in [5] that does not include `retract` operations. Consider a service that offers computing resources like units of CPUs of a

(PREF)
$$c\,|\,\pi.U \xrightarrow{\pi} c\,|\,U \quad \pi = \tau, \overline{x}\langle y\rangle, x\langle y\rangle$$

(TELL)
$$c\,|\,\texttt{tell}\ d.U \xrightarrow{\tau} c\otimes d\,|\,U \ \ \text{if } c\otimes d \neq 0$$

(ASK)
$$c\,|\,\texttt{ask}\ d.U \xrightarrow{\tau} c\,|\,U \quad \text{if } c \preceq d$$

(SUM)
$$\frac{c\,|\,\pi_i.U_i \xrightarrow{\alpha} U'}{c\,|\,\sum \pi_i.U_i \xrightarrow{\alpha} U'}$$

(PAR)
$$\frac{P \xrightarrow{\alpha} P' \quad \mathrm{bn}(\alpha)\cap\mathrm{fn}(U) = \emptyset}{P\,|\,U \xrightarrow{\alpha} P'\,|\,U}$$

(COMM)
$$\frac{c\,|\,U \xrightarrow{x\langle y\rangle} c\,|\,U' \quad c\,|\,V \xrightarrow{\overline{z}\langle w\rangle} c\,|\,V' \quad c\otimes(y=w)\neq 0 \quad c\preceq x=z}{c\,|\,U\,|\,V \xrightarrow{\tau} c\otimes(y=w)\,|\,U'\,|\,V'}$$

(RES)
$$\frac{P \xrightarrow{\alpha} P' \quad x \notin \mathrm{n}(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}$$

(OPEN-I)
$$\frac{P \xrightarrow{z\langle x\rangle} P' \quad \mathrm{store}(P) \npreceq x=z}{(x)P \xrightarrow{z(x)} P'}$$

(STRUCT)
$$\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q}$$

Fig. 2. Labelled Transition System of the cc-pi calculus

given power and suppose the service provider and a client want to reach a SLA. The provider and the client can be described by the following cc-pi processes.

$$\mathsf{Client_{req}}(r) \quad \equiv (y)\,(\texttt{tell}\ c_{\mathsf{req}}(y).\overline{r}\langle y\rangle)$$
$$\mathsf{Provider_{off,N}}(r) \equiv (x_0)\,(\texttt{tell}\ (x_0 = \mathsf{N}).\mathsf{Ac\_Req_{off}}(x_0, r))$$
$$\mathsf{Ac\_Req_{off}}(x,r) \equiv (v)\,(x')\,(\texttt{tell}\ (x' = x - v \otimes x' \geq 0).\texttt{tell}\ d_{\mathsf{off}}(v)).\,r\langle v\rangle.\mathsf{Ac\_Req_{off}}(x',r))$$

The client starts by placing a constraint $c_{\mathsf{req}}(y)$ that specifies that the requested resources y must be at least req, then it contacts the provider on channel r. If the synchronisation succeeds, the negotiation is concluded. On the other side, the provider initially fixes the maximum number of total available resources N, then starts to accept requests by imposing two constraints on the number of resources v that will be allocated to each client: $(x' = x - v) \otimes x' \geq 0$ states that v must be less than the total available resources (x is initially N) and that the remaining resources are $x' \geq 0$; $d_{\mathsf{off}}(v)$ specifies that v must be less than a fixed maximum number of resources off that are offered to every client. Finally, the provider tries to reach an agreement with a client over channel r and, in case of success, is ready to offer the remaining resources x' to other clients. The negotiation between a client $\mathsf{Client_{req}}(r)$ and a provider $\mathsf{Provider_{off,N}}(r)$ can be modelled in cc-pi as follows. First, each party consumes its $\texttt{tell}$ prefixes:

$$(r)\,(\mathsf{Provider_{off,N}}(r)\,|\,\mathsf{Client_{req}}(r)) \xrightarrow{\tau}{}^{*} (r)\,(y)\,(x_0)\,(v)\,(x')\,(c_{\mathsf{req}}(y)\otimes d_{\mathsf{off}}(v)\otimes(x_0 = \mathsf{N}) \otimes$$
$$(x' = x_0 - v \otimes x' \geq 0)\,|\,\overline{r}\langle y\rangle\,|\,r\langle v\rangle.\mathsf{Ac\_Req_{off}}(x',r))$$

If the fusion $y = v$ yields a consistent store of constraint, the synchronisation on r can take place and the two parties have reached an agreement expressed by the constraint

$$c_{\mathsf{req}}(y) \otimes d_{\mathsf{off}}(v) \otimes (x_0 = \mathsf{N}) \otimes (x' = x_0 - v) \otimes (x' \geq 0) \otimes (y = v) \qquad (1)$$

As mentioned in the previous section, the cc-pi calculus is parametric with respect to named constraints. Hence, in cc-pi we can capture different constraint satisfaction problems by changing the underlying named c-semiring of a given process while keeping the same process specification. To highlight this feature, we now consider two instantiations

of the constraint system adopted in the above negotiation scenario and we show that they lead to different solutions. In both cases, we assume an assignment $\eta : \mathcal{N} \to \mathbb{N}$ of names to non-negative integers and we take the constraints to be functions $c : (\mathcal{N} \to \mathbb{N}) \to A$ that map each name assignment to a value of a c-semiring $S = \langle A, \oplus, \otimes, 0, 1 \rangle$. Constraints are combined by using the operations of S, as shown in §2.1. The two cases below correspond to different choices of the c-semiring S.

Crisp constraint interpretation Consider $S = \langle \{\mathsf{False}, \mathsf{True}\}, \vee, \wedge, \mathsf{False}, \mathsf{True} \rangle$ that leads to solutions consisting of the set of tuples of legal domain values. For instance, the interpretation of the constraint $c = (\mathsf{a} \geq x) \otimes (y \geq \mathsf{b})$, where x, y are names, a and b are in $\mathbb{N}$, and $\geq$ stands for 'greater than or equal' over $\mathbb{N}$, is that $c\eta = \mathsf{True}$ if $\mathsf{a} \geq \eta(x)$ and $\eta(y) \geq \mathsf{b}$, while $c\eta = \mathsf{False}$ otherwise. We define the constraints c_{req} and d_{off} as follows:
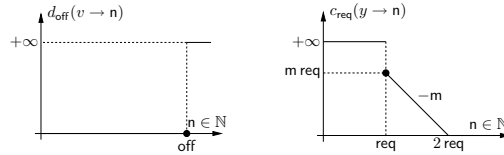
$$c_{\mathsf{req}}(y) \stackrel{\text{def}}{=} y \geq \mathsf{req} \qquad d_{\mathsf{off}}(v) \stackrel{\text{def}}{=} \mathsf{off} \geq v.$$

Assuming the interpretation of name equalities is as expected, the store of constraints (1) has a solution if $\min\{\mathsf{off}, \mathsf{N}\} \geq \mathsf{req}$. For instance, $\mathsf{Provider}_{7,15}$ can reach an agreement with Client_5, but not with Client_8. Moreover, the constraint system resulting from the negotiation between a provider and n clients has a solution if
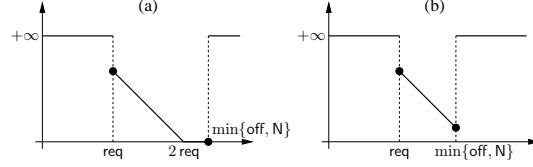
$$\mathsf{req}_i \leq \mathsf{off} \quad \text{and} \quad \sum_i \mathsf{req}_i \leq \mathsf{N} \quad \text{for} \quad i = 1, \dots, n.$$

For example, if there are three clients and each of them requests at least 6 units of resources, a provider $\mathsf{Provider}_{7,15}$ can only reach an agreement with two of them.

Weighted constraint interpretation Consider the c-semiring of weighted constraint satisfaction problems $S_W = \langle \mathbb{R}^+ \cup \{+\infty\}, \min, +, 0, +\infty \rangle$, which associates a cost to each domain tuple. In our example, this c-semiring allows to model the viewpoint of the client that wishes to minimise the total cost of the proposed solution. Note that in this case the associated ordering $\preceq$ over constraints reduces to $\geq$ over reals, i.e. a value is preferred to another if it is smaller. The interpretation of a named constraint $c = (x = y)$ is that, for η an assignment of names to non-negative integers, $c\eta = 0$ if $\eta(x) = \eta(y)$, while $c\eta = +\infty$ otherwise. The constraints $y = 0$, $x_0 = \mathsf{N}$, $x' = x_0 - v$, and $y = v$, where 0 and N are values in the domain $\mathbb{N}$, can be interpreted similarly. We define the constraints d_{off} and c_{req} as below.

The constraint d_{off} is a simple translation of the analogous constraint given in the crisp case, i.e. $d_{\mathsf{off}}(v \to \mathsf{n}) = 0$ if $\mathsf{off} \geq \mathsf{n}$, while $d_{\mathsf{off}}(v \to \mathsf{n}) = +\infty$ otherwise. On the other side, the constraint $c_{\mathsf{req}}(y)$ specifies that: (i) if y assumes a value that is less than req, then the cost is maximum; (ii) if the value of y is between req and $2\mathsf{req}$, the cost decreases according to the slope $-\mathsf{m}$; (iii) for every value that is greater than $2\mathsf{req}$ the cost is minimum, meaning that the client has no additional benefit in acquiring more than $2\mathsf{req}$ resources. The possible solutions of the constraint system (1) are as follows.

If $\min\{\mathsf{off},\mathsf{N}\} < \mathsf{req}$, the system has no solution. If $\min\{\mathsf{off},\mathsf{N}\} \geq 2\mathsf{req}$ the system yields a set of solutions which all have the maximum level of preference 0 (case (a)). Finally, if $\mathsf{req} \leq \min\{\mathsf{off},\mathsf{N}\} < 2\mathsf{req}$ the solution is selected by means of the c-semiring $\oplus$ operation (min over reals) in that interval, thus leading to $\min\{\mathsf{off},\mathsf{N}\}$ (case (b)).

Remark 1. The present semantics does not specify how to solve the constraint system generated at each step. In fact, the consistency check performed when placing new constraints, either through a `tell` action or by a synchronisation, only requires that the resulting constraint is different from the bottom element of the c-semiring 0. While in the crisp case this choice amounts to take optimal solutions (i.e., name instantiations that lead to constraint evaluating to the top element 1), in the more general setting of soft constraints this semantics does not provide a way to discard non-optimal solutions.

5 Open bisimulation

We now define a process equivalence *à la* open pi-calculus bisimulation. In our setting, the obvious counterpart of closing with respect to substitutions is to close with respect to constraints in parallel. We require that two equivalent processes have the same static and dynamic behaviour. Hereafter, by $c \preceq (\alpha = \beta)$ we abbreviate: (i) $c \preceq (x=y) \otimes (w=z)$, if $\alpha = \overline{x}\langle w \rangle$ and $\beta = \overline{y}\langle z \rangle$ (and analogously for input actions), (ii) $c \preceq (x=y)$, if $\alpha = \overline{x}(w)$ and $\beta = \overline{y}(w)$, where $\preceq$ and $\otimes$ are the partial order and the product operations of c-semirings; $c \preceq \tau = \tau$ stands for $c \preceq 1$.

Definition 1 (open bisimilarity). Open bisimilarity $(\sim^o)$ *is the largest symmetric relation S between processes such that $P S Q$ implies:*

1. $\mathsf{store}(P) = \mathsf{store}(Q)$;
2. *If $P \xrightarrow{\alpha} P'$ with $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$ then $Q \xrightarrow{\beta} Q'$ and $P' S Q'$, for some Q' and β such that $\mathsf{store}(P) \preceq \alpha = \beta$;*
3. $c \,|\, P S c \,|\, Q$, *for all constraints $c \neq 0$.*

The first item above states that two processes must expose the same stores. This requirement does not take dynamic behaviours of processes into account. From this viewpoint, the equivalence of stores is a generalisation to constraints of the *static equivalence* defined in applied pi-calculus [1]. The condition $\mathsf{store}(P) \preceq \alpha = \beta$ in the second item intuitively means that the label identification must be entailed by the constraints of P, i.e. the labels must coincide in the 'context' $\mathsf{store}(P)$. For instance, $x = y \,|\, \overline{x}\langle z \rangle$ and $x = y \,|\, \overline{y}\langle z \rangle$ satisfy this requirement. Finally, the third item of the above definition is the counterpart of closing with respect to substitutions in open bisimulation for pi-calculus processes. As an example, the processes $\mathsf{ask}\, c.U$ and $\mathbf{0}$ are not bisimilar because $c \,|\, \mathsf{ask}\, c.U$ can make a move while $c \,|\, \mathbf{0}$ is stuck. In fact, it is possible to show that

$\sim^o$ coincides with a classical notion of contextual equivalence. We now state that $\sim^o$ is preserved by every operator of the calculus. The proof is a generalisation of the proofs of the analogous results on open pi-calculus without restriction and Pi-F calculus.

Theorem 1. $\sim^o$ *is a congruence.*

Example 2. Consider the example depicted in § 4.1. Suppose there are two providers $\mathsf{Provider}_{4,40}$ and $\mathsf{Provider}_{3,40}$ that have the same amount of available resources but that offer different units of resources to each client. Obviously, these two providers behave differently when interacting with other processes. For instance, only the first provider can reach an agreement with a client that requires at least 4 resources, i.e. $\mathsf{Client}_4 \,|\, \mathsf{Provider}_{4,40}$ can make a τ-transition while $\mathsf{Client}_4 \,|\, \mathsf{Provider}_{3,40}$ is stuck. It can be easily shown that the two processes are not open bisimilar. After $\mathsf{Provider}_{4,40}$ and $\mathsf{Provider}_{3,40}$ place their own constraints, they both evolve to the processes

$$(r)(x_0)(v)(x')(x_0 = 40 \otimes x' = x_0 - v \otimes x' \geq 0 \otimes v \leq \mathsf{off}_i \,|\, r\langle v\rangle.\mathsf{Ac\_Req}_{\mathsf{off}_i}(x',r))$$

where off_i are 4 and 3, respectively. Since the names are all bound, the stores are both empty and, hence, the target processes cannot be distinguished. However, after the processes 'extrude' the name v over r, their respective stores become

$$(r)(x_0)(x')(x_0 = 40 \otimes x' = x_0 - v \otimes x' \geq 0 \otimes v \leq \mathsf{off}_i)$$

which are equal to $v \leq \min\{40, \mathsf{off}_i\}$. Thus, the stores of the two processes are now distinguished and the first condition of Def. 1 does not hold. Similarly, we can show that the processes $\mathsf{Provider}_{4,40}$ and $\mathsf{Provider}_{4,32}$ are not bisimilar.

5.1 Symbolic characterisation

We now give an efficient version of open bisimulation and we prove that the two notions coincide. For lack of space, we omit the proof of this result. Let $\mathcal{L} = \mathcal{A} \cup \{c \,|\, c \text{ is a named constraint}\}$, ranged over by λ, be a set of labels, and assume the label τ coincides with the top element of the named c-semiring 1.

We define a transition system whose transitions are of the form $P \overset{\lambda}{\rightarrowtail} Q$, where λ can be either a standard label or a constraint e that is the *maximal* element (according to $\preceq$) such that $e \,|\, \mathsf{store}(P)$ allows P to evolve to Q. Recall that $c \preceq d$ intuitively means that d is 'less restrictive' than c. Hence, transitions that are labelled by maximal constraints specify minimal conditions.

The *symbolic transition semantics* of processes is the smallest relation $P \overset{\lambda}{\rightarrowtail} Q$, defined by the rules in Fig. 3 plus a rule (S-OPEN-O) for output and the symmetric version of rule (S-COMM). The symbolic transition system is the same as its 'concrete' counterpart but for rules (S-ASK) and (S-COMM). According to rule (S-ASK), a process $\mathsf{ask}\ d.U$ in parallel with a constraint c can make a transition labelled by the least restrictive constraint whose combination with c entails d, i.e. by the maximal element of the set $\{x \in A \,|\, c \otimes x \preceq d\}$. The assumption on the completeness of c-semirings ensures that such an element, noted $d \div c$, exists and is unique (see § 2). Remark that, in general, it does not hold that $d \div c \otimes c = d$. Rule (S-COMM) follows the same intuition, though in this case the condition that must be entailed is $x = z$.

$$
\begin{array}{ll}
\text{(S-PREF)} & \text{(S-TELL)} \\[4pt]
c\,|\,\pi.U \xrightarrowtail{\pi} c\,|\,U \quad \pi = \tau, \overline{x}\langle y\rangle, x\langle y\rangle & c\,|\,\mathtt{tell}\ d.U \xrightarrowtail{\tau} c\otimes d\,|\,U \ \text{ if } c\otimes d \neq 0
\end{array}
$$

$$
\begin{array}{ll}
\text{(S-ASK)} & \text{(S-SUM)} \qquad\qquad \text{(S-PAR)} \\[4pt]
c\,|\,\mathtt{ask}\ d.U \xrightarrowtail{d\div c} c\,|\,U &
\dfrac{c\,|\,\pi_i.U_i \xrightarrowtail{\lambda} U'}{c\,|\,\sum \pi_i.U_i \xrightarrowtail{\lambda} U'} \qquad
\dfrac{P \xrightarrowtail{\lambda} P' \quad \mathrm{bn}(\lambda)\cap \mathrm{fn}(U) = \emptyset}{P\,|\,U \xrightarrowtail{\lambda} P'\,|\,U}
\end{array}
$$

$$
\begin{array}{ll}
\text{(S-COMM)} & \text{(S-RES)} \\[4pt]
\dfrac{c\,|\,U \xrightarrowtail{x\langle y\rangle} c\,|\,U' \quad c\,|\,V \xrightarrowtail{\overline{z}\langle w\rangle} c\,|\,V' \quad c\otimes (y=w) \neq 0}{c\,|\,U\,|\,V \xrightarrowtail{(x=z)\div c} c\otimes (y=w)\,|\,U'\,|\,V'} &
\dfrac{P \xrightarrowtail{\lambda} P' \quad x \notin \mathrm{n}(\lambda)}{(x)P \xrightarrowtail{\lambda} (x)P'}
\end{array}
$$

$$
\begin{array}{ll}
\text{(S-OPEN-I)} & \text{(S-STRUCT)} \\[4pt]
\dfrac{P \xrightarrowtail{z\langle x\rangle} P' \quad \mathrm{store}(P) \npreceq x = z}{(x)P \xrightarrowtail{z(x)} P'} &
\dfrac{P \equiv P' \quad P' \xrightarrowtail{\lambda} Q' \quad Q' \equiv Q}{P \xrightarrowtail{\lambda} Q}
\end{array}
$$

Fig. 3. Symbolic Semantics of the cc-pi calculus

Definition 2 (symbolic bisimilarity). Symbolic (open) bisimilarity ($\sim^s$) *is the largest symmetric relation S between processes such that $P S Q$ implies:*

1. $\mathrm{store}(P) = \mathrm{store}(Q)$;
2. *If $P \xrightarrowtail{\alpha} P'$ with $\alpha \neq \tau$ and $\mathrm{bn}(\alpha)\cap\mathrm{fn}(Q) = \emptyset$ then $Q \xrightarrowtail{\beta} Q'$ and $P' S Q'$, for some Q' and β such that $\mathrm{store}(P) \preceq \alpha = \beta$;*
3. *If $P \xrightarrowtail{c} P'$ then $Q \xrightarrowtail{d} Q'$ and $c\,|\,P' S c\,|\,Q'$, for some Q' and d such that $c \preceq d$.*

The last condition above reminds the analogous clause of symbolic open pi-calculus in which a process that can evolve under a certain condition can be simulated by another process that can make a transition labelled with a weaker condition.

Example 3. Consider again the example shown in § 4.1. Assume two providers $\mathrm{Provider}_{10,8}$ and $\mathrm{Provider}_{15,8}$ that offer different amounts of resources to each client but that have the same number of remaining resources. Trivially, these two processes are equivalent, because they can only satisfy clients that require maximum 8 resources. However, proving that the two processes are open bisimilar requires checking their behaviour in presence of any constraint. Let us see why they are symbolically bisimilar. Initially, the two processes place their own constraints and evolve to processes that have the same empty store of constraints. Next, when the number of offered resources v is communicated the stores of constraints of the processes become $v \leq \min\{\mathrm{off}_i, 8\}$, which admit the same solutions regardless off_i is 15 or 10.

Theorem 2. *Symbolic bisimilarity $\sim^s$ and open bisimilarity $\sim^o$ coincide.*

6 Embedding polyadic Pi-F calculus

We start by recalling the Pi-F calculus. For better relating the calculus with cc-pi, we present the Pi-F in the standard pi-calculus fashion rather than in the 'commitment'

style [16]. Assume the set of labels $\mathcal{M} = \{\tau, z\langle \widetilde{y}\rangle, \overline{z}\langle \widetilde{y}\rangle, (\widetilde{x})z\langle \widetilde{w}\rangle, (\widetilde{x})\overline{z}\langle \widetilde{w}\rangle \mid \widetilde{x} \subseteq \widetilde{w}\}$ and let μ range over $\mathcal{M}$. The syntax, structural congruence and labelled transition system of Pi-F processes are shown in Figure 4. The syntax (Fig. 4(a)) is similar to the syntax of cc-pi processes but for the fact that in Pi-F input and output prefixes are polyadic, that summation, `tell`, `ask` are missing, and that explicit fusions $\widetilde{x} = \widetilde{y}$ replace constraints c. Despite the original presentation of the calculus, this syntax rules out processes containing name fusions under prefixes. This choice follows the analogous restriction applied in cc-pi, which avoids that two processes synchronise and, simultaneously, add some constraints to the store, thus possibly yielding an inconsistency. However, we can release this restriction in both calculi if we consider the fragment of cc-pi with explicit fusions rather than arbitrary constraints. To emphasise this syntactical analogy with cc-pi, we have chosen also in this case to distinguish between processes U and P. The structural axioms (Fig. 4(b)) are the same as in cc-pi but for the fact that the axioms dealing with constraints are replaced by the axioms for explicit fusions. The definition of equivalence relation $\text{Eq}(P)$ (Fig 4(c)) specifies the explicit fusions of a process P. We write $\varphi \cup \psi$ for the equivalence-closed union of the equivalence relations φ and ψ, $\varphi \setminus x$ for when x is a singleton class and all other names are related as in φ, and Id for the identity relation. In [15] several bisimulations for the Pi-F calculus are proposed, including a symbolic bisimulation, and they are all proved equivalent. For convenience, here we consider the *inside-outside* bisimulation that is the closest to open bisimulation.

Definition 3 (inside-outside bisimilarity). Inside-outside bisimilarity $(\sim^{io})$ is the largest symmetric relation S between Pi-F processes such that $P S Q$ implies:

- $\text{Eq}(P) = \text{Eq}(Q)$;
- If $P \xrightarrow{\mu}_f P'$ with $\text{bn}(\mu) \cap \text{fn}(Q) = \emptyset$ then $Q \xrightarrow{\mu}_f Q'$ and $P' S Q'$;
- $P \mid x = y \ S \ Q \mid x = y$, for all fusions $x = y$.

We give below a translation of Pi-F processes into cc-pi where we take the underlying named c-semiring to be the named c-semiring of Herbrand constraint systems with a signature including the operations for tupling, as shown in Example 1. Remark that in this case $x = \langle y_1, \ldots, y_n \rangle$ denotes the unification of the name x with the term $\langle y_1, \ldots, y_n \rangle$ and that a name equality $\widetilde{x} = \widetilde{y}$, with $\widetilde{x}$ and $\widetilde{y}$ of the same arity n, can be modelled as the constraint $z = \langle x_1, \ldots, x_n \rangle \otimes w = \langle y_1, \ldots, y_n \rangle \otimes z = w$. We abbreviate $\langle x_1, \ldots, x_n \rangle$ by $\widetilde{x}$.

Definition 4. *Let* $[\![\_]\!]$ *be the following translation of Pi-F agents:*

$$[\![\tau.U]\!] = \tau.[\![U]\!] \quad [\![\overline{x}\langle \widetilde{y}\rangle.U]\!] = (z)(\overline{x}\langle z\rangle.[\![U]\!] \mid z = \widetilde{y}) \quad [\![x\langle \widetilde{y}\rangle.U]\!] = (z)(x\langle z\rangle.[\![U]\!] \mid z = \widetilde{y})$$

$$[\![0]\!] = 0 \quad [\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!] \quad [\![(x)P]\!] = (x)[\![P]\!] \quad [\![I(\widetilde{y})]\!] = [\![[\widetilde{y}/\widetilde{x}]U]\!] \quad \text{if } I(\widetilde{x}) \stackrel{\text{def}}{=} U$$

Theorem 3. $P \sim^{io} Q$ *iff* $[\![P]\!] \sim^o [\![Q]\!]$.

Proof (Hint). By theorem 2 and by an analogous result proved in [15] for the Pi-F calculus, the theorem can be restated by replacing $\sim^o$ and $\sim^{io}$ with their respective symbolic versions. This fact greatly simplifies the proof. Another key point is that the instance of cc-pi that we consider includes more contexts that Pi-F, i.e. the contexts $\widetilde{x} = \widetilde{y}$, where the arity of $\widetilde{x}$ and $\widetilde{y}$ is non-null and it is the same for both tuples. In fact, the cc-pi processes that belong to the inverse image $[\![\_]\!]^{-1}$ of the translation cannot be

$$\pi ::= \tau \mid \overline{x}\langle \widetilde{y}\rangle \mid x\langle \widetilde{y}\rangle \quad U ::= \mathbf{0} \mid U|U \mid \pi.U \mid (x)U \mid I(\widetilde{y}) \quad P ::= U \mid \widetilde{x}=\widetilde{y} \mid P|P \mid (x)P$$

<div align="center">(a) syntax</div>

$$P|\mathbf{0} \equiv_F P \qquad P|Q \equiv_F Q|P \qquad (P|Q)|R \equiv_F P|(Q|R) \qquad (x)(y)P \equiv_F (y)(x)P$$

$$P|(x)Q \equiv_F (x)(P|Q) \quad \text{if } x \notin \text{fn}(P) \qquad\qquad I(\widetilde{y}) \equiv_F [\widetilde{y}/\widetilde{x}]U \quad \text{if } I(\widetilde{x}) \overset{\text{def}}{=} U$$

$$x=x \equiv_F \mathbf{0} \quad (x)(x=y) \equiv_F \mathbf{0} \qquad\qquad x=y \equiv_F y=x \qquad x=y|y=z \equiv_F x=z|y=z$$

$$x=y|x\langle \widetilde{z}\rangle.P \equiv_F x=y|y\langle \widetilde{z}\rangle.P \qquad\qquad w=y|x\langle \widetilde{z}\rangle.P \equiv_F w=y|x\langle \widetilde{z}\rangle[y/w].P \quad \text{if } w \in \widetilde{z}$$

$$x=y|\overline{x}\langle \widetilde{z}\rangle.P \equiv_F x=y|\overline{y}\langle \widetilde{z}\rangle.P \qquad\qquad w=y|\overline{x}\langle \widetilde{z}\rangle.P \equiv_F w=y|\overline{x}\langle \widetilde{z}\rangle[y/w].P \quad \text{if } w \in \widetilde{z}$$

<div align="center">(b) structural congruence</div>

$$\text{Eq}(\mathbf{0}) = \text{Id} \qquad \text{Eq}(x=y) = \{(x,y),(y,x)\} \cup \text{Id} \qquad \text{Eq}(\pi.U) = \text{Id}$$

$$\text{Eq}(P|Q) = \text{Eq}(P) \cup \text{Eq}(Q) \quad \text{Eq}((x)P) = \text{Eq}(P) \setminus x \quad \text{Eq}(I(\widetilde{y})) = \text{Eq}([\widetilde{y}/\widetilde{x}]U) \quad \text{if } I(\widetilde{x}) \overset{\text{def}}{=} U$$

<div align="center">(c) equivalence relation $\text{Eq}(P)$</div>

(PREF)
$$\pi.U \overset{\pi}{\to}_f U$$

(COMM)
$$\frac{P \overset{x\langle \widetilde{y}\rangle}{\to}_f P' \quad Q \overset{\overline{x}\langle \widetilde{w}\rangle}{\to}_f Q' \quad |\widetilde{y}| = |\widetilde{w}|}{P|Q \overset{\tau}{\to}_f P'|Q'|\widetilde{y}=\widetilde{w}}$$

(PAR)
$$\frac{P \overset{\mu}{\to}_f P' \quad \text{bn}(\mu) \cap \text{fn}(Q) = \emptyset}{P|Q \overset{\mu}{\to}_f P'|Q}$$

(RES)
$$\frac{P \overset{\mu}{\to}_f P' \quad x \notin \text{n}(\mu)}{(x)P \overset{\mu}{\to}_f (x)P'}$$

(OPEN-I)
$$\frac{P \overset{(\widetilde{w})z\langle \widetilde{y}\rangle}{\to}_f P' \quad (x,z) \notin \text{Eq}(P),\ x \in \widetilde{y} \setminus \widetilde{w}}{(x)P \overset{(x\widetilde{w})z\langle \widetilde{y}\rangle}{\to}_f P'}$$

(STRUCT)
$$\frac{P \equiv_F P' \overset{\mu}{\to}_f Q' \equiv_F Q}{P \overset{\mu}{\to}_f Q}$$

<div align="center">(d) operational semantics (omitting a rule (OPEN-O) for output)</div>

<div align="center">**Fig. 4.** Pi-F calculus</div>

discriminated by these contexts. To see this point, note that checking equivalence over the above contexts corresponds to taking the symbolic transitions $[\![P]\!]^{-1} \overset{x=y\otimes c}{\rightarrowtail} [\![Q]\!]^{-1}|c$ where $[\![P]\!]^{-1} \overset{x=y}{\rightarrowtail} [\![Q]\!]^{-1}$ also holds. The first kind of transition is not maximal and it can be discarded since there is another transition that is maximal.

7 Conclusions

In general our labelled transition system takes any name instantiation that leads to constraints not evaluating to the bottom element of the c-semiring and it does not provide a way to discard non-optimal solutions. We plan to generalise the notion of consistency to α-consistency, where α is a strictly non-negative threshold. Accordingly, we could study a variant of the present semantics in which, for instance, the consistency check

in the rules for placing constraints is substituted by an α-consistency check. We also intend to enrich our semantics in order to model non-deterministic timed behaviours and to compare it to paradigms such as timed concurrent constraint programming [10].

It would also be interesting to further explore the expressiveness of cc-pi by providing fully abstract encodings of other calculi, such as the applied pi-calculus and the pi-calculus. The main idea behind embedding applied pi-calculus would be to characterise a variant of the named c-semiring for Herbrand constraints that models a generic signature along with an equational theory. On the other side, translating the pi-calculus into cc-pi seems harder. The main challenge in translating the pi-calculus would be to express in terms of named constraints the concept of distinctions, which are used to define open bisimulation on the cc-pi calculus with restriction operator.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115. ACM Press, 2001.
2. S. Bistarelli and F. Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In *ECAI*, pages 63–67. IOS Press, 2006.
3. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
4. M. Boreale and R. De Nicola. A symbolic semantics for the pi-calculus. *Inf. Comput.*, 126(1):34–52, 1996.
5. M. G. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *Proceedings of ESOP 2007, 16th European Symposium on Programming*, volume 4421 of *Lect. Notes in Comput. Sci.*, pages 18–32. Springer Verlag, 2007.
6. J. F. Diaz, C. Rueda, and F. Valencia. A calculus for concurrent processes with constraints. *CLEI Electronic Journal*, 1(2), 1998.
7. D. Gilbert and C. Palamidessi. Concurrent constraint programming with process mobility. In *Computational Logic 2000*, volume 1861 of *Lect. Notes in Comput. Sci.*, pages 463–477. Springer Verlag, 2000.
8. M. Hennessy and H. Lin. Symbolic bisimulations. *Theor. Comp. Sci.*, 138:353–389, 1995.
9. R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.
10. C. Palamidessi and F. Valencia. A temporal concurrent constraint programming calculus. In *Proc. of CP 2001, 7th Int. Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lect. Notes in Comput. Sci.*, pages 302–316. Springer Verlag, 2001.
11. S. Rudeanu and D. Vaida. Semirings in operations research and computer science. *Fundam. Inf.*, 61(1):61–85, 2004.
12. D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Inform.*, 33:69–97, 1996.
13. V. Saraswat. *Concurrent Constraint Programming*. PhD thesis, Carnegie Mellon University, 1993.
14. V. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. POPL'90*. ACM Press, 1990.
15. L. Wischik and P. Gardner. Strong bisimulation for the explicit fusion calculus. In *Proceedings of FOSSACS 2007, Foundations of Software Science and Computational Structures*, volume 2987 of *Lect. Notes in Comput. Sci.*, pages 484–498. Springer Verlag, 2004.
16. L. Wischik and P. Gardner. Explicit fusions. *Theoret. Comput. Sci.*, 340(3):606–630, 2005.