TLP **11** (4–5): 469–485, 2011. © Cambridge University Press 2011
doi:10.1017/S1471068411000123

*Observational equivalences for linear logic concurrent constraint languages**

RÉMY HAEMMERLÉ
Technical University of Madrid, Madrid, Spain

Abstract

Linear logic Concurrent Constraint programming (LCC) is an extension of concurrent constraint programming (CC), where the constraint system is based on Girard's linear logic instead of the classical logic. In this paper, we address the problem of program equivalence for this programming framework. For this purpose, we present a structural operational semantics for LCC based on a label transition system and investigate different notions of observational equivalences inspired by the state of art of process algebras. Then, we demonstrate that the asynchronous π-calculus can be viewed as simple syntactical restrictions of LCC. Finally, we show that LCC observational equivalences can be transposed straightforwardly to classical Concurrent Constraint languages and Constraint Handling Rules, and investigate the resulting equivalences.

KEYWORDS: Concurrent Constraint programming, linear logic, observational equivalences, bisimulation, π-calculus, Constraint Handling Rules

1 Introduction

The class of Concurrent Constraint (CC) languages (Saraswat and Rinard 1990) was introduced as a generalization of concurrent logic programming (Maher 1987) with constraint logic programming (Jaffar and Lassez 1987). Nonetheless, it has strong similarities with more classical models of concurrency such as the Calculus of Communicating Systems (CCS), the Chemical Abstract Machine (CHAM), or the π-calculus. For example, its semantics has been originally expressed by process algebras similar to CSS (Saraswat and Rinard 1990) or later in the style of the CHAM (Fages *et al.* 2001). Furthermore, it generalizes Actor model (Kahn and Saraswat 1990) and possesses the phenomenon of channel mobility of the π-calculus (Laneve and Montanari 1992).

Nonetheless, any CC language differs from the usual models of concurrency because it relies on a constraint system for specifying relationship (entailment) between messages (constraints), which confers to it a "monotonic" essence. Indeed, in CC, processes can only add information by posting constraints or checking that enough information is available to entail a *guard*. *Linear logic CC* languages

* A version of the paper including the proofs is available as technical report (Haemmerlé 2011).

(LCC) (Saraswat and Lincoln 1992) have been introduced as a generalization of CC, in which processes can consume information by means of the ask operation, hence, breaking the monotonicity of CC. The main idea of this extension is to view the constraint system as Girard's linear logic (Girard 1987) theory instead of classical logic theory. It results in a simple framework that unifies constraint programming and asynchronous process algebras.

Since the beginning of the nineties, the semantics foundation of LCC has been well studied (see for instance Best *et al.* (1997); Ruet and Fages (1997); Fages *et al.* (2001); Hammerlé *et al.* (2007)), but surprisingly, the formal comparison with classical models of concurrency has received little attention. Indeed, during the same period, the use of constraints in the context of concurrency seems to have received more than a little attention. For instance, the fusion calculus (Parrow and Victor 1998) introduced at the end of the nineties can be viewed as a generalization of the π-calculus with unification constraints. Several hybrid process algebras with constraint mechanisms have also been proposed (see for example Díaz *et al.* (1998); Glibert and Palamidessi (2000); Buscemi and Montanari (2007)).

In this paper, we investigate observational equivalence for LCC. Here, we understand observational equivalence in a broad sense: two processes are observationally equivalent if, in any environment, an external observer cannot possibly tell the difference when one process is unplugged and the other one plugged in. In order to provide a relevant instantiation for this intuitive definition, it is necessary to take into account the execution paradigm, in which the processes will be considered. Indeed, in CC frameworks, there typically exist two possible execution paradigms: the "backtracking" paradigm (from logic programs), which allows reversible executions, and the "committed choice" paradigm (from process algebras), which does not. In the following, we propose the *may-testing equivalence* and the *barbed congruence* as natural instances of observable equivalence for LCC when considered in these respective paradigms. We also propose the *logical equivalence* and the *labelled bisimulation* that will provides simpler characterization for the two former notions.

In order to define such equivalences, we will look at LCC from a point of view slightly different from the classical one: here constraints are not posted into a central blackboard anymore, but they are processes that can migrate, merge, and emit as message a part of the information they represent; meanwhile, ask processes just wait for messages that "logically" match their guards. Hence, it is possible to express the operational semantics of LCC by an elegant labeled transition system. We then show that the asynchronous π-calculus can be viewed as a subcalculus of LCC, and that the usual π-calculus observational equivalences are particular instances of the ones of LCC. Finally, we investigate particular properties of LCC observational equivalences when they are transposed into classical CC and Constraint Handling Rules (CHR).

2 A process calculi semantics for LCC

In this paper, we assume given a denumerable set $\mathscr{V}$ of variables, a denumerable set Σ_c of predicate symbols (denoted by γ), and a denumerable set Σ_f of function and constant symbols. First-order terms built from $\mathscr{V}$ and Σ_f will be denoted by t.

Sequences of variables or terms will be denoted by bold face letters such as $\mathbf{x}$ or $\mathbf{t}$. For an arbitrary formula A, $\mathrm{fv}(A)$ denotes the set of free variables occurring in A, and $A[\mathbf{x}\backslash\mathbf{t}]$ represents A, in which the occurrences of variables $\mathbf{x}$ have been replaced by terms $\mathbf{t}$ (with the usual renaming of bound variables, avoiding variable clashes).

2.1 Syntax

In this section, we give a presentation of LCC languages, where declarations are replaced by replication of guarded processes. Indeed, replicated asks generalize usual declarations to closures with environment represented by the free variables in the ask (Haemmerlé *et al.* 2007). In LCC, we distinguish four syntactical categories, as specified by the following grammar:

$$\begin{array}{ll} c ::= \mathbf{1} \mid \mathbf{0} \mid \gamma(\mathbf{t}) \mid c \otimes c \mid \exists x.c \mid\ !c & (\textit{constraints}) \\ \alpha ::= \tau \mid c \mid (\mathbf{x})\overline{c} & (\textit{LCC-actions}) \\ G ::= \forall \mathbf{x}(c \rightarrow P) \mid G + G & (\textit{LCC-guards}) \\ P ::= \overline{c} \mid P|P \mid \exists x P \mid\ !G \mid G & (\textit{LCC-processes}) \end{array}$$

Constraints are formulas built from terms, constraint symbols, and the logical operators: $\mathbf{1}$ (true), $\mathbf{0}$ (false), the conjunction ($\otimes$), the existential quantifier ($\exists$), and the modality ($!$). The three kinds of actions are the *silent action* τ, the *input action* c, which represents a constraint for which a process waits, and the *output action* $(\mathbf{x})\overline{c}$ ($\mathbf{x}$ being the variables *extruded* by the action), which represents the constraint posted by a process. The order of the extruded variables in an output message is irrelevant; hence, if $\mathbf{y}$ is a permutation of the sequence $\mathbf{x}$, we will consider $(\mathbf{x})\overline{c}$ equal to $(\mathbf{y})\overline{c}$. In LCC processes, an overlined constraint $\overline{c}$ stands for *asynchronous tell*, $|$ for *parallel composition*, $\exists$ for *variable hiding*, $\rightarrow$ for *blocking ask*, $+$ for guarded choice, and $!$ for *replication*. As one can see, the syntax for LCC processes does not include specific construction for the null process. Indeed, this latter can be emulated by the trivial constraint $\overline{\mathbf{1}}$, which represents no information.

For convenience, if $\mathbf{x}$ is empty, we will abbreviate $\forall \mathbf{x}(c \rightarrow P)$ and $(\mathbf{x})\overline{c}$ as $c \rightarrow P$ and $\overline{c}$, respectively. $\exists \mathbf{x} A$ will be a notation for $\exists x_1 \ldots \exists x_n A$ if A is a constraint or an LCC process, and $\mathbf{x}$ is the sequence of variables $x_1, \ldots, x_n$. Moreover, for any finite multiset of processes $\{P_1, \ldots, P_n\}$, we will use $\Pi_{i=1}^{n} P_i$ as abbreviations for $P_1|\cdots|P_n$. As usual, the existential and universal quantifiers in constraints and LCC processes are considered as variable binders. Conventionally, we consider the variables $\mathbf{x}$ as free in any action of the form $(\mathbf{x})\overline{c}$. We use $\mathrm{ev}(\alpha)$ as an abbreviation for the extruded variables of α (i.e., $\mathrm{ev}(\alpha) = \mathbf{x}$, if α is an action of the form $(\mathbf{x})\overline{c}$, $\mathrm{ev}(\alpha) = \emptyset$ otherwise).

LCC languages are parametrized by *a (linear) constraint system*, which is a pair $(\mathscr{C}, \Vdash_{\mathscr{C}})$, where $\mathscr{C}$ is the set of all constraints and $\Vdash_{\mathscr{C}}$ is a subset of $\mathscr{C} \times \mathscr{C}$, which defines the nonlogical axioms of the system. For a given constraint system $(\mathscr{C}, \Vdash_{\mathscr{C}})$, the entailment relation $\vdash_{\mathscr{C}}$ is the smallest relation containing $\Vdash_{\mathscr{C}}$ and closed by the rules of intuitionistic linear logic. We will use the notation $A \dashv\vdash_{\mathscr{C}} B$ to mean that both sequents $A \vdash_{\mathscr{C}} B$ and $B \vdash_{\mathscr{C}} A$ hold.

In this paper, we are interested in studying classes of LCC processes obtained by syntactical restrictions on the constraints that they can use. These restrictions will simulate the power of the observer in LCC subcalculi and/or the visibility limitations

Table 1. *Labeled transition system for LCC*

$$\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q}\ (cong) \qquad \frac{P|G \xrightarrow{\alpha} Q}{P|(G+G') \xrightarrow{\alpha} Q}\ (sum)$$

$$\frac{P \xrightarrow{\alpha}_{\mathscr{C}} P' \quad \mathrm{ev}(\alpha) \cap \mathrm{fv}(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_{\mathscr{C}} P'|Q}\ (\mathscr{C}\text{-}comp) \qquad \frac{P \xrightarrow{\alpha}_{\mathscr{C}} Q \quad y \notin \mathrm{fv}(\alpha)}{\exists yP \xrightarrow{\alpha}_{\mathscr{C}} \exists yQ}\ (\mathscr{C}\text{-}rest)$$

$$\frac{\begin{array}{c} c \vdash_{\mathscr{C}} \exists \mathbf{x}(d \otimes e) \quad \exists \mathbf{x} d \vdash_{\mathscr{C}} \exists \mathbf{x}' d' \quad \mathbf{x}\mathbf{x}' \cap \mathrm{fv}(c) = \emptyset \\ c' \vdash_{\mathscr{C}} \exists \mathbf{x}(d' \otimes e) \text{ is a most general choice} \end{array}}{\overline{c} \xrightarrow{(\mathbf{x}')\overline{d'}}_{\mathscr{C}} \overline{e}}\ (\mathscr{C}\text{-}out) \qquad \frac{P \xrightarrow{(\mathbf{x})\overline{c}}_{\mathscr{C}} Q}{\exists yP \xrightarrow{(y\mathbf{x})\overline{c}}_{\mathscr{C}} Q}\ (\mathscr{C}\text{-}ext)$$

$$\frac{\begin{array}{c} c \vdash_{\mathscr{C}} \exists \mathbf{y}(d[\mathbf{x}\backslash\mathbf{t}] \otimes e) \quad \mathbf{y} \cap \mathrm{fv}(c, d, A) = \emptyset \\ \exists \mathbf{y}(d[\mathbf{x}\backslash\mathbf{t}] \otimes e) \text{ is a most general choice} \end{array}}{\overline{c}|\forall \mathbf{x}(d \to A) \xrightarrow{\tau}_{\mathscr{C}} \exists \mathbf{y}.(A[\mathbf{x}\backslash\mathbf{t}]|\overline{e})}\ (\mathscr{C}\text{-}sync) \qquad \overline{\mathbf{1}} \xrightarrow{c}_{\mathscr{C}} \overline{c}\ (\mathscr{C}\text{-}in)$$

imposed by *ad hoc* scope mechanisms such as module systems. In practice, they will be specified by means of two subsets of $\mathscr{C}$ that will limit the possible constraints a process can respectively ask or tell. Formally, for all subsets $\mathscr{D}$ and $\mathscr{E}$, we say that a process P is $\mathscr{D}$*-ask restricted* (respectively, $\mathscr{E}$*-tell restricted*) if it is obtained by the grammar for processes, where any ask $\forall \mathbf{x}(c \to P)$ (respectively, any tell $\overline{c}$) satisfies $(\exists \mathbf{x}.c) \in \mathscr{D}$ (respectively, $c \in \mathscr{E}$). More generally, we say that P is a $\mathscr{D}\mathscr{E}$-process if P is both $\mathscr{D}$-ask and $\mathscr{E}$-tell restricted.

2.2 Operational semantics

In Table 1, we define, for a given constraint system $(\mathscr{C}, \Vdash_{\mathscr{C}})$, the operational semantics of LCC by means of a labeled transition system. As usual, in process algebras, this semantics uses a structural congruence. This congruence, noted $\equiv_{\mathscr{C}}$, is defined as the smallest equivalence satisfying α-renaming of bound variables, commutativity, and associativity for parallel composition, summation, and the following identities:

$$P|\overline{\mathbf{1}} \equiv_{\mathscr{C}} P \qquad \exists z \overline{\mathbf{1}} \equiv_{\mathscr{C}} \overline{\mathbf{1}} \qquad \exists x \exists y P \equiv_{\mathscr{C}} \exists y \exists x P \qquad !P \equiv_{\mathscr{C}} P|!P$$

$$\frac{c \otimes d \dashv\vdash_{\mathscr{C}} e}{\overline{c}|\overline{d} \equiv_{\mathscr{C}} \overline{e}} \qquad \frac{P \equiv_{\mathscr{C}} P'}{P|Q \equiv_{\mathscr{C}} P'|Q} \qquad \frac{z \notin \mathrm{fv}(P)}{P|\exists zQ \equiv_{\mathscr{C}} \exists z(P|Q)} \qquad \frac{P \equiv_{\mathscr{C}} P'}{\exists x.P \equiv_{\mathscr{C}} \exists x.P'}$$

The side condition "$c \vdash_{\mathscr{C}} \exists \mathbf{y}(d[\mathbf{x}\backslash\mathbf{t}] \otimes e)$ is a most general choice" is a reasonable restriction, which guarantees that the transition does not weaken constraints within a process, as can do the logical entailment (for instance, we want to avoid entailment such as $!c \vdash_{\mathscr{C}} c \otimes \mathbf{1}$). It can be defined as: for any constraint e', all terms $\mathbf{t}'$ and all variables $\mathbf{y}'$ if $c \vdash_{\mathscr{C}} \exists \mathbf{y}'(d[\mathbf{x}\backslash\mathbf{t}'] \otimes e')$ and $\exists \mathbf{y}' e' \vdash_{\mathscr{C}} \exists \mathbf{y} e$ hold, then so do $\exists \mathbf{y} d[\mathbf{x}\backslash\mathbf{t}] \vdash_{\mathscr{C}} \exists \mathbf{y}' d[\mathbf{x}\backslash\mathbf{t}']$ and $\exists \mathbf{y} e \vdash_{\mathscr{C}} \exists \mathbf{y}' e'$. In the constraint systems, we will consider in this paper, such a deduction is always possible.

The notion of *weak transition* is defined classically:

$$(P \overset{\tau}{\Rightarrow}_{\mathscr{C}} Q) \overset{def}{\iff} (P \xrightarrow{\tau}{}^{*}_{\mathscr{C}} Q) \qquad (P \overset{\alpha}{\Rightarrow}_{\mathscr{C}} Q) \overset{def}{\iff} (P \xrightarrow{\tau}{}^{*}_{\mathscr{C}} \xrightarrow{\alpha}_{\mathscr{C}} \xrightarrow{\tau}{}^{*}_{\mathscr{C}} Q) \quad (\text{for } \alpha \neq \tau)$$

In the asynchronous context of this paper, it seems natural to restrict the observation to outputs. As argued by Amadio *et al.* (1998), the intuition is that an observer cannot know that a message he has sent has been actually received. Moreover, since an observer has no way of knowing if the execution of a particular process is terminated unless he receives a programmed acknowledgment, we will disregard classical (L)CC observables, which deal with termination such as success stores (Saraswat *et al.* 1991; Fages *et al.* 2001), and consider only accessible constraints (Haemmerlé *et al.* 2007). Formally, for any set $\mathscr{D} \subset \mathscr{C}$, the set of *$\mathscr{D}$-accessible constraints* for a process P is defined as follows:

$$\mathscr{O}^{\mathscr{D}}(P) = \left\{ (\exists \mathbf{x}.c) \in \mathscr{D} \mid \text{ there exists } P' \text{ such that } P \stackrel{\tau}{\Rightarrow}_{\mathscr{C}} \exists \mathbf{x}.(P'|c) \right\}$$

The semantics we propose has important links with the one defined by Best *et al.* (1997) but it is in some important aspects more general. In particular, the language we consider provides replication and explicit operators for both universal and existential quantifications, all of which are important features. Indeed, on the one hand, replication and existential quantification are crucial to internalize declarations and closures in processes (Haemmerlé *et al.* 2007); while, on the other hand, universal quantification cannot be emulated by tell processes in every constraint system, especially linear ones (Fages *et al.* 2001). Another difference is that our system uses the *asynchronous input* rule as initially proposed by Honda and Yoshida (1995) for the π-calculus. This rule, which allows an observer to do any input action at any time, is not designed to be observed directly but rather to simplify bisimulation-based definitions within asynchronous frameworks (Amadio *et al.* 1998).

Example 2.1 (*Dining philosophers*)
As suggested by Best *et al.* (1997), the dining philosophers problem has an extremely simple solution in LCC. Here is an adaptation of the solution proposed by Ruet and Fages (1997). The atomic constraints are $\mathtt{frk}(i)$ and $\mathtt{eat}(j)$ for $i, j \in \mathbb{N}$, and $\vdash_{\mathscr{C}}$ is the trivial entailment relation. Assuming the following encoding for the ith philosopher among n, a solution for the problem consists of the process $\Pi_{i=0}^{n-1} \left(P_i^n | \overline{\mathtt{frk}(i)}\right)$.

$$P_i^n = !\left(\mathtt{frk}(i) \otimes \mathtt{frk}(i{+}1 \bmod n) \rightarrow \left(\overline{\mathtt{eat}(i)} | \mathtt{eat}(i) \rightarrow \left(\overline{\mathtt{frk}(i)} | \overline{\mathtt{frk}(i{+}1)}\right)\right)\right)$$

This solution suffers neither deadlock nor starvation problems: the system can always advance to a different state, and at least one philosopher will eventually eat.

2.3 Logical semantics

In this section, we show that the results of logical semantics from LCC (Fages *et al.* 2001; Haemmerlé *et al.* 2007) can be shifted to the version of LCC we propose in this paper. It will provide us with a powerful tool to reason about processes. It is worth noting that the logical semantics proposed here is slightly different from the usual one since it uses an additional conjunction with $\top$. As shown by the next theorem, this modification is harmless when regarding accessible constraints, but yields a more relevant notion of equivalence. (Refer to the discussion in Section 3.1.) Note that the conjunction with $\top$ is not necessary in case of translation of a parallel composition

and hiding since it commutes with $\otimes$ and $\exists$ (i.e., $(A \otimes \top) \otimes (B \otimes \top) \dashv\vdash_{\mathscr{C}} A \otimes B \otimes \top$ and $\exists x(A \otimes \top) \dashv\vdash_{\mathscr{C}} \exists x(A) \otimes \top$).

Definition 2.2
Processes are translated into linear logic formulas as follows:

$$\begin{array}{rclrclrcl} \bar{c}^{\dagger} & = & c \otimes \top & (P|Q)^{\dagger} & = & P^{\dagger} \otimes Q^{\dagger} & (P+Q)^{\dagger} & = & (P^{\dagger} \,\&\, Q^{\dagger}) \otimes \top \\ (!P)^{\dagger} & = & !(P^{\dagger}) \otimes \top & (\exists x P)^{\dagger} & = & \exists x P^{\dagger} & (\forall \mathbf{x}(c \to P))^{\dagger} & = & \forall \mathbf{x}(c \multimap P^{\dagger}) \otimes \top \end{array}$$

Theorem 2.3 (*Logical semantics*)
For any process P and any set $\mathscr{D}$ of linear constraints, $\mathscr{O}^{\mathscr{D}}(P) = \left\{ d \in \mathscr{D} \mid P^{\dagger} \vdash_{\mathscr{C}} d^{\dagger} \right\}$.

3 Observational equivalence relations for LCC

In this section, we propose some equivalence relations for LCC processes.

An important property of processes related by equivalences is their dependence on the environment. More precisely, two equivalent processes must be indistinguishable by an observer in any context (i.e., equivalences must be congruences). Formal *contexts*, written $C[\,]$, are processes with a special constant $[\,]$, the hole. Putting a term P into the holes of a context $C[\,]$ gives the term noted $C[P]$. In practice, we define all our congruences for *evaluation contexts* (Fournet and Gonthier 2005), a particular class of contexts , where the hole occurs exactly once and not under a guard nor a replication. These contexts, also called static contexts (Milner 1989), describe environments that can communicate with an observed process and filter its messages but can neither substitute variables of the process nor replicate it. In this paper, without explicit statement of the contrary, all congruence properties will refer to these contexts only. In particular, we will use the terminology "*full congruence*" to refer to the congruence with respect to arbitrary contexts. In the framework of LCC, $\mathscr{DE}$*-contexts* and $\mathscr{DE}$*-congruence* will refer to evaluation contexts and congruence built from $\mathscr{DE}$-processes.

3.1 Logical equivalence

Strictly speaking, the first notion of equivalence we consider is not observational, but stems naturally from the logical semantics of the language. Indeed, the logical semantics ensures that processes with logically equivalent translations have the same accessible constraints. This notion of equivalence is specially interesting since it can be proved using automated theorem provers such as llprover (Tamura 1998).

Definition 3.1 (*Logical equivalence*)
The *(weak) logical equivalence* on LCC-processes is defined as follows:

$$P \circ\!\!-\!\!\circ_{\mathscr{C}} Q \overset{def}{\iff} P^{\dagger} \dashv\vdash_{\mathscr{C}} Q^{\dagger}$$

We call this equivalence "weak" because it is strictly less discriminating than the one we would obtain using usual logical semantics of LCC. Nonetheless, the present definition is more relevant since it does not distinguish Girard's exponential connective, noted ! in Linear logic, from Milner's replication, noted also ! in process

algebras. Indeed, for any linear logic formula A, $!A \otimes A \otimes \top \dashv\vdash !A \otimes \top$ holds, whereas $!A \otimes A \dashv\vdash !A$ does not. The proposition we give next states that the use of $\top$ does not break the congruence property of logical equivalence.

Proposition 3.2
Weak logical equivalence is a full congruence.

3.2 May-testing equivalence

The following equivalence relates to testing semantics (Nicola and Hennessy 1984). We argue that this relation provides a canonical notion of observational equivalence for LCC if considered within the "backtracking" execution paradigm. Indeed, it is defined as the largest congruence that respects accessible constraints. For the sake of generality, we defined may testing in a parametric way according to input/output filters.

Definition 3.3 (*May-testing equivalence*)
Let $\mathscr{D}$ and $\mathscr{E}$ be two subsets of $\mathscr{C}$. The *may* $\mathscr{D}\mathscr{E}$*-testing* $\simeq_{\mathscr{D}\mathscr{E}}$ is the largest $\mathscr{D}\mathscr{E}$-congruence that respects $\mathscr{D}$-accessible constraints, formally:

$$P \simeq_{\mathscr{D}\mathscr{E}} Q \overset{def}{\iff} \text{for any evaluation } \mathscr{D}\mathscr{E}\text{-context } C[\,],\ \mathscr{O}^{\mathscr{D}}(C[P]) = \mathscr{O}^{\mathscr{D}}(C[Q]).$$

Quite naturally, logical equivalence implies any may-testing equivalence relation. One can use logical semantics and Proposition 3.2 to demonstrate it. It is worth noting that the inclusion is strict. For instance, the processes $c \to \exists x.P$ and $\exists x.(c \to P)$, where x is free in P and not in c, are clearly equivalent with respect to any may-testing equivalence but are not logically equivalent in linear logic.

Example 3.4
Contrary to the processes in Example 2.1, the following implementation for the ith dining philosopher does not use atomic consumptions of constraint conjunctions:

$$Q_i^n = \ !\big(\mathtt{frk}(i) \to \big(\mathtt{frk}(i{+}1 \bmod n) \to \big(\overline{\mathtt{eat}(i)} \,|\, \mathtt{eat}(i) \to \big(\overline{\mathtt{frk}(i)} \,|\, \overline{\mathtt{frk}(i{+}1 \bmod n)}\big)\big)\big)\big)$$

Although the solutions built with such philosophers face deadlock and starvation problems, the two implementations of philosopher cannot be distinguished by may testing (i.e., for all $i,n \in \mathbb{N}$, $P_i^n \simeq_{\mathscr{C}\mathscr{C}} Q_i^n$). Note that in the "backtracking" execution paradigm, there is no reason to distinguish such processes. Indeed, the possibility of reversing executions makes deadlocks invisible from an external point of view.

3.3 Labeled bisimulation

In the framework of process algebra, bisimulation-based equivalence relations are the most commonly used notion of equivalence. Contrary to the may-testing equivalences and the barbed congruences presented in the following, the labeled bisimulation proofs do not require explicit context closure. Indeed, as shown in Theorem 3.6, congruence is not a requirement, but a derived property. Hence, the proofs can be established by coinduction, by considering only few steps. As we have done for may testing, our definition of bisimulation is parametrized by input/output filters.

Definition 3.5 (*Labeled bisimulation*)
Let $\mathscr{D}$ and $\mathscr{E}$ be two subsets of $\mathscr{C}$. An action is *$\mathscr{D}\mathscr{E}$-relevant* for a process Q if it is either a silent action, or an input action in $\mathscr{E}$, or an output action of the form $(\mathbf{x})\overline{c}$ with $(\exists \mathbf{x}.c) \in \mathscr{D}$ and $\mathbf{x} \cap \mathrm{fv}(Q) = \emptyset$. A symmetrical relation $\mathscr{R}$ is a *$\mathscr{D}\mathscr{E}$-bisimulation* if for all P, P', Q, α such that $P\mathscr{R}Q$, $P \xrightarrow{\alpha}_{\mathscr{C}} P'$, and α is $\mathscr{D}\mathscr{E}$-relevant for Q, there exists Q' such that $Q \overset{\alpha}{\Rightarrow}_{\mathscr{C}} Q'$ and $P'\mathscr{R}Q'$. The largest $\mathscr{D}\mathscr{E}$-bisimulation is called *$\mathscr{D}\mathscr{E}$-bisimilarity* and is denoted with $\approx_{\mathscr{D}\mathscr{E}}$.

Theorem 3.6
For all sets of constraints $\mathscr{D}$ and $\mathscr{E}$, the $\mathscr{D}\mathscr{E}$-bisimilarity is a $\mathscr{D}\mathscr{E}$-congruence.

3.4 Barbed congruence

Barbed bisimulation has been introduced by Milner and Sangiorgi (1992) as a uniform way to describe bisimulation-based equivalences for any calculus. From the definition of observables we give in Section 2.2, we derive a notion of barbed bisimulation in the standard way. As with many other barbed bisimulations, the obtained equivalence is too rough. For example, no barbed bisimulation distinguishes between processes $\overline{\mathbf{1}}$ and $c \to P$ (with $\not\vdash_{\mathscr{C}} c$), which exhibit clearly different behaviors when they are put in parallel with a constraint stronger than c. For this reason, we refine our bisimulation by enforcing congruence property following Fournet and Gonthier (2005). The resulting relation yields an instance of the intuitive notion of observational equivalence for LCC considered within the "committed-choice" paradigm.

Definition 3.7 (*Barbed congruence*)
Let $\mathscr{D}$ and $\mathscr{E}$ be two subsets of $\mathscr{C}$. A symmetrical relation $\mathscr{R}$ is a *$\mathscr{D}\mathscr{E}$-barbed bisimulation* if for all P, P', Q such that $P\mathscr{R}Q$, and $P \xrightarrow{\tau}_{\mathscr{C}} P'$, then there exists Q' such that $\mathscr{O}^{\mathscr{D}}(P) \subseteq \mathscr{O}^{\mathscr{D}}(Q)$, $Q \overset{\tau}{\Rightarrow}_{\mathscr{C}} Q'$ and $P'\mathscr{R}Q'$. The *barbed $\mathscr{D}\mathscr{E}$-congruence*, written $\cong_{\mathscr{D}\mathscr{E}}$, is the largest $\mathscr{D}\mathscr{E}$-congruence that is a $\mathscr{D}\mathscr{E}$-barbed bisimulation.

Clearly, barbed $\mathscr{D}\mathscr{E}$-congruence is more precise than may $\mathscr{D}\mathscr{E}$-testing equivalence. It is worth noting that it is in general strictly distinct from logical equivalence. For instance, $c \to \exists x.P$ and $\exists x.(c \to P)$ are $\mathscr{C}\mathscr{C}$-barbed congruent but not logically equivalent, while $c \to d \to \mathbf{1}$ and $c \otimes d \to \mathbf{1}$ are logically equivalent but not barbed congruent. In general, direct proofs of barbed congruence are tedious since they require explicit context closure. Fortunately, the barbed congruence coincides with labeled bisimulation. Barbed congruence can therefore be established by simpler proofs based on the coinductive principle of labeled bisimulation.

Theorem 3.8
For all sets of constraints $\mathscr{D}$ and $\mathscr{E}$, $\cong_{\mathscr{D}\mathscr{E}}$ and $\approx_{\mathscr{D}\mathscr{E}}$ coincide.

Example 3.9
The encoding of philosophers proposed in the two previous examples cannot be distinguished by may testing. Nonetheless, their behavior can be separated by barbed congruence. For instance, one can disprove $P_1^3 \cong_{\mathscr{C}\mathscr{C}} Q_1^3$. The following implementation refines the one of Example 3.4 by allowing a philosopher to put back the first fork he takes:

$$R_i^n = !\big(\texttt{frk}(i) \to \big(\overline{\texttt{frk}(i)} + \texttt{frk}(i+1 \bmod n) \to \big(\overline{\texttt{eat}(i)} | \texttt{eat}(i) \to \big(\overline{\texttt{frk}(i)} | \overline{\texttt{frk}(i+1 \bmod n)}\big)\big)\big)\big)$$

Although, solutions built with this latter implementation of philosophers still face starvation problems, the external behavior of these philosophers cannot be distinguished anymore from the ones of Example 2.1, i.e., $P_i^n \cong_{\mathscr{CC}} R_i^n$ for any $i, n \in \mathbb{N}$.

4 LCC: a natural generalization of asynchronous calculi

In this section, we show that LCC language generalizes asynchronous π-calculus. The asynchronous π-calculus is a variant of the π-calculus, where the emission is nonblocking. In practice, it is obtained by a simple syntactical restriction prohibiting output prefixing.

We briefly recall the syntax of the asynchronous π-calculus. Our notations and definitions are mostly standard. For convenience, we will use a denumerable subset of LCC variables as channel names. In this language, three syntactical categories are distinguished, as specified by the following grammar:

$$\begin{array}{ll} \alpha ::= \tau \mid \bar{x}y \mid \bar{x}(y) \mid (y)x(y) & (\pi\textit{-actions}) \\ G ::= \tau.P \mid x(y).P \mid !P & (\pi\textit{-guards}) \\ P ::= \mathbf{0} \mid \bar{x}y \mid P|P \mid \nu x P \mid G & (\pi\textit{-processes}) \end{array}$$

A π-calculus process (or π-process for short) is one of the following: the *null process* $\mathbf{0}$, the silent prefix $\tau.P$, the *message reception* $x(y).P$, the *asynchronous emission* $\bar{x}y$, the parallel composition of processes $P|Q$, the replication of processes $!P$, or the *scope restriction* $\nu y P$.

In this section, we assume the notion of reduction, which we write $\longrightarrow_\pi$, the may-testing equivalence, which we write $\simeq_\pi$, the labeled bisimulation, which we write $\approx_\pi$, and the barbed congruence, which we write $\cong_\pi$, as defined by Fournet and Gonthier (2005). We propose now a very simple interpretation of the asynchronous π-calculus into LCC following the preliminary ideas of Soliman (2003).

Definition 4.1 (*LCC interpretation of the asynchronous π-calculus*)
Let $\mathscr{C}_\pi$ be the trivial constraint system (i.e., a constraint system without nonlogical axioms), based on the predicate alphabet $\Sigma_c = \{\gamma\}$. The LCC interpretation $[\![\ ]\!]_\pi$ of π-actions and π-processes is defined recursively as follows:

$$\begin{array}{llll} [\![\tau]\!]_\pi = \tau & [\![xy]\!]_\pi = \gamma(x,y) & [\![\bar{x}(y)]\!]_\pi = \overline{\gamma(x,y)} & [\![(y)\bar{x}(y)]\!]_\pi = (y)\overline{\gamma(x,y)} \\ [\![\mathbf{0}]\!]_\pi = \overline{\mathbf{1}} & [\![\bar{x}z]\!]_\pi = \overline{\gamma(x,z)} & [\![\tau.P]\!]_\pi = \mathbf{1} \to [\![P]\!]_\pi & [\![x(y).P]\!]_\pi = \forall y(\gamma(x,y) \to [\![P]\!]_\pi) \\ [\![!P]\!]_\pi = ![\![P]\!]_\pi & [\![\nu x P]\!]_\pi = \exists x [\![P]\!]_\pi & [\![P|Q]\!]_\pi = [\![P]\!]_\pi | [\![Q]\!]_\pi & \end{array}$$

It can be noted that this mapping is completely compositional and does not need fresh names. Furthermore, the replacement of declarations by replicated asks leads to a translation, where each construct of the π-calculus is mapped to a unique construct of LCC. In fact, we can consider that this interpretation enforces a syntactical restriction on LCC processes, by allowing synchronization only on constraints of the form $\exists y.\gamma(x,y)$. Formally, assuming $\mathscr{D}_\pi = \{\mathbf{1}\} \cup \{\exists y.\gamma(x,y) \mid xy \in \mathscr{V} \wedge x \neq y\}$

and $\mathscr{E}_\pi = \{\mathbf{1}\} \cup \{\gamma(x,y) \mid xy \in \mathscr{V}\}$, the codomain of $[\![\ ]\!]_\pi$ is precisely the set of $\mathscr{D}_\pi\mathscr{E}_\pi$-processes. Furthermore, the following results ensure that there is a one-to-one correspondence between transitions of the two formalisms.

Theorem 4.2
$P \xrightarrow{\tau}_\pi Q$ if and only if $[\![P]\!]_\pi \xrightarrow{\tau}_{\mathscr{C}_\pi} [\![Q]\!]_\pi$.

The theorem and the simplicity of the interpretation emphasize that the π-calculus is syntactically and semantically a subcalculus of LCC. The only transition of LCC that is not captured by the π-calculus semantics is the simultaneous emission of messages (i.e., a constraint of the form $\gamma(x_1,y_1) \otimes \cdots \otimes \gamma(x_n,y_n)$). We argue that observing simultaneous emission is not relevant in asynchronous context, where the observer has no way of knowing the order in which the messages have been emitted. In fact, the LCC constraint system makes messages behave similarly to molecules within the CHAM (i.e., messages can combine by "cooling" and dissociate by "heating" (Berry and Boudol 1992)).

The following theorem states that may-testing equivalence, labeled bisimilarity, and barbed congruence are instances of equivalence relations we defined for LCC.

Theorem 4.3
Let $\mathscr{D}_\pi = \{\exists y.\gamma(x,y) \mid x \in \mathscr{V} \setminus \{y\}\}$ and $\mathscr{D}_\pi^\star = \mathscr{D}_\pi \cup \{\gamma(x,y) \mid xy \in \mathscr{V}\}$. For all π-processes P and Q, we have:

(i) $P \simeq_\pi Q$ if and only if $[\![P]\!]_\pi \simeq_{\mathscr{D}_\pi\mathscr{C}_\pi} [\![Q]\!]_\pi$.
(ii) $P \approx_\pi Q$ if and only if $[\![P]\!]_\pi \approx_{\mathscr{D}_\pi^\star\mathscr{C}_\pi} [\![Q]\!]_\pi$.
(iii) $P \cong_\pi Q$ if and only if $[\![P]\!]_\pi \cong_{\mathscr{D}_\pi\mathscr{C}_\pi} [\![Q]\!]_\pi$.

5 Observational equivalence relations for CC framework

5.1 Observational equivalence relations for classical CC

LCC languages are refinements of CC languages. Indeed, the monotonicity of the CC store can simply be restored with the exponential connective ! of linear logic, allowing duplication of hypotheses and thus avoiding constraint consumption during synchronization (Fages *et al.* 2001). Hence, all the observational equivalence relations we defined for LCC can be transposed effortless to classical CC. This is particularly interesting since few attempts can be found in the literature to endow CC with process equivalence techniques.

In order to further discuss properties of the resulting relations, we will not enter into the details of a particular encoding of CC into LCC, but just assume that the encoding of classical constraints respects two reasonable properties. We will say that a linear constraint c is *classical* within the linear constraint system $\mathscr{C}$ (or $\mathscr{C}$-classical for short), if it can be both logically weakened (i.e., $c \vdash_\mathscr{C} \mathbf{1}$), and deduced without weakening the hypotheses (i.e., for any d, if $d \vdash_\mathscr{C} c \otimes \top$, then $d \vdash_\mathscr{C} c \otimes d$). We note $\mathscr{C}_c$ the set of $\mathscr{C}$-classical constraints. Assuming that processes deal with classical constraints, we are able to prove some interesting laws. It must be underlined that in the full generality of LCC none of them holds.

Proposition 5.1
Let c, c', d, and d' be four $\mathscr{C}$-classical constraints satisfying $c \vdash_{\mathscr{C}} c'$ and $d \vdash_{\mathscr{C}} d'$. For any constraint e, all variables $\mathbf{x}$ not free in d, and all processes P and Q, the following relations hold:

(1) $\forall \mathbf{x}\,(c \to \overline{c'}) \cong_{\mathscr{C}\mathscr{C}} \overline{\mathbf{1}}$
(2) $\forall \mathbf{x}\,(c \to \overline{e}) \cong_{\mathscr{C}\mathscr{C}} \forall \mathbf{x}\,(c \to \overline{c \otimes e})$
(3) $\forall \mathbf{x}\,(c \to \overline{e}) \cong_{\mathscr{C}\mathscr{C}} \forall \mathbf{x}\,(c \to \overline{c \otimes e})$
(4) $\big((c' \to \overline{d}) \mid (c \to \overline{d'})\big) \cong_{\mathscr{C}\mathscr{C}} (c' \to d)$
(5) $(d \to \forall \mathbf{x}\,(e \to P)) \cong_{\mathscr{C}} \forall \mathbf{x}\,(d \otimes e \to P)$
(6) $!\,(c \to P) \cong_{\mathscr{C}\mathscr{C}} (c \to !P)$
(7) $\big((c \to P) \mid (c \to Q)\big) \cong_{\mathscr{C}\mathscr{C}} \big(c \to (P \mid Q)\big)$
(8) $(c \to G + c \to H) \cong_{\mathscr{C}\mathscr{C}} (c \to (G+H))$

The proof of the propositions relies on the following lemma, that states a process emits classical constraints without weaken itself.

Lemma 5.2
Let $\mathscr{D}$ and $\mathscr{E}$ be two sets of constraints, P and P' two processes, and c a $\mathscr{C}$-classical constraint. If $P \xrightarrow{\overline{c}}_{\mathscr{C}} P'$ then $P \equiv_{\mathscr{C}} P'$.

The may-testing relation $\simeq_{\mathscr{C}_c}$ coincides with an equivalence used by Saraswat to connect operational and denotational semantics of CC (Saraswat *et al.* 1991). Weaker versions of laws (1)–(6) are proved indirectly for this relation. Saraswat has also defined a bisimulation semantics for CC (Saraswat and Rinard 1990). The bisimulation he proposed is strong (i.e., it is based on $\xrightarrow{\alpha}_{\mathscr{C}}$ instead of $\overset{\alpha}{\Rightarrow}_{\mathscr{C}}$), and is, therefore, maybe too discriminative for an asynchronous framework such as CC. For instance, none of the above laws, except (2), can hold for any reasonable notion of strong bisimulation. This difference aside, Saraswat's bisimulation seems still too discriminative. Indeed, on contrary to $\cong_{\mathscr{C}_c\mathscr{C}_c}$, it distinguishes processes like $(x<1 \to \overline{c}) \,|\, (x<2 \to \overline{c})$ and $(x<2 \to \overline{c}) \,|\, (x<2 \to \overline{c})$ (where $x < y$ is the usual arithmetic inequality constraint), whereas there is no reasonable justification to do so (in both strong and weak case).

5.2 Observational equivalence relations for CHR

The CHR programming language (Frühwirth 2009) is a multiset rewriting language over first-order terms with constraints over arbitrary mathematical structures. Initially introduced for programming constraint solvers, CHR has evolved since to a programming language in its own right.

5.2.1 CHR syntax

The formalization of CHR assumes a language of *built-in constraints* containing the equality $=$, **false**, and **true** over some theory $\mathscr{CT}$ and defines *user-defined constraints* using a different set of predicate symbols. We require the nonlogical axioms of $\mathscr{CT}$ to be formulas of the form $\forall(\mathbb{C} \to \exists Z.\mathbb{D})$, where both $\mathbb{C}$ and $\mathbb{D}$ stand for possibly empty conjunctions of built-in constraints. Constraint theories satisfying such requirements correspond to Saraswat's *simple constraints systems* (1991).

A *CHR program* is a finite set of eponymous rules of the form $(r \;@\; \mathbb{K}\backslash\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{C}, \mathbb{B})$, where $\mathbb{K}$, $\mathbb{H}$ are multisets of user-defined constraints, called *kept head* and *removed head*, respectively, $\mathbb{G}$ is a conjunction of built-in constraints called *guard*, $\mathbb{C}$

Table 2. *Translation from CHR to LCC*

Built-in const.	$(c_1 \wedge \cdots \wedge c_n)^\times$	$=$	$!c_1 \otimes \cdots \otimes !c_n$
CHR const.	$(d_1, \ldots, d_n)^\times$	$=$	$d_1 \otimes \cdots \otimes d_n$
Rules	$r \mathbin{@} (\mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{B})^\times$	$=$	$!\forall(\mathbb{K}^\times \otimes \mathbb{H}^\times \otimes \mathbb{G}^\times \to \exists Y\,(\mathbb{K}^\times \otimes \mathbb{G}^\times \otimes \mathbb{B}^\times))$
Program	$\{r_1, \ldots, r_n\}^\times$	$=$	$r_1^\times \mid \cdots \mid r_n^\times$
State	$\langle \mathbb{E}; \mathbb{C}; X \rangle^\circ$	$=$	$\exists\,(\mathbb{E}^\times \otimes \mathbb{C}^\times)$

where $Y = \big(\mathrm{fv}(\mathbb{G}, \mathbb{B}) \setminus \mathrm{fv}(\mathbb{H}, \mathbb{K})\big)$ and $Z = \big(\mathrm{fv}(\mathbb{E}, \mathbb{C}) \setminus X\big)$.

is a conjunction of built-in constraints, $\mathbb{B}$ is a multiset of user-defined constraints, and r is an arbitrary identifier assumed unique in the program called *rule name*. Rules, where both heads are empty are prohibited. The empty guard **true** can be omitted together with the symbol |. Similarly, empty kept heads can be omitted together with the symbol \. Propagation rules (i.e., rules with empty removed head) can be written using the alternative syntax: $r \mathbin{@} \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{C}, \mathbb{B}$. A *state* is a tuple $\langle \mathbb{C}; \mathbb{E}; X \rangle$, where $\mathbb{C}$ is a multiset of CHR constraints, $\mathbb{E}$ is a conjunction of built-in constraints, and X is a set of variables.

5.2.2 From constraints handling rules to LCC

In a recent paper, Martinez (2010) has proposed a translation from CHR to a subset of LCC (and vice versa) that preserves language semantics with strong bisimilarity. This result allows us to transpose straightforwardly our different notions of observational equivalence to CHR. To the best of our knowledge, it is the first attempt to provide CHR with such equivalence techniques.

In Table 2, we recall Martinez's LCC interpretation of basic CHR constructs. A CHR state σ together with a CHR program $\mathscr{P}$ are interpreted as the process $(\sigma^\times | \mathscr{P}^\times)$. The constraint theory $\mathscr{CT}$ is translated using a standard translation of intuitionistic logic into linear logic. More precisely, in the remainder of this section, $(\mathscr{C}, \Vdash_{\mathscr{C}})$ is the constraint system, where $\mathscr{C}$ is built from the built-in and CHR constraints and $\Vdash_{\mathscr{C}}$ is defined by : $(\forall(\mathbb{C} \to \exists \mathbb{D})) \in \mathscr{CT}$ if and only $\mathbb{C}^\times \vdash_{\mathscr{C}} \exists X \mathbb{D}^\times$.

Due to space limitation, we do not recall the operational semantics of CHR, but use translations of CHR as particular instances of LCC processes. Thanks to Martinez's semantics preservation theorem (2010), we can do so without loss of generality as long as the CHR abstract semantics is concerned. In fact, we know that for any CHR state σ and any CHR program $\mathscr{P}$, $(\sigma^\times, \mathscr{P}^\times) \overset{\tau}{\Rightarrow}_{\mathscr{C}} Q$ if and only if σ can be rewritten by $\mathscr{P}$ (with respect to CHR abstract semantics) to a state σ' subject to $Q \equiv (\sigma'^\times, \mathscr{P}^\times)$. For the sake of conciseness, we will write $\sigma \mapsto_{\mathscr{P}} \sigma'$ for $(\sigma^\times | \mathscr{P}^\times) \overset{\tau}{\Rightarrow}_{\mathscr{C}} (\sigma^\times | \mathscr{P}^\times)$.

5.2.3 Confluence up to

Confluence is an important property for CHR programs, which ensures that any computation for a goal results in the same final state (i.e., modulo the structural equivalence $\equiv_{\mathscr{C}}$), no matter which of the applicable rules are used. Here, we propose a straightforward extension, called confluence up to, where structural equivalence

is replaced by an observational one. The resulting notion differs from the so-called observable confluence (Duck *et al.* 2007) in the following sense: observable confluence consists of proving that a program is confluent on an interesting subset of the states, while confluence up to consists of proving that a (possibly nowhere confluent) program is apparently confluent to an external observer.

Definition 5.2 (*Confluence up to*)
Let $\mathscr{D}$ and $\mathscr{E}$ be two sets of linear constraints. A CHR program $\mathscr{P}$ is *confluent up to* $\cong_{\mathscr{D}\mathscr{E}}$ if whenever $\sigma \mapsto^*_{\mathscr{P}} \sigma_1$ and $\sigma \mapsto^*_{\mathscr{P}} \sigma_2$, there exist σ_1 and σ_2 such that $\sigma_1 \mapsto^*_{\mathscr{P}} \sigma'_1$, $\sigma_2 \mapsto^*_{\mathscr{P}} \sigma'_2$, and $(\sigma'^{\times}_1|\mathscr{P}^{\times}) \cong_{\mathscr{D}\mathscr{E}} (\sigma'^{\times}_2|\mathscr{P}^{\times})$.

The following proposition states that CHR transitions with respect to a confluent program are not observable by any barbed congruences observing only classical constraints. The choice of limiting observation to classical constraints makes sense since CHR programs are usually embedded in a (host language) module that prohibits an external observer synchronizing on internal CHR constraints; the observer can only post CHR constraints using the module interface. As it is the case for Proposition 5.1, the proof relies on Lemma 5.2.

Proposition 5.4
Let $\mathscr{C}_c$ be a set of $\mathscr{C}$-classical constraints and $\mathscr{D}$ a set of linear constraints. If $\mathscr{P}$ is confluent up to $\cong_{\mathscr{C}_c\mathscr{D}}$, then $(\sigma^{\times}|\mathscr{P}^{\times}) \stackrel{\tau}{\Rightarrow}_{\mathscr{C}} P$ implies $(\sigma^{\times}|\mathscr{P}^{\times}) \cong_{\mathscr{C}_c\mathscr{D}} P$.

As corollary, we obtain that barbed congruences and may-testing equivalences coincide when they observe only classical (i.e., built-in) constraints. This supports the intuitive idea that a confluent program has the same meaning in the "backtracking" and the "committed choice" execution paradigms—bearing in mind that both relations are the respective instances of observation equivalences for these paradigms.

Corollary 5.5
Let $\mathscr{C}_c$ be a set of $\mathscr{C}$-classical constraints. Let $\mathscr{P}$ and $\mathscr{P}'$ be two CHR programs confluent up to $\cong_{\mathscr{C}_c\mathscr{D}}$. For all states σ and σ', $(\sigma^{\times}|\mathscr{P}^{\times}) \simeq_{\mathscr{C}_c\mathscr{D}} (\sigma'^{\times}|\mathscr{P}'^{\times})$ if and only if $(\sigma^{\times}|\mathscr{P}^{\times}) \cong_{\mathscr{C}_c\mathscr{D}} (\sigma'^{\times}|\mathscr{P}'^{\times})$.

5.2.4 Application

Observational equivalences are commonly used to prove correctness of a realistic (or efficient) implementation with respect to a given specification. See, for instance, numerous examples in Milner's book (1989). Here, we illustrate such a use in the context of CHR. For instance, let us assume given the following specification program $\mathscr{P}_s$:

symmetry	$@\ \mathsf{eq}(x, y) \Longrightarrow \mathsf{eq}(y, x)$
transitivity	$@\ \mathsf{eq}(x, y), \mathsf{eq}(y, z) \Longrightarrow \mathsf{eq}(y, z)$
decompose	$@\ \mathsf{eq}(\mathrm{t}(x_f, x_l, x_r), \mathrm{t}(y_f, y_l, y_r)) \Longrightarrow x_f = y_f, \mathsf{eq}(x_l, y_l), \mathsf{eq}(x_r, y_r)$

One can be easily convinced that this program specifies a Rational Terms solver limited to labeled binary trees: a binary node is represented by a term $\mathrm{t}(x_f, x_l, x_r)$, where x_f is a label (or functor), and x_l, x_r are the left and right subtrees, respectively.

Here, we aim at providing a program observationally equivalent to $\mathscr{P}_s$ that is usable in practice. As argued previously, since a CHR solver is typically isolated in a host module, it is reasonable to restrict the power of the observer such that it cannot observe CHR constraints and can post only public (or exported) CHR constraints. Hence, we choose $\mathscr{C}_c$ and $\mathscr{C}_c^{\mathsf{eq}} = \mathscr{C}_c \cup \mathscr{C}^{\mathsf{eq}}$ (where $\mathscr{C}^{\mathsf{eq}}$ is the set of constraints of the form $\mathsf{eq}(s,t)$) as input and output filters, respectively. Since CHR is a committed choice language, we have to provide a program $\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}$-barbed congruent with $\mathscr{P}_s$.

A possible implementation for the Rational Terms problem has been proposed by Frühwirth (2009). This program uses extra-logical constraints such as $var/1$. Here, we prefer writing pure programs since the status of the extra-logical constraints is not firmly defined in the theoretical semantics. For this reason, we propose the program $\mathscr{P}_i$ given below. To solve the problem, this program roughly emulates Prolog's unification algorithm (Aït-Kaci 1991)—a constraint $\mathsf{eq}(t,s)$ encodes an equations to be solved, and a constraint $x \twoheadrightarrow t$ encodes the unification (or the binding) of a variable x with a term t. We argue that $\mathscr{P}_i$ is more realistic than $\mathscr{P}_s$ since it terminates under the refined semantics of CHR (Duck *et al.* 2004)—which selects rules in the syntactical order—whereas $\mathscr{P}_s$ has no terminating derivation.

reflex	$@\ \mathsf{eq}(x,x) \iff \mathbf{true}.$
decompose	$@\ \mathsf{eq}(\mathrm{t}(x_f,x_l,x_r),\mathrm{t}(y_f,y_l,y_r)) \iff x_f = y_f, \mathsf{eq}(x_l,y_l), \mathsf{eq}(x_r,y_r).$
orient	$@\ \mathsf{eq}(\mathrm{t}(x_f,x_l,x_r),y) \iff \mathsf{eq}(y,\mathrm{t}(x_f,x_l,x_r)).$
deref_left	$@\ x \twoheadrightarrow z \backslash \mathsf{eq}(x,y) \iff \mathsf{eq}(z,y)$
deref_right	$@\ y \twoheadrightarrow z \backslash \mathsf{eq}(x,y) \iff \mathsf{eq}(x,z)$
unif	$@\ \mathsf{eq}(x,y) \iff x \twoheadrightarrow y.$

Unfortunately, $\mathscr{P}_i$ is not $\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}$-barbed congruent with the specification $\mathscr{P}_s$. For instance, for any σ_s subject to $\langle \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathbf{true}, \emptyset\rangle \mapsto^*_{\mathscr{P}_s} \sigma_s$, we have $\mathbf{false} \in \mathscr{O}^{\mathscr{C}_c}(\sigma_s^{\times}|\mathscr{P}_s^{\times})$, but for $\sigma_i = \langle x \twoheadrightarrow \mathrm{t}(a,y,z)), x \twoheadrightarrow \mathrm{t}(a,y,z)), \mathbf{true}, \emptyset\rangle$, we have $\langle \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathbf{true}, \emptyset\rangle \mapsto^*_{\mathscr{P}_i} \sigma_i$ and $\mathbf{false} \notin \mathscr{O}^{\mathscr{C}_c}(\sigma_i^{\times}|\mathscr{P}_i^{\times})$. One simple idea to circumvent this problem is to "complete" P_i (Abdennadher and Frühwirth 1998) (i.e., to make it confluent by adding new rules). For instance, one can add at the end of $\mathscr{P}_i$ the following rules. Intuitively, these rules "repair" states that do not respect the binding invariant (i.e., only variables are bound, only once, and not to themselves), which is normally preserved by the refined semantics—as far as the observer do not performed built-in unification.

repair_1	$@\ \mathrm{t}(x_f,x_l,x_r) \twoheadrightarrow y \iff \mathsf{eq}(y,\mathrm{t}(x_f,x_l,x_r)).$
repair_2	$@\ x \twoheadrightarrow y \backslash x \twoheadrightarrow z \iff \mathsf{eq}(x,z).$
repair_3	$@\ x \twoheadrightarrow x \iff \mathbf{true}.$

The resulting program $\mathscr{P}_i^{\star}$ is confluent up to $\cong_{\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}}$ and $\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}$-barbed congruent with $\mathscr{P}_s$. The proof can be sketched as follows: assume the function $(\,)^{\mathsf{eq}}$ defined on atomic constraints as $c^{\mathsf{eq}} = eq(t,s)$ if c is of the form $(t \twoheadrightarrow s)$, or $c^{\mathsf{eq}} = c$ otherwise. Consider the relation $\mathscr{R} = \{(P^{\times}|c), (P^{\times}|c^{\mathsf{eq}}) | c \in \mathscr{C}\}$, where $()^{\mathsf{eq}}$ is extended to nonatomic constraints in the straightforward way. First, we prove by coinductive reasoning on the transition from $(P^{\times}|c)$ that $\mathscr{R}$ is a $\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}$-bismulation, or thanks to Theorem 3.8, a $\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}$-barbed congruence. Then, by using a straightforward extension of strong confluence for abstract rewriting system (Huet 1980), we show that $\mathscr{P}_i^{\star}$ is confluent

up to $\mathscr{R}$, i.e., confluent up to $\cong_{\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}}$. Finally, we demonstrate by a structural induction on the $\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}$-contexts that $\mathscr{P}_i^\star \simeq_{\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}} \mathscr{P}_s$, or thanks to Corollary 5.5, $\mathscr{P}_i^\star \cong_{\mathscr{C}_c\mathscr{C}_c^{\mathsf{eq}}} \mathscr{P}_s$.

Therefore, $\mathscr{P}_i^\star$ is a correct implementation of $\mathscr{P}_s$. But, since we have proven that $\mathscr{P}_i^\star$ is also confluent, we know it can be interpreted under any rule selection strategy (in particular, under the one of the refined semantics) without loosing completeness. For this reason, and because the "repair" rules are never called under the refined semantics as long as the observer does not performed built-in unification, $\mathscr{P}_i$ interpreted in the refined semantics is also a correct implementation of $\mathscr{P}_s$. Note that Frühwirth's Rational Terms also cannot deal with built-in unifications because of the nonmonotonicity of extra-logical constraints, while $\mathscr{P}_i^\star$ can.

To the best of our knowledge, the only existing notion of equivalence for CHR programs that can be related to observation equivalences is the so-called operational equivalence (Abdennadher and Frühwirth 1999). This notion means that given two confluent and terminating programs, the computation of a query in both programs terminates in the same state. Nonetheless, we argue that observable equivalences are more general than operational equivalence since they can also be applied to programs such as $\mathscr{P}_i^*$, which is nonterminating, nonconfluent, and whose final states contain distinct CHR constraints

6 Conclusion

In the first part of this paper, we have defined and investigated a structural operational semantics for LCC with quantified ask and replication. In light of this new semantics, we have proposed and studied several observational equivalence relations. To the best of our knowledge, it is the first attempt to provide LCC with such tools, even though it was identified early on as a worthwhile goal of investigation by Ruet (Ruet and Fages 1997).

In the second part of this paper, we related LCC and its observational equivalence to asynchronous process and CC frameworks. In particular, we have shown that the asynchronous π-calculi can be viewed as subcalculi of LCC. We have shown, moreover, that some of the usual observational equivalence relations defined for this calculus are particular instances of the ones we have defined for LCC. Finally, we have shown that LCC observational equivalences can be transposed straightforwardly to classical CC and CHR. We have demonstrated some interesting properties of the resulting equivalences. In particular, we have studied the relation between barbed-congruence and confluence of CHR programs. We also illustrated how barbed congruence can be used to prove realistic implementation constraint solvers with respect to a simple specification.

An immediate further work could be to investigate the properties of the observational equivalence relations presented here. For instance, establishing sufficient conditions to ensure that observational equivalences are full congruences would be interesting. It should also be worthwhile to formally compare LCC with more exotic asynchronous calculi, such as hybrid process calculi with constraints (Díaz *et al.* 1998; Parrow and Victor 1998; Gilbert and Palamidessi 2000; Buscemi and Montanari 2007) or extended calculi with security primitives (Abadi *et al.* 2000),

where the linear constraint system would play a more prominent role. Finally, the further investigation of CHR bisimulation seems promising.

Acknowledgements

The research leading to these results has received funding from the Madrid Regional Government under the CM project P2009/TIC/1465 (PROMETIDOS), the Spanish Ministry of Science under the MEC project TIN-2008-05624 *DOVES*, and the EU Seventh Framework Programme FP/2007-2013 under grant agreement 215483 (S-CUBE).

We thank the reviewers for their helpful and constructive comments.

References

Abadi, M., Fournet, C. and Gonthier, G. 2000. Authentication primitives and their compilation. In *Principles of Programming Languages*. ACM Press, 302–315.

Abdennadher, S. and Frühwirth, T. 1998. On completion of constraint handling rules. In *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 1520, Springer, 25–39.

Abdennadher, S. and Frühwirth, T. W. 1999. Operational equivalence of CHR programs and constraints. In *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 1713. Springer, 43–57.

Aït-Kaci, H. 1991. *Warren's Abstract Machine, A Tutorial Reconstruction*. Logic Programming. MIT Press.

Amadio, R. M., Castellani, I. and Sangiorgi, D. 1998. On bisimulations for the asynchronous pi-calculus. *Theoretical Computer Science 195*(2), 291–324.

Berry, G. and Boudol, G. 1992. The chemical abstract machine. *Theoretical Computer Science 96*(1), 217–248.

Best, E., de Boer, F. and Palamidessi, C. 1997. Partial order and SOS semantics for linear constraint programs. In *Proc. of Coordination*. Lecture Notes in Computer Science, vol. 1282. Springer, 256–273.

Buscemi, M. G. and Montanari, U. 2007. CC-Pi: A constraint-based language for specifying service level agreements. In *European Symposium on Programming*. Lecture Notes in Computer Science, vol. 4421. Springer, 18–32.

Díaz, J. F., Rueda, C. and Valencia, F. D. 1998. Pi+- calculus: A calculus for concurrent processes with constraints. *CLEI Electronic Journal 1*(2).

Duck, G. J., Stuckey, P. J., García de la Banda, M. and Christian, H. 2004. The refined operational semantics of Constraint Handling Rules. In *Proc. of International Conference on Logic Programming*. Lecture Notes in Computer Science, vol. 3132. Springer, 90–104.

Duck, G. J., Stuckey, P. J. and Sulzmann, M. 2007. Observable confluence for constraint handling rules. In *Proc. of International Conference on Logic Programming*. Lecture Notes in Computer Science, vol. 4670. Springer, 224–239.

Fages, F., Ruet, P., and Soliman, S. 2001. Linear concurrent constraint programming: Operational and phase semantics. *Information and Computation 165*(1), 14–41.

Fournet, C. and Gonthier, G. 2005. A hierarchy of equivalences for asynchronous calculi. *Journal of Logic Algebra Programming 63*(1), 131–173.

Frühwirth, T. 2009. *Constraint Handling Rules*. Cambridge University Press.

Gilbert, D. and Palamidessi, C. 2000. Concurrent constraint programming with process mobility. In *Computational Logic*. Lecture Notes in Computer Science, vol. 1861. Springer, 463–477.

Girard, J.-Y. 1987. Linear logic. *Theoretical Computer Science 50*(1), 1–101.

Haemmerlé, R. 2011. *Toward Observational Equivalences for Linear Logic Concurrent Constraint Languages*. Technical Report CLIP5/2011, Technical University of Madrid (UPM).

Haemmerlé, R., Fages, F. and Soliman, S. 2007. Closures and modules within linear logic concurrent constraint programming. In *Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science, vol. 4855. Springer, 554–556.

Honda, K. and Yoshida, N. 1995. On reduction-based process semantics. *Theoretical Computer Science 151*(2), 437–486.

Huet, G. 1980 October. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *Journal of the ACM 27*(4), 797–821.

Jaffar, J. and Lassez, J.-L. 1987. Constraint logic programming. In *Proc. of 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, 111–119.

Kahn, K. M. and Saraswat, V. A. 1990. Actors as a special case of concurrent constraint programming. In *Proc. of Object Oriented Programming, Systems, Languages and Applications/European Conference on Object-Oriented Programming*, 57–66.

Laneve, C. and Montanari, U. 1992. Mobility in the CC-paradigm. In *Mathematical Foundations of Computer Science*. Lecture Notes in Computer Science, vol. 629. Springer, 336–345.

Maher, M. J. 1987. Logic semantics for a class of committed-choice programs. In *Proc. of International Conference on Logic Programming (ICLP '87)*. MIT Press.

Martinez, T. 2010. Semantics-preserving translations between linear concurrent constraint programming and constraint handling rules. In *Principles and Practice of Declarative Programming*. ACM Press, 57–66.

Milner, R. 1989. *Communication and Concurrency*. Prentice Hall.

Milner, R. and Sangiorgi, D. 1992. Barbed bisimulation. In *Proc. of International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 623. Springer, 685–695.

Nicola, R. D. and Hennessy, M. 1984. Testing equivalences for processes. *Theoretical Computer Science 34*, 83–133.

Parrow, J. and Victor, B. 1998. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Logic in Computer Science*. IEEE, 176–185.

Ruet, P. and Fages, F. 1997. Concurrent constraint programming and mixed non-commutative linear logic. In *Proc. of Annual Conference on Computer Science Logic*. Lecture Notes in Computer Science, vol. 1414. Springer, 406–423.

Saraswat, V. A. and Lincoln, P. 1992. *Higher-Order Linear Concurrent Constraint Programming*. Technical Report, Xerox Parc.

Saraswat, V. A. and Rinard, M. C. 1990. Concurrent constraint programming. In *Principles of Programming Languages*. ACM Press, 232–245.

Saraswat, V. A., Rinard, M. C. and Panangaden, P. 1991. Semantic foundations of concurrent constraint programming. In *Principles of Programming Languages*. ACM, 333–352.

Soliman, S. 2003. Pi-calculus and LCC, a space odyssey. Research Report 4855, Institut National de Recherche en Informatique et en Automatique.

Tamura, N. 1998. *User's Guide of a Linear Logic Theorem Prover*. [online]. Kobe University. URL: `http://bach.istc.kobe-u.ac.jp/llprover/`.