Toward Observational Equivalences for Linear Logic Concurrent Constraint Languages

May 2011

facultad de informática

universidad politécnica de madrid

Rémy Haemmerlé

TR Number CLIP5/2011.0

Technical Report Number: CLIP5/2011.0
May, 2011

Authors

Rémy Haemmerlé
Technical University of Madrid

Keywords

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

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 LCC observational equivalences can be transposed straightforwardly to classical Concurrent Constraint languages and Constraint Handling Rules, and investigate the resulting equivalences.

Contents

1 Introduction

The class of Concurrent Constraint languages (briefly, CC) (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 CCS, the CHAM, or the π-calculus. For example, its semantics has been originally expressed by process algebras similar to CSS (Saraswat and Rinard 1990) or a later in the style of the CHAM (Fages, Ruet, and Soliman 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 (briefly, 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, de Boer, and Palamidessi 1997; Ruet and Fages 1997; Fages, Ruet, and Soliman 2001; Haemmerlé, Fages, and Soliman 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, the π+-calculus (Díaz, Rueda, and Valencia 1998), the extended CC of Gilbert and Palamidessi (Gilbert and Palamidessi 2000), or the more recent CC-π (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 propose also 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 (briefly, LTS). We then show that the asynchronous π-calculus can be viewed as a sub-calculus 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 Constraints Handling Rules (CHR).

The paper is organized as follows. In Section 2, we introduce LCC in a generalized framework with replication and without CC declarations as we have partially done in a previous work (Haemmerlé, Fages, and Soliman 2007). We express the operational semantics of this language by means of an LTS and show that linear logical semantics of LCC (Fages, Ruet, and Soliman 2001) still holds, proving at the same time that the operational semantics proposed here is coherent with the usual one. Section 3 presents different parametric equivalencies for LCC processes, namely logical equivalence, may testing equivalence, labeled bisimulation, and barbed congruence. In Section 4, we present a simple LCC interpretation of the asynchronous π-calculus and prove that some of the equivalences we proposed in the previous section coincide with the usual ones for this paradigm. Finally, in Section 5 we explain how LCC observational equivalences can be transposed to classical CC and CHR using existing LCC interpretations of CC and CHR, and investigate properties of resulting equivalences.

2 A process calculi semantics for Linear Logic CC

In this paper, we assume given a denumerable set $\mathcal{V}$ 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 $\mathcal{V}$ 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, Fages, and Soliman 2007). In LCC, we distinguish four syntactical categories as specified by the following grammar:

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

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 or logic formulas $\{A_1, \ldots, A_n\}$, we will use $\bigotimes_{i=1}^n A_i$, $\Pi_{i=1}^n A_i$, and $\Sigma_{i=1}^n A_i$ as abbreviations for$A_1 \otimes \cdots \otimes A_m$, $A_1|\cdots|A_m$, and $A_1 + \cdots + A_n$ respectively. 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 $(\mathcal{C}, \Vdash_{\mathcal{C}})$ where $\mathcal{C}$ is the set of all constraints and $\Vdash_{\mathcal{C}}$ is a subset of $\mathcal{C} \times \mathcal{C}$ which defines the non-logical axioms of the system. For a given constraint system $(\mathcal{C}, \Vdash_{\mathcal{C}})$, the entailment relation $\vdash_{\mathcal{C}}$ is the smallest relation containing $\Vdash_{\mathcal{C}}$ and closed by the rules of intuitionistic linear logic (See Appendix A). We will use the notation $A \dashv\vdash_{\mathcal{C}} B$ to mean that both sequents $A \vdash_{\mathcal{C}} B$ and $B \vdash_{\mathcal{C}} A$ 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 the observer in LCC sub-calculi and/or the visibility limitations imposed by ad-hoc scope mechanisms such as module systems. In practice, they will be specified by means of two subsets of $\mathcal{C}$, that will limit the possible constraints a process can respectively ask or tell. Formally for all subsets $\mathcal{D}$ and $\mathcal{E}$, we say that a process P is *$\mathcal{D}$-ask restricted* (resp. *$\mathcal{E}$-tell restricted*) if it is obtained by the grammar for processes where any ask $\forall\mathbf{x}(c \rightarrow P)$ (resp. any tell $\overline{c}$) satisfies $(\exists\mathbf{x}.c) \in \mathcal{D}$ (resp. $c \in \mathcal{E}$). More generally, we say that P is a $\mathcal{DE}$-process if P is both $\mathcal{D}$-ask and $\mathcal{E}$-tell restricted.

2.2 Operational semantics

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

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

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

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

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

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

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

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

Table 1: Labeled transition system for Linear Logic CC

The notion of *weak transition* is defined classically:

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

In the asynchronous context of this paper, it seems natural to restrict the observation to outputs. As argued by Amadio, Castellani, and Sangiorgi (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, Rinard, and Panangaden 1991; Fages, Ruet, and Soliman 2001), and consider only accessible constraints (Haemmerlé, Fages, and Soliman 2007). Formally for any set $\mathcal{D}\subset\mathcal{C}$, the set of *$\mathcal{D}$-accessible constraints* for a process P is defined as:

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

The semantics we propose has important links with the one defined by Best, de Boer, and Palamidessi (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, Fages, and Soliman 2007); while, on the other hand universal quantification cannot be emulated by tell processes in every constraint system, especially linear ones (Fages, Ruet, and Soliman 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, Castellani, and Sangiorgi 1998).

Example 2.1 (Dining philosophers)**.** *As suggested by Best, de Boer, and Palamidessi (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* $\vdash_{\mathcal{C}}$ *is the trivial entailment relation. Assuming the following encoding for the* i^{th} *philosopher among* n*, a solution for the problem consists of the process* $\Pi_{i=0}^{n-1}\left(P_i^n|\overline{\mathtt{frk}(i)}\right)$.

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

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, Ruet, and Soliman 2001; Haemmerlé, Fages, and Soliman 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 3.2.) Note the conjunction with $\top$ is not necessary in case of translation of a parallel composition and hidings, since it commutes with $\otimes$ and $\exists$ (i.e. $(A \otimes \top) \otimes (B \otimes \top) \dashv\vdash_{\mathcal{C}} A \otimes B \otimes \top$ and $\exists x(A \otimes \top) \dashv\vdash_{\mathcal{C}} \exists x(A) \otimes \top$).

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

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

Theorem 2.3 (Logical semantics)**.** *For any process* P *and set* $\mathcal{D}$ *of linear constraints,*

$$\mathcal{O}^{\mathcal{D}}(P) = \left\{ d \in \mathcal{D} \mid P^{\dagger} \vdash_{\mathcal{C}} d^{\dagger} \right\}.$$

3 Observational equivalence relations for Linear logic CC

In this section, we propose some equivalence relations for LCC-processes. Most of them are related to the ones defined by Kobaychi and Yonezawa for ACL (Kobayashi and Yonezawa 1993), a concurrent linear logic language. It is worth noting that the important differences between ACL and LCC make their work inapplicable to the framework we consider here. First, the linear logic interpretation of process in both families of languages is dual. That is, in ACL, A transits to B if $B \vdash A$ whereas, in LCC, A transits to B if $A \vdash B$. Secondly, ACL provides only recursion while recursion can be replaced by replication in LCC (Haemmerlé, Fages, and Soliman 2007). Last but not least, LCC is parametrized by a constraint system that has no equivalent in ACL.

3.1 Contexts and congruences

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 that 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, *$\mathcal{DE}$-contexts* and *$\mathcal{DE}$-congruence* will refer to evaluation contexts and congruence built from $\mathcal{DE}$-processes.

3.2 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:*

$$P \multimap\!\!\circ_{\mathcal{C}} Q \overset{def}{\Longleftrightarrow} P^{\dagger} \dashv\vdash_{\mathcal{C}} Q^{\dagger}$$

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.*

The proof is based on the following technical lemma, that shows weak logical equivalence is a full-precongruence.

Lemma 3.3. *For all processes P, Q, and any arbitrary context C, if $P^{\dagger} \vdash_{\mathcal{C}} Q^{\dagger}$, then $C[P]^{\dagger} \vdash_{\mathcal{C}} C[Q]^{\dagger}$.*

3.3 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 3.4 (May-testing equivalence)**.** *Let $\mathcal{D}$ and $\mathcal{E}$ be two subsets of $\mathcal{C}$. The may $\mathcal{DE}$-testing, $\simeq_{\mathcal{DE}}$, is the largest $\mathcal{DE}$-congruence that respects $\mathcal{D}$-accessible constraints, formally:*

$$P \simeq_{\mathcal{DE}} Q \overset{def}{\Longleftrightarrow} \textit{for any evaluation } \mathcal{DE}\textit{-context } C[\,],\ \mathcal{O}^{\mathcal{D}}(C[P]) = \mathcal{O}^{\mathcal{D}}(C[Q]).$$

Quite naturally, logical equivalence implies any may testing equivalence relation. One can use logical semantics and Theorem 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 3.5. *Contrary to the processes in Theorem 2.1, the following implementation for the i^{th} dining philosopher does not use atomic consumptions of constraint conjunctions:*

$$Q_i^n = \;!\Big(\mathtt{frk}(i) \to \Big(\mathtt{frk}(i{+}1 \ mod\ n) \to \Big(\overline{\mathtt{eat}(i)} | \mathtt{eat}(i) \to \Big(\overline{\mathtt{frk}(i)} | \overline{\mathtt{frk}(i{+}1 \ mod\ n)}\Big)\Big)\Big)\Big)$$

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 $i, n \in \mathbb{N}$, $P_i^n \simeq_{\mathcal{CC}} Q_i^n$). 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.

3.4 Labeled Bisimulation

In the framework of process algebra, bisimulation-based equivalence relations are the most commonly used notion of equivalence. Contrary to testing equivalence and barbed congruence presented in the next subsection, the labeled bisimulation proofs do not require explicit context closure. Indeed, as shows Theorem 3.7, 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 and output filters.

Definition 3.6 (Labeled bisimulation)**.** *Let $\mathcal{D}$ and $\mathcal{E}$ be two subsets of $\mathcal{C}$. A action is $\mathcal{DE}$-relevant for a process Q if it is either a silent action, or an input action in $\mathcal{E}$, or an output action of the form $(\mathbf{x})\overline{c}$ with $(\exists \mathbf{x}.c) \in \mathcal{D}$ and $\mathbf{x} \cap \mathrm{fv}(Q) = \emptyset$. A symmetrical relation $\mathcal{R}$ is a $\mathcal{DE}$-bisimulation if for all P, P', Q, α such that $P\mathcal{R}Q$, $P \xrightarrow{\alpha}_{\mathcal{C}} P'$, and α is $\mathcal{DE}$-relevant for Q, there exists Q' such that $Q \overset{\alpha}{\Rightarrow}_{\mathcal{C}} Q'$ and $P'\mathcal{R}Q'$. The largest $\mathcal{DE}$-bisimulation is called $\mathcal{DE}$-bisimilarity and is denoted with $\approx_{\mathcal{DE}}$.*

Theorem 3.7. *For all sets of constraints $\mathcal{D}$ and $\mathcal{E}$, the $\mathcal{DE}$-bisimilarity is a $\mathcal{DE}$-congruence.*

3.5 Barbed congruence

Barbed bisimulation has been introduced by Milner and Sangiorgi (1992) as an 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 $\nvdash_{\mathcal{C}} c$), which exhibit clearly different behaviours 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 3.8 (Barbed congruence)**.** *Let $\mathcal{D}$ and $\mathcal{E}$ be two subsets of $\mathcal{C}$. A symmetrical relation $\mathcal{R}$ is a $\mathcal{DE}$-barbed bisimulation if for all P, P', Q such that $P\mathcal{R}Q$, and $P \xrightarrow{\tau}_{\mathcal{C}} P'$, then there exists Q' such that $\mathcal{O}^{\mathcal{D}}(P) \subseteq \mathcal{O}^{\mathcal{D}}(Q)$, $Q \overset{\tau}{\Rightarrow}_{\mathcal{C}} Q'$ and $P'\mathcal{R}Q'$. The barbed $\mathcal{DE}$-congruence, written $\cong_{\mathcal{DE}}$, is the largest $\mathcal{DE}$-congruence that is a $\mathcal{DE}$-barbed bisimulation.*

Clearly, barbed $\mathcal{DE}$-congruence is more precise than may $\mathcal{DE}$-testing equivalence. It is worth noting that it is in general strictly distinct from logical equivalence. For instance, $c \rightarrow \exists x.P$ and $\exists x.(c \rightarrow P)$ are $\mathcal{CC}$-barbed congruent but not logically equivalent, while $c \rightarrow d \rightarrow \mathbf{1}$ and $c \otimes d \rightarrow \mathbf{1}$ 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 3.9. *For all sets of constraints $\mathcal{D}$ and $\mathcal{E}$, $\cong_{\mathcal{DE}}$ and $\approx_{\mathcal{DE}}$ coincide.*

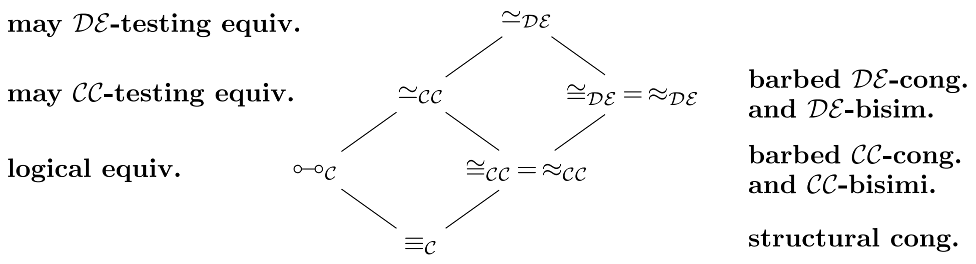
Example 3.10. *The encoding of philosophers proposed in the two previous examples cannot be distinguished by may-testing. Nonetheless their behavior can be separated by barbed congruence. For instance, one can disprove $P_1^3 \cong_{\mathcal{CC}} Q_1^3$. The following implementation refines the one of Theorem 3.5 by allowing a philosopher to put back the first fork he takes:*

$$R_i^n = !\Big(\mathtt{frk}(i) \rightarrow \Big(\overline{\mathtt{frk}(i)} + \mathtt{frk}(i{+}1 \;\mathit{mod}\; n) \rightarrow \Big(\overline{\mathtt{eat}(i)} | \mathtt{eat}(i) \rightarrow \Big(\overline{\mathtt{frk}(i)} | \overline{\mathtt{frk}(i{+}1 \;\mathit{mod}\; n)}\Big)\Big)\Big)\Big)$$

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

3.6 A hierarchy of equivalences for Linear CC

Figure 1 summarizes the relationship between all process equivalence relations we have discussed so far. All solid lines represent inclusions which are in the general case strict, equivalences at the bottom being more discriminatiing than the ones at the top. In this figure, $\mathcal{D}$ and $\mathcal{E}$ stands for arbitrary subsets of $\mathcal{C}$.



$$\begin{tikzcd}
\textbf{may } \mathcal{DE}\textbf{-testing equiv.} & & & \simeq_{\mathcal{DE}} \arrow[dl, dash] \arrow[dr, dash] & & \\
\textbf{may } \mathcal{CC}\textbf{-testing equiv.} & & \simeq_{\mathcal{CC}} \arrow[dl, dash] \arrow[dr, dash] & & \cong_{\mathcal{DE}} = \approx_{\mathcal{DE}} \arrow[dl, dash] & \textbf{barbed } \mathcal{DE}\textbf{-cong. and } \mathcal{DE}\textbf{-bisim.} \\
\textbf{logical equiv.} & \multimapboth_{\mathcal{C}} \arrow[dr, dash] & & \cong_{\mathcal{CC}} = \approx_{\mathcal{CC}} \arrow[dl, dash] & & \textbf{barbed } \mathcal{CC}\textbf{-cong. and } \mathcal{CC}\textbf{-bisimi.} \\
& & \equiv_{\mathcal{C}} & & & \textbf{structural cong.}
\end{tikzcd}$$

Figure 1: Inclusion of process equivalence relations in LCC

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 non-blocking. In practice, it is obtained by a simple syntactical restriction prohibiting output prefixing.

4.1 Syntax and Operational Semantics of the asynchronous π-calculus

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* νyP. Both input prefix $x(y).P$ and restriction $\nu y.P$ bind the variable y in P. Abbreviations $\mathrm{fv}(\alpha)$ and $\mathrm{bv}(\alpha)$ are defined for the π-actions in the following way:

$$\mathrm{fv}((y)\bar{x}(y)) = \{x\} \quad \mathrm{fv}(\bar{x}(y)) = \mathrm{fv}(xy) = \{x, y\} \quad \mathrm{bv}(\bar{x}(y)) = \{y\} \quad \mathrm{bv}(\bar{x}y) = \mathrm{bv}(xy) = \emptyset$$

The operational semantics is based on the structural congruence $\equiv_\pi$, defined as the smallest congruence satisfying α-renaming of bound variables, commutativity and associativity for parallel composition, together with the following identities:

$$P|\mathbf{0} \equiv_\pi P \quad \nu x\mathbf{0} \equiv_\pi \mathbf{0} \quad \nu x\nu yP \equiv_\pi \nu y\nu xP \quad !P \equiv_\pi !P|P \quad \frac{z \notin \mathrm{fv}(P)}{P|\nu zQ \equiv_\pi \nu z(P|Q)}$$

Labeled reduction step $\xrightarrow{\alpha}_\pi$ of the pure asynchronous π-calculus is the smallest relation on processes that meets the rules of Table 2 together with rule (*cong*) as defined for LCC in Table 1. The notion of *weak transition* for the π-calculus, $\Rightarrow_\pi$ is defined classically.

$$\tau.P \xrightarrow{\tau}_\pi P \quad (\pi\text{-}\tau) \qquad \mathbf{0} \xrightarrow{xz}_\pi \bar{x}z \quad (\pi\textit{-in}) \qquad \bar{x}y \xrightarrow{\bar{x}y}_\pi \mathbf{0} \quad (\pi\textit{-out})$$

$$\frac{P \xrightarrow{\alpha}_\pi P' \quad \mathrm{ev}(\alpha) \cap \mathrm{fv}(Q) = \emptyset}{P|Q \xrightarrow{\alpha}_\pi P'|Q} \ (\pi\textit{-comp}) \quad \bar{x}z|x(y).P \xrightarrow{\tau}_\pi P[y\backslash z] \quad (\pi\textit{-sync})$$

$$\frac{P \xrightarrow{\bar{x}y}_\pi Q \quad x \neq y}{\nu yP \xrightarrow{\bar{x}(y)}_\pi Q} \ (\pi\textit{-out}_{ex}) \qquad \frac{P \xrightarrow{\alpha}_\pi Q \quad x \notin \mathrm{fv}(\alpha) \cup \mathrm{ev}(\alpha)}{\nu xP \xrightarrow{\alpha}_\pi \nu xQ} \ (\pi\textit{-rest})$$

Table 2: Labeled transition system for the asynchronous π-calculus

4.2 Observational equivalence relations for asynchronous π-calculus

We recall now some classical observational equivalence relations for the asynchronous π-calculus. As for LCC we will need a notion of context. In particular, the *evaluation contexts* of the π-calculus (or π-*contexts,* for short) are given by the following grammar, where Q stands for an arbitrary π-process:

$$C_\pi[\,] = [\,] \mid (Q|C_\pi[\,]) \mid (C_\pi[\,]|Q) \mid \nu x.C_\pi$$

The basic observation predicate is the *commitment* on x, $\Downarrow_{\bar{x}}$, that detects whether a process P emits a message on the name x, that is formally:

$$P \Downarrow_{\bar{x}} \overset{def}{\Longleftrightarrow} \text{ there exist a } \pi\text{-process } Q \text{ and a name } y \text{ s.t.} P \overset{\bar{x}y}{\Longrightarrow}_\pi Q \text{ or } P \overset{\bar{x}(y)}{\Longrightarrow}_\pi Q$$

All the observational equivalence relations we consider have been studied by Fournet and Gonthier (Fournet and Gonthier 2005):

- The *may testing equivalence* is the largest congruence, $\simeq_\pi$, respecting $\Downarrow_{\bar{x}}$, that is $P \simeq_\pi Q$ if for any evaluation π-context $C[\,]$ and any name x, $C[P] \Downarrow_{\bar{x}}$ is equivalent to $C[Q] \Downarrow_{\bar{x}}$.
- The *labeled bisimilarity* is the largest symmetrical relation, $\approx_\pi$, verifying $P \approx_\pi Q$, $P \xrightarrow{\alpha}_\pi P'$ and $\mathrm{ev}(\alpha) \cap \mathrm{fv}(Q) = \emptyset$ implies there exists Q' such that $Q \overset{\alpha}{\Rightarrow}_\pi Q'$ and $P' \approx_\pi Q'$.
- A *barbed bisimulation* is a symmetrical relation $\mathcal{R}$ verifying whenever $P\mathcal{R}Q$:
  - $P \Downarrow_{\bar{x}}$ implies $Q \Downarrow_{\bar{x}}$
  - $P \overset{\tau}{\Rightarrow}_\pi P'$ implies the existence of a Q' such taht $Q \overset{\tau}{\Rightarrow}_\pi Q'$ and $P'\mathcal{R}Q'$.
- The *barbed congruence* is the largest congruence, $\cong_\pi$, that is a barbed bisimulation.

4.3 LCC interpretation of the pure asynchronous π-calculus

We propose now a very simple interpretation of the asynchronous π-calculus into LCC following the preliminary ideas of Soliman (Soliman 2003).

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

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

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

Theorem 4.2. *For all two π-processes P and Q and any π-action α we have:*

$$P \xrightarrow{\alpha}_\pi Q \text{ if and only if } [\![P]\!]_\pi \xrightarrow{[\![\alpha]\!]_\pi}_{\mathcal{C}_\pi} [\![Q]\!]_\pi.$$

The theorem and the simplicity of the interpretation emphasizes 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 $\mathcal{D}_\pi = \{\exists y.\gamma(x, y) \mid x \in \mathcal{V} \setminus \{y\}\}$ and $\mathcal{D}^\star_\pi = \mathcal{D}_\pi \cup \{\gamma(x, y) \mid xy \in \mathcal{V}\}$. For all π-processes P and Q we have:*

(i) $P \simeq_\pi Q$ *if and only if* $[\![P]\!]_\pi \simeq_{\mathcal{D}_\pi \mathcal{C}_\pi} [\![Q]\!]_\pi$.

(ii) $P \approx_\pi Q$ *if and only if* $[\![P]\!]_\pi \approx_{\mathcal{D}^\star_\pi \mathcal{C}_\pi} [\![Q]\!]_\pi$.

(iii) $P \cong_\pi Q$ *if and only if* $[\![P]\!]_\pi \cong_{\mathcal{D}_\pi \mathcal{C}_\pi} [\![Q]\!]_\pi$.

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, Ruet, and Soliman 2001). Hence, all the observational equivalence relations we defined for LCC can be transposed effortless to classical CC. That 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 $\mathcal{C}$ (or $\mathcal{C}$-classical for short), if it can be both logically weakened (i.e. $c \vdash_\mathcal{C} \mathbf{1}$), and deduced without weakening the hypotheses (i.e. for any d, if $d \vdash_\mathcal{C} c \otimes \top$, then $d \vdash_\mathcal{C} c \otimes d$). We note $\mathcal{C}_c$ the set of $\mathcal{C}$-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 $\mathcal{C}$-classical constraints satisfying $c \vdash_\mathcal{C} c'$ and $d \vdash_\mathcal{C} d'$. For any constraint e, all variables $\mathbf{x}$ not free in d, and all processes P and Q, the following relations hold:*

(1) $\forall \mathbf{x}\,(c \to \overline{c'}) \cong_{\mathcal{C}\mathcal{C}} \overline{\mathbf{1}}$ (2) $\forall \mathbf{x}\,(c \to \overline{e}) \cong_{\mathcal{C}\mathcal{C}} \forall \mathbf{x}\,(c \to \overline{c \otimes e})$

(3) $\forall \mathbf{x}\,(c \to \overline{e}) \cong_{\mathcal{C}\mathcal{C}} \forall \mathbf{x}\,(c \to \overline{c \otimes e})$ (4) $\left((c' \to \overline{d}) \mid (c \to \overline{d'})\right) \cong_{\mathcal{C}\mathcal{C}} (c' \to d)$

(5) $(d \to \forall \mathbf{x}\,(e \to P)) \cong_{\mathcal{C}} \forall \mathbf{x}\,(d \otimes e \to P)$ (6) $!\,(c \to P) \cong_{\mathcal{C}\mathcal{C}} (c \to !P)$

(7) $((c \to P) \mid (c \to Q)) \cong_{\mathcal{C}\mathcal{C}} (c \to (P \mid Q))$ (8) $(c \to G + c \to H) \cong_{\mathcal{C}\mathcal{C}} (c \to (G + H))$

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

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

The may-testing relation $\simeq_{\mathcal{C}_c}$ coincides with an equivalence used by Saraswat to connect operational and denotational semantics of CC (Saraswat, Rinard, and Panangaden 1991). Weaker versions of laws (1) to (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 $\xrightarrow{\alpha}_{\mathcal{C}}$ instead of $\overset{\alpha}{\Rightarrow}_{\mathcal{C}}$), 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 $\cong_{\mathcal{C}_c\mathcal{C}_c}$, it distinguishes processes like $(x<1 \to \overline{c}) \mid (x<2 \to \overline{c})$ and $(x<2 \to \overline{c}) \mid (x<2 \to \overline{c})$ (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 Constraint Handling Rules (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 Constraint Handling Rules Syntax

The formalization of CHR assumes a language of *built-in constraints* containing the equality $=$, **false**, and **true** over some theory $\mathcal{CT}$ and defines *user-defined constraints* using a different set of predicate symbols. We require the non-logical axioms of $\mathcal{CT}$ 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*, $\mathbb{C}$ 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 $\mid$. Similarly, empty keptheads can be omitted together with the symbol $\backslash$. Propagation rules (i.e.rules with empty removed head) can be written using the alternative syntax: $r \,@\, \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{C}, \mathbb{B}$. 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.

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 @ $(\mathbb{K} \backslash \mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{B})^\times$	$=$	$!\forall(\mathbb{K}^\times \otimes \mathbb{H}^\times \otimes \mathbb{G}^\times \to \exists Y(\mathbb{K}^\times \otimes \mathbb{G}^\times \otimes \mathbb{B}^\times))$
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 $Y = (\mathrm{fv}(\mathbb{G}, \mathbb{B}) \setminus \mathrm{fv}(\mathbb{H}, \mathbb{K}))$ and $Z = (\mathrm{fv}(\mathbb{E}, \mathbb{C}) \setminus X)$.

Table 3: Translation from CHR to LCC

5.2.2 From Constraints Handling Rules to Linear Logic CC

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

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 $\mathcal{P}$, $(\sigma^\times, \mathcal{P}^\times) \overset{\tau}{\Rightarrow}_\mathcal{C} Q$ if and only if σ can be rewritten by $\mathcal{P}$ (w.r.t. CHR abstract semantics) to a state σ' s.t. $Q \equiv (\sigma'^\times, \mathcal{P}^\times)$. For the sake of conciseness, we will write $\sigma \mapsto_\mathcal{P} \sigma'$ for $(\sigma^\times | \mathcal{P}^\times) \overset{\tau}{\Rightarrow}_\mathcal{C} (\sigma^\times | \mathcal{P}^\times)$.

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 $\equiv_\mathcal{C}$) 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 form the so-called observable confluence (Duck, Stuckey, and Sulzmann 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 5.3 (Confluence up-to). *Let $\mathcal{D}$ and $\mathcal{E}$ be two sets of linear constraints. A CHR program $\mathcal{P}$ is confluent up to $\cong_{\mathcal{DE}}$ if whenever $\sigma \mapsto_\mathcal{P}^* \sigma_1$ and $\sigma \mapsto_\mathcal{P}^* \sigma_2$, there exist σ_1 and σ_2 such that $\sigma_1 \mapsto_\mathcal{P}^* \sigma_1'$, $\sigma_2 \mapsto_\mathcal{P}^* \sigma_2'$, and $(\sigma_1'^\times | \mathcal{P}^\times) \cong_{\mathcal{DE}} (\sigma_2'^\times | \mathcal{P}^\times)$.*

The following proposition states that CHR transitions w.r.t. a confluent program are not observable by any barbed congruences obsevring only classical constraints. The choice of limiting observation to classical constraints makes sens since CHR programs are usually embedded in a (host language) module that prohibs 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 Theorem 5.1, the proof relies on Theorem 5.2.

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

As corollary, we obtain that barbed congruences and may-testing equivalences conincide 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 "commited choice" exuction paradigms – bearing in mind that both relations are the respective instances of observation equivalences for these paradigms.

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

5.2.4 Application

Observational equivalences are commonly used to prove correctness of a realistic (or efficient) implementation w.r.t. 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 $\mathcal{P}_s$:

$$\begin{array}{ll}
\text{symmetry} & @\ \mathtt{eq}(x,y) \Longrightarrow \mathtt{eq}(y,x) \\
\text{transitivity} & @\ \mathtt{eq}(x,y), \mathtt{eq}(y,z) \Longrightarrow \mathtt{eq}(y,z) \\
\text{decompose} & @\ \mathtt{eq}(\mathrm{t}(x_f,x_l,x_r), \mathrm{t}(y_f,y_l,y_r)) \Longrightarrow x_f = y_f, \mathtt{eq}(x_l,y_l), \mathtt{eq}(x_r,y_r)
\end{array}$$

One can be easily convinced that this program specifies a Rational Terms (RT) solver limited to labelled 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 $\mathcal{P}_s$ 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 $\mathcal{C}_c$ and $\mathcal{C}_c^{\mathtt{eq}} = \mathcal{C}_c \cup \mathcal{C}^{\mathtt{eq}}$ (where $\mathcal{C}^{\mathtt{eq}}$ is the set of constraints of the form $\mathtt{eq}(s,t)$) as input and output filters, respectively. Since CHR is a committed choice language, we have to provide a program $\mathcal{C}_c\mathcal{C}_c^{\mathtt{eq}}$-barbed congruent with $\mathcal{P}_s$.

A possible implementation for the RT 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 $\mathcal{P}_i$ given below. To solve the problem, this program roughly emulates Prolog's unification algorithm (Aït-Kaci 1991) – a constraint $\mathtt{eq}(t,s)$ encodes an equations to be solved, and a constraint $x \multimap t$ encodes the unification (or the binding) of a variable x with a term t. We argue that $\mathcal{P}_i$ is more realistic than $\mathcal{P}_s$ since it terminates under the refined semantics of CHR (Duck et al. 2004) – which selects rules in the syntactical order whereas $\mathcal{P}_s$ has no terminating derivation.

$$
\begin{array}{ll}
\text{reflex} & @\ \mathsf{eq}(x,x) \iff \mathbf{true}. \\
\text{decompose} & @\ \mathsf{eq}(\mathrm{t}(x_f,x_l,x_r),\mathrm{t}(y_f,y_l,y_r)) \iff x_f = y_f, \mathsf{eq}(x_l,y_l), \mathsf{eq}(x_r,y_r). \\
\text{orient} & @\ \mathsf{eq}(\mathrm{t}(x_f,x_l,x_r),y) \iff \mathsf{eq}(y,\mathrm{t}(x_f,x_l,x_r)). \\
\text{deref\_left} & @\ x \twoheadrightarrow z \backslash \mathsf{eq}(x,y) \iff \mathsf{eq}(z,y) \\
\text{deref\_right} & @\ y \twoheadrightarrow z \backslash \mathsf{eq}(x,y) \iff \mathsf{eq}(x,z) \\
\text{unif} & @\ \mathsf{eq}(x,y) \iff x \twoheadrightarrow y.
\end{array}
$$

Unfortunately, $\mathcal{P}_i$ is not $\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}$-barbed congruent with the specification $\mathcal{P}_s$. For instance, for any σ_s s.t. $\langle \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathbf{true}, \emptyset\rangle \mapsto^*_{\mathcal{P}_s} \sigma_s$, we have $\mathbf{false} \in \mathcal{O}^{\mathcal{C}_c}(\sigma_s^{\times}|\mathcal{P}_s^{\times})$, 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 $\langle \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathsf{eq}(x,\mathrm{t}(a,y,z)), \mathbf{true}, \emptyset\rangle \mapsto^*_{\mathcal{P}_i} \sigma_i$ and $\mathbf{false} \notin \mathcal{O}^{\mathcal{C}_c}(\sigma_i^{\times}|\mathcal{P}_i^{\times})$. 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 $\mathcal{P}_i$ 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.

$$
\begin{array}{ll}
\text{repair}_1 & @\ \mathrm{t}(x_f,x_l,x_r) \twoheadrightarrow y \iff \mathsf{eq}(y,\mathrm{t}(x_f,x_l,x_r)). \\
\text{repair}_2 & @\ x \twoheadrightarrow y \backslash x \twoheadrightarrow z \iff \mathsf{eq}(x,z). \\
\text{repair}_3 & @\ x \twoheadrightarrow x \iff \mathbf{true}.
\end{array}
$$

The resulting program $\mathcal{P}_i^{\star}$ is confluent up to $\cong_{\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}}$ and $\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}$-barbed congruent with $\mathcal{P}_s$. 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 $\mathcal{R} = \{(P^{\times}|c),(P^{\times}|c^{\mathsf{eq}})|c \in \mathcal{C}\}$ where $()^{\mathsf{eq}}$ is extended to non-atomic constraints in the straightforward way. First, we prove by coindutive reasoning on the transition from $(P^{\times}|c)$ that $\mathcal{R}$ is a $\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}$-bismulation, or thanks to Theorem 3.9 a $\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}$-barbed congruence. Then, by using a straightforward extension of strong confluence for abstract rewritting system (Huet 1980), we show that $\mathcal{P}_i^{\star}$ is confluent up to $\mathcal{R}$, i.e., confluent up to $\cong_{\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}}$. Finally, we demonstrate by a structural induction on the $\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}$-contexts that $\mathcal{P}_i^{\star} \simeq_{\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}} \mathcal{P}_s$, or thanks to Theorem 5.5, $\mathcal{P}_i^{\star} \cong_{\mathcal{C}_c\mathcal{C}_c^{\mathsf{eq}}} \mathcal{P}_s$.

Therefore, $\mathcal{P}_i^{\star}$ is a correct implementation of $\mathcal{P}_s$. But, since we have proven that $\mathcal{P}_i^{\star}$ 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, $\mathcal{P}_i$ interpreted in the refined semantics is also a correct implementation of $\mathcal{P}_s$. Note that Frühwirth's RT also cannot deal with built-in unifications because of the non-monotonicity of extra-logical constraints, while $\mathcal{P}_i^{\star}$ 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 $\mathcal{P}_i^{\star}$ which is non-terminating, non-confluent, 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 illustrated also how barbed-congruence can be used to prove realistic implementation constraint solvers w.r.t. 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, Rueda, and Valencia 1998; Parrow and Victor 1998; Gilbert and Palamidessi 2000; Buscemi and Montanari 2007) or extended calculi with security primitives (Abadi, Fournet, and Gonthier 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).

References

Abadi, Martín, Cédric Fournet, and Georges Gonthier. 2000. “Authentication Primitives and Their Compilation.” *POPL*. ACM PRESS, 302–315.

Abdennadher, Slim, and Thom Frühwirth. 1998. “On Completion of Constraint Handling Rules.” *CP*, Volume 1520 of *LNCS*. Springer, 25–39.

Abdennadher, Slim, and Thom W. Frühwirth. 1999. “Operational Equivalence of CHR Programs and Constraints.” *CP*, Volume 1713 of *LNCS*. Springer, 43–57.

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

Amadio, Roberto M., Ilaria Castellani, and Davide Sangiorgi. 1998. "On Bisimulations for the Asynchronous pi-Calculus." *Theor. Comput. Sci.* 195 (2): 291–324.

Berry, Gérard, and Gérard Boudol. 1992. "The Chemical Abstract Machine." *Theor. Comput. Sci.* 96 (1): 217–248.

Best, Eike, Frank de Boer, and Catuscia Palamidessi. 1997. "Partial Order and SOS Semantics for Linear Constraint Programs." *COORDINATION*, Volume 1282 of *LNCS*. Springer, 256–273.

Buscemi, Maria Grazia, and Ugo Montanari. 2007a. "CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements." *ESOP*, Volume 4421 of *LNCS*. Springer, 18–32.

———. 2007b. "Open Bisimulation for the Concurrent Constraint Pi-Calculus." *ESOP, LNCS* 4960:254–268.

Díaz, Juan Francisco, Camilo Rueda, and Frank D. Valencia. 1998. "Pi+- Calculus: A Calculus for Concurrent Processes with Constraints." *CLEI Electron. J.* 1, no. 2.

Duck, Gregory J., Peter J. Stuckey, María García de la Banda, and Holzbaur Christian. 2004. "The Refined Operational Semantics of Constraint Handling Rules." *ICLP*, Volume 3132 of *LNCS*. Springer, 90–104.

Duck, Gregory J., Peter J. Stuckey, and Martin Sulzmann. 2007. "Observable Confluence for Constraint Handling Rules." *ICLP*, Volume 4670 of *LNCS*. Springer, 224–239.

Fages, François, Paul Ruet, and Sylvain Soliman. 2001. "Linear Concurrent Constraint Programming: Operational and Phase Semantics." *Inf. Comput.* 165 (1): 14–41.

Fournet, Cédric, and Georges Gonthier. 2005. "A hierarchy of equivalences for asynchronous calculi." *J. Log. Algebr. Program.* 63 (1): 131–173.

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

Gilbert, David, and Catuscia Palamidessi. 2000. "Concurrent Constraint Programming with Process Mobility." *Computational Logic*, Volume 1861 of *LNCS*. Springer, 463–477.

Girard, Jean-Yves. 1987. "Linear logic." *Theoretical Computer Science*, vol. 50(1).

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

Haemmerlé, Rémy, François Fages, and Sylvain Soliman. 2007. "Closures and Modules within Linear Logic Concurrent Constraint Programming." *FSTTCS*, Volume 4855 of *LNCS*. Springer, 554–556.

Honda, Kohei, and Nobuko Yoshida. 1995. "On Reduction-Based Process Semantics." *Theor. Comput. Sci.* 151 (2): 437–486.

Huet, Gérard. 1980. "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 (October).

Jaffar, Joxan, and Jean-Louis Lassez. 1987, January. "Constraint logic programming." *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany.* ACM, 111–119.

Kahn, Kenneth M., and Vijay A. Saraswat. 1990. "Actors as a Special Case of Concurrent Constraint Programming." *OOPSLA/ECOOP*. 57–66.

Kobayashi, Naoki, and Akinori Yonezawa. 1993. "Logical, testing and observation equivalence for processes in a linear logic programming." Technical Report 93-4, Dep. of Computer Science, University of Tokyo.

Laneve, Cosimo, and Ugo Montanari. 1992. "Mobility in the CC-Paradigm." *MFCS*, Volume 629 of *LNCS*. Springer, 336–345.

Maher, Michael J. 1987. "Logic semantics for a class of committed-choice programs." *Proceedings of ICLP'87, International Conference on Logic Programming.* MIT Press.

Martinez, Thierry. 2010. "Semantics-preserving translations between Linear Concurrent Constraint Programming and Constraint Handling Rules." *PPDP.* ACM PRESS, 57–66.

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

Milner, Robin, and Davide Sangiorgi. 1992. "Barbed Bisimulation." *ICALP*, Volume 623 of *LNCS.* Springer, 685–695.

Nicola, Rocco De, and Matthew Hennessy. 1984. "Testing Equivalences for Processes." *Theor. Comput. Sci.* 34:83–133.

Parrow, Joachim, and Björn Victor. 1998. "The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes." *LICS.* IEEE, 176–185.

Ruet, Paul, and François Fages. 1997. "Concurrent constraint programming and mixed non-commutative linear logic." *CSL*, Volume 1414 of *LNCS.* Springer, 406–423.

Saraswat, Vijay A., and Patrick Lincoln. 1992. "Higher-order linear concurrent constraint programming." Technical Report, Xerox Parc.

Saraswat, Vijay A., and Martin C. Rinard. 1990. "Concurrent Constraint Programming." *POPL.* ACM PRESS, 232–245.

Saraswat, Vijay A., Martin C. Rinard, and Prakash Panangaden. 1991. "Semantic foundations of concurrent constraint programming." *POPL.* ACM.

Soliman, Sylvain. 2003. "Pi-calculus and LCC, a Space Odyssey." Research report 4855, INRIA.

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

A Intuitionistic linear logic sequent calculus

$$A \vdash A \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Delta, \Gamma \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma, \mathbf{1} \vdash A} \qquad \vdash \mathbf{1} \qquad \Gamma \vdash \top \qquad \Gamma, \mathbf{0} \vdash A$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}$$

$$\frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Delta, \Gamma, A \multimap B \vdash C} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}$$

$$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A} \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}$$

$$\frac{\Gamma, A[t/x] \vdash B}{\Gamma, \forall x A \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\ x \notin \mathrm{fv}(\Gamma) \qquad \frac{\Gamma, A \vdash B}{\Gamma, \exists x A \vdash B}\ x \notin \mathrm{fv}(y\Gamma, B) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

B Proofs

B.1 Preliminaries

In this section, we show the proof tree of a transition can always be assumed to be in a special form, we call *normal*. This preliminary result will simplify nomber of proofs.

For any rule $(\star)$, let $(\star^{\equiv})$ be the *meta-rule* (i.e. composition of atomic rules) obtained by composition (form bottom to top) of (*cong*) following by $(\star)$, e.g. ($\mathcal{C}$-*sync*$^{\equiv}$) and ($\mathcal{C}$-*comp*$^{\equiv}$), are notations for the respective following meta-rules.

$$\dfrac{P \equiv_{\mathcal{C}} P' \quad \dfrac{}{P' \xrightarrow{\tau}_{\mathcal{C}} Q'}\,(\mathcal{C}\text{-sync}) \quad Q' \equiv_{\mathcal{C}} Q}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\text{cong})$$

$$\dfrac{P \equiv_{\mathcal{C}} P' \quad \dfrac{\pi}{P' \xrightarrow{\tau}_{\mathcal{C}} Q'}\,(\mathcal{C}\text{-comp}) \quad Q' \equiv_{\mathcal{C}} Q}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\text{cong})$$

We said that a (meta-)rules $(\star)$ permutes with a (meta-)rule $(\bullet)$ if :

$$\text{For any } \dfrac{\dfrac{\pi}{P' \xrightarrow{\tau}_{\mathcal{C}} Q'}\,(\bullet)}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\star) \text{ there exists a proof of the form } \dfrac{\dfrac{\pi}{P'' \xrightarrow{\tau}_{\mathcal{C}} Q''}\,(\star)}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\bullet)$$

We said that a (meta-)rules $(\star)$ merges with a (meta-)rule $(\bullet)$ if :

$$\text{For any } \dfrac{\dfrac{\pi}{P' \xrightarrow{\tau}_{\mathcal{C}} Q'}\,(\star)}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\bullet) \text{ there exists a proof of the form } \dfrac{\pi}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\star)$$

Remark B.1. *($\mathcal{C}$-comp$^{\equiv}$) permutes with ($\mathcal{C}$-rest$^{\equiv}$) and ($\mathcal{C}$-out$^{\equiv}_{\text{ex}}$). (sum$^{\equiv}$) permutes with ($\mathcal{C}$-rest$^{\equiv}$) and ($\mathcal{C}$-comp$^{\equiv}$). ($\mathcal{C}$-sync$^{\equiv}$), ($\mathcal{C}$-comp$^{\equiv}$), ($\mathcal{C}$-rest$^{\equiv}$), and (sum$^{\equiv}$) merge with (cong). ($\mathcal{C}$-comp$^{\equiv}$) merges with itself.*

For any meta-rule $(\bullet)$, $(\bullet)^*$ stands for an arbitrary number of successive applications of the rules $(\bullet)$.

Lemma B.2. *Let P and Q be two processes without summation. If $P \xrightarrow{\alpha}_{\mathcal{C}} Q$ is a valid transition, then it has a proof of one of the three following forms:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P'' \xrightarrow{\tau}_{\mathcal{C}} Q''}\,(\mathcal{C}\text{-sync}^{\equiv})}{P'' \xrightarrow{\tau}_{\mathcal{C}} Q''}\,(\mathcal{C}\text{-sum}^{\equiv})^*}{P' \xrightarrow{\tau}_{\mathcal{C}} Q'}\,(\mathcal{C}\text{-comp}^{\equiv})}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\mathcal{C}\text{-rest}^{\equiv})^* \qquad \dfrac{\dfrac{\dfrac{}{P'' \xrightarrow{c}_{\mathcal{C}} Q''}\,(\mathcal{C}\text{-in}^{\equiv})^*}{P' \xrightarrow{c}_{\mathcal{C}} Q'}\,(\mathcal{C}\text{-comp}^{\equiv})}{P \xrightarrow{c}_{\mathcal{C}} Q}\,(\mathcal{C}\text{-rest}^{\equiv})^* \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{P''' \xrightarrow{(\mathbf{z}''')\overline{c'''}}_{\mathcal{C}} Q'''}\,(\mathcal{C}\text{-out}^{\equiv})}{P'' \xrightarrow{(\mathbf{z}'')\overline{c''}}_{\mathcal{C}} Q''}\,(\mathcal{C}\text{-comp}^{\equiv})^*}{P' \xrightarrow{(\mathbf{z}')\overline{c'}}_{\mathcal{C}} Q'}\,(\mathcal{C}\text{-out}^{\equiv}_{\text{ex}})^*}{P \xrightarrow{(\mathbf{z})\overline{c}}_{\mathcal{C}} Q}\,(\mathcal{C}\text{-rest}^{\equiv})^*$$

Proof. First, notice that any proof tree of $P \xrightarrow{\tau}_{\mathcal{C}} Q$ can be transformed straightforwardly into a proof using only (*cong*), ($\mathcal{C}$-*comp*$^{\equiv}$), ($\mathcal{C}$-*sync*$^{\equiv}$), ($\mathcal{C}$-*rest*$^{\equiv}$), (*sum*$^{\equiv}$), ($\mathcal{C}$-*out*$^{\equiv}_{\text{ex}}$), ($\mathcal{C}$-*out*$^{\equiv}$), and ($\mathcal{C}$-*in*$^{\equiv}$) by inserting a rule (*cong*) between each rule. The resulting tree can be transformed into an other one of the expected form by applying remark B.1 as much as possible. ☐

Lemma B.3 (Normal form proof)**.** *Let P and Q be two processes without summation. If $P \xrightarrow{\alpha}_{\mathcal{C}} Q$ is a valid transition, then it has a proof of one of the three following forms:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{c \vdash_{\mathcal{C}} \exists \mathbf{y}(d_k[\mathbf{x}_k\backslash\mathbf{t}] \otimes e)}{\overline{c}|\forall \mathbf{x}_k(d_k \to Q_k) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{y}\,(\overline{e}|Q_k[\mathbf{x}_k\backslash\mathbf{t}])}\,(\mathcal{C}\textit{-sync})}{\overline{c}|\Sigma_{j\in J}\forall \mathbf{x}_j(d_j \to Q_j) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{y}\,(\overline{e}|Q_k[\mathbf{x}_k\backslash\mathbf{t}])}\,(\mathcal{C}\textit{-sum})^*}{\Pi_{i\in I}G_i|\overline{c}|\Sigma_{j\in J}\forall \mathbf{x}_j(d_j \to Q_j) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{y}\,(\Pi_{i\in I}G_i|\overline{e}|Q_k[\mathbf{x}_k\backslash\mathbf{t}])}\,(\mathcal{C}\textit{-comp}^{\equiv})}{\exists \mathbf{z}\,(\Pi_{i\in I}G_i|\overline{c}|\Sigma_{j\in J}\forall \mathbf{x}_j(d_j \to Q_j)) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{zy}\,(\Pi_{i\in I}G_i|\overline{e}|Q_k[\mathbf{x}_k\backslash\mathbf{t}])}\,(\mathcal{C}\textit{-rest})^*}{P \xrightarrow{\tau}_{\mathcal{C}} Q}\,(\textit{cong})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{1} \xrightarrow{c}_{\mathcal{C}} \overline{c}}\,(\mathcal{C}\textit{-in})}{P' \xrightarrow{c}_{\mathcal{C}} P'|\overline{c}}\,(\mathcal{C}\textit{-comp}^{\equiv})}{\exists \mathbf{z} P' \xrightarrow{c}_{\mathcal{C}} \exists \mathbf{z}(P'|\overline{c})}\,(\mathcal{C}\textit{-rest})^*}{P \xrightarrow{c}_{\mathcal{C}} Q}\,(\textit{cong}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{c \vdash_{\mathcal{C}} \exists \mathbf{x}(d \otimes e)}{\overline{c} \xrightarrow{(\mathbf{x})\overline{d}}_{\mathcal{C}} \overline{e}}\,(\mathcal{C}\textit{-out})}{\overline{c}|P' \xrightarrow{(\mathbf{x})\overline{d}}_{\mathcal{C}} \overline{e}|P'}\,(\mathcal{C}\textit{-comp}^{\equiv})}{\exists \mathbf{y}\,(\overline{c}|P') \xrightarrow{(\mathbf{x})\overline{d}}_{\mathcal{C}} \overline{e}|P'}\,(\mathcal{C}\textit{-out}_{\text{ex}})^*}{\exists \mathbf{zy}\,(\overline{c}|P') \xrightarrow{(\mathbf{x})\overline{d}}_{\mathcal{C}} \exists \mathbf{z}\,(\overline{e}|P')}\,(\mathcal{C}\textit{-rest})^*}{P \xrightarrow{(\mathbf{x})\overline{d}}_{\mathcal{C}}}\,(\textit{cong})$$

, where I and J are two disjoint finite sets of indices and $k \in J$, the G_i' stand for possibly replicated guards, $\mathbf{y} \notin \text{fv}(\Pi_{i\in I}G_i)$, and $\mathbf{x} \cap \mathbf{z} = \emptyset$.

Proof. The case are by induction on the number of application of ($\mathcal{C}$*-rest*$^{\equiv}$), (*sum*$^{\equiv}$), and ($\mathcal{C}$*-out*$_{\text{ex}}$) in the proof π obtain by the previous lemma. □

B.2 Proofs of Section 2. (A process calculi semantics for Linear Logic CC)

The proof of this section are tedious but follows quite straightforwardly the proofs of logical semantics of (Ruet and Fages 1997; Fages, Ruet, and Soliman 2001; Haemmerlé, Fages, and Soliman 2007).

Lemma B.4. *If $P \xrightarrow{\alpha}_{\mathcal{C}} Q$ then $\text{ev}(\alpha) \cap \text{fv}(P) = \emptyset$.*

Proof. By induction on the proof of $P \xrightarrow{\alpha}_{\mathcal{C}} Q$. □

Proposition B.5 (Soundness)**.** *For all processes P and Q the four following propositions hold:*

(i) *If $P \equiv_{\mathcal{C}} Q$ then $P^{\dagger} \dashv\vdash_{\mathcal{C}} Q^{\dagger}$.*

(ii) *If $P \xrightarrow{(\mathbf{x})\overline{c}}_{\mathcal{C}} Q$ then $P^{\dagger} \vdash_{\mathcal{C}} (\exists \mathbf{x}(c \otimes Q))^{\dagger}$.*

(iii) *If $P \xrightarrow{\tau}_{\mathcal{C}} Q$ then $P^{\dagger} \vdash_{\mathcal{C}} Q^{\dagger}$.*

Proof. (i) is by induction on the derivation tree of $\equiv_{\mathcal{C}}$. Other cases are by induction on the derivation tree of $P \xrightarrow{\alpha}_{\mathcal{C}} Q$. □

Proposition B.6 (Completeness)**.** *For any multiset of processes* Γ*, any constraint* c*, and all variables* $\mathbf{x}$*:*

$$\textit{If } \Gamma^\dagger \vdash_{\mathcal{C}} \exists \mathbf{x}.c^\dagger \textit{ then there exists } P \textit{ such that } \Pi(\Gamma) \overset{(\mathbf{x})\overline{c}}{\Longrightarrow}_{\mathcal{C}} P$$

W. e prove the more general result:

$$\text{If } \Gamma^\dagger \vdash_{\mathcal{C}} c \text{ or } \Gamma^\dagger \vdash_{\mathcal{C}} c \otimes \top, \text{ then } \Pi(\Gamma) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(d|P) \text{ with } \exists \mathbf{x}.d \vdash_{\mathcal{C}} c$$

by induction on the proof π of $\Gamma^\dagger \vdash_{\mathcal{C}} c$ or $\Gamma^\dagger \vdash_{\mathcal{C}} c^\dagger$:

- π is a axiom. Γ is a multiset of constraints, that is $\Pi(\Gamma) \equiv_{\mathcal{C}} \bigotimes(\Gamma)|\bar{1}$.
- π ends with a cut (without lost of generality we can assume the cuts deal only with non-logical axioms):

$$\frac{\Gamma^\dagger \vdash_{\mathcal{C}} \quad \overline{c \vdash_{\mathcal{C}} d}}{\Gamma^\dagger \vdash_{\mathcal{C}} d} \quad \text{or} \quad \frac{\overline{c \vdash_{\mathcal{C}} c'} \quad \Gamma^\dagger, c' \vdash_{\mathcal{C}} d}{\Gamma^\dagger, c \vdash_{\mathcal{C}} d} \quad \text{or} \quad \frac{\overline{c \vdash_{\mathcal{C}} c'} \quad \Gamma^\dagger, c' \vdash_{\mathcal{C}} d \otimes \top}{\Gamma^\dagger, c \vdash_{\mathcal{C}} d \otimes \top}$$

  The first case is immediate. For the two other cases, we know, by induction hypothesis, that $(\pi(\Gamma)|c') \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}.(d|P)$ with $\exists \mathbf{x} d \vdash_{\mathcal{C}} d'$. Just notice that any ask which can be waken up by c' can be by c.

- π ends with a right introduction of $\otimes$:

$$\frac{\Gamma^\dagger \vdash_{\mathcal{C}} c \quad \Delta \vdash_{\mathcal{C}} d}{\Gamma^\dagger, \Delta^\dagger \vdash_{\mathcal{C}} c \otimes d} \quad \text{or} \quad \frac{\Gamma^\dagger \vdash_{\mathcal{C}} c \quad \Delta \vdash_{\mathcal{C}} \top}{\Gamma^\dagger, \Delta^\dagger \vdash_{\mathcal{C}} c \otimes \top}$$

  For the left hand side case, we know by induction hypothesis that $\Pi(\Gamma) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(c'|P)$ and $\Pi(\Delta) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{y}(d'|Q)$ with $\exists \mathbf{x} c' \vdash_{\mathcal{C}} c$, $\exists \mathbf{y} d' \vdash_{\mathcal{C}} d$. Without lost og generality we can assume that $\mathbf{x} \cap \text{fv}(c, d', Q) = \emptyset$ and $\mathbf{y} \cap \text{fv}(d, c, P) = \emptyset$. By monotony of $\overset{\tau}{\rightarrow}_{\mathcal{C}}$, we get $\Pi(\Delta, \Gamma) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(c'|P)|\Pi(\Delta) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(c'|P)|\exists \mathbf{y}(d'|Q)$ that is using structural congruence $\Pi(\Delta, \Gamma) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{xy}(c' \otimes d'|P|Q)$. It is straightforward to check that $\exists \mathbf{xy}(c' \otimes d') \vdash_{\mathcal{C}} c \otimes d$. the right hand side case is trivial.

- π ends with a left introduction of $\exists$:

$$\frac{\Gamma^\dagger, A\dagger \vdash_{\mathcal{C}} c}{\Gamma^\dagger, \exists x.A^\dagger \vdash_{\mathcal{C}} c} x \notin \text{fv}(\Gamma, c) \quad \text{or} \quad \frac{\Gamma^\dagger, A\dagger \vdash_{\mathcal{C}} c \otimes \top}{\Gamma^\dagger, \exists x.A^\dagger \vdash_{\mathcal{C}} c \otimes \top} x \notin \text{fv}(\Gamma, c)$$

  By induction hypothesis, we get that $A|\Pi(\Gamma) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(d|P)$ with $\exists \mathbf{x}.d \vdash_{\mathcal{C}} c$ and without lost of generality $\mathbf{x} \cap \text{fv}(c) = \emptyset$. We conclude using ($\mathcal{C}$-ext) rule.

- π ends with a left introduction of $\multimap$:

$$\frac{\Gamma^\dagger \vdash d \quad \Delta^\dagger, A^\dagger \vdash c}{\Gamma^\dagger, \Delta^\dagger, d \multimap A^\dagger \vdash_{\mathcal{C}} c} \quad \text{or} \quad \frac{\Gamma^\dagger \vdash d \quad \Delta^\dagger, A^\dagger \vdash c \otimes \top}{\Gamma^\dagger, \Delta^\dagger, d \multimap A^\dagger \vdash_{\mathcal{C}} c \otimes \top}$$

  By induction hypothesis, we know that $\Pi(\Gamma) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(d'|P)$ and (ii) $\Pi(\Delta)|A \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{y}(c'|Q)$. Without lost of generality we suppose that $\mathbf{x} \cap \mathbf{y} = \emptyset$. By monotony of $\overset{\tau}{\rightarrow}_{\mathcal{C}}$ and using the rules ($\mathcal{C}$-sync) and $\mathcal{C}$-rest), we get: $\Pi(\Gamma, \Delta) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(d'|P)|\Pi(\Delta) \equiv_{\mathcal{C}} \exists \mathbf{x} d'|P|\Pi(\Delta) \overset{\tau}{\Rightarrow}_{\mathcal{C}} \exists \mathbf{x}(c'|Q|P)$.

- ends with a left introduction of $\forall$:

  $$\frac{\Gamma, \forall \mathbf{x} A \vdash c}{\Gamma, A[\mathbf{x}\backslash\mathbf{t}] \vdash c} \quad \text{or} \quad \frac{\Gamma, \forall \mathbf{x} A \vdash c \otimes \top}{\Gamma, A[\mathbf{x}\backslash\mathbf{t}] \vdash c \otimes \top}$$

  A is necessarily the translation of an *ask* of the form $d \to B$. It is straightforward to check that if $d[\mathbf{x}\backslash\mathbf{t}] \to B[\mathbf{x}\backslash\mathbf{t}]$ can be reduced then $\forall \mathbf{x}(d \to B)$ can be reduced too.

- π ends with a dereliction: There are two subcases:

  $$\frac{\Gamma^\dagger, A^\dagger \vdash_{\mathcal{C}} c}{\Gamma^\dagger, !A^\dagger \vdash_{\mathcal{C}} c} \quad \text{or} \quad \frac{\Gamma^\dagger, A^\dagger \vdash_{\mathcal{C}} c \otimes \top}{\Gamma^\dagger, !A^\dagger \vdash_{\mathcal{C}} c \otimes \top}$$

  $!A$ is either the translation of a constraint or a replicated guard. The former subcase is immediate since any ask that can be waken by a constraint c can be waken up by the stronger constraint $!c$. For the latter case we have by induction hypothesis $\Pi(\Gamma, A) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{x} c' | P$. If A reduces during these derivation, then we infer using (repl) rules, and monotony of $\xrightarrow{\tau}_{\mathcal{C}}$, $\Pi(\Gamma, !A) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{x} c' | P | !A$. It it does not reduce we infer $\Pi(\Gamma) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{x} c' | Q$ with $P \equiv_{\mathcal{C}} A | Q$ that is $\Pi(\Gamma, !A) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{x} c' | Q | !A$.

- π ends with a contraction: there are two subcases

  $$\frac{\Gamma^\dagger, !A^\dagger, !A^\dagger \vdash_{\mathcal{C}} c}{\Gamma^\dagger, !A^\dagger \vdash_{\mathcal{C}} c} \quad \text{or} \quad \frac{\Gamma^\dagger, !A^\dagger, !A^\dagger \vdash_{\mathcal{C}} c \otimes \top}{\Gamma^\dagger, !A^\dagger \vdash_{\mathcal{C}} c \otimes \top}$$

  As in the previous case $!A$ is either the translation of a constraint or a replicated guards. The former subcase is trivial since any ask that can be waken up by a constraint $!c \otimes !c$ can be waken up by a constraint $!c$. The second one is trivial since any constraint consumed by two instances of the same guard can be consumed by only one of this instance.

- Other cases are straightforward consequence of the induction hypothesis. □

□

Proof of Theorem 2.3. The theorem is direct corollary of the two previous lemmas. □

B.3 Proofs of Section 3. (Observational equivalence relations for Linear logic CC)

B.3.1 Proofs of Section 3.2. (Logical equivalence)

Proof of **??**. y induction on the proof of $\Gamma \vdash_{\mathcal{C}} P^\dagger \otimes \top$ one can show that the relation on intuitionistic linear logic formulas $\{(\bigotimes(\Gamma), P) \mid \Gamma \vdash_{\mathcal{C}} P^\dagger \otimes \top\}$ is a full precongruence. The conclusion is then straightforward. □

B.3.2 Proofs of Section 3.4. (Labeled Bisimulation)

Note that both structural equivalence and transition are stable by renaming.

Lemma B.7 (Stability under renaming)**.** *Let P and Q be two processes, α be an action, and ρ be a renaming. We have:*

(1) *If $P \equiv_{\mathcal{C}} Q$, then $P\rho \equiv_{\mathcal{C}} Q\rho$* (2) *If $P \xrightarrow{\alpha}_{\mathcal{C}} Q$, then $P\rho \xrightarrow{\alpha\rho}_{\mathcal{C}} Q\rho$*

Proof. Let σ be a renaming. We proceed by induction on the proof tree pi of $P \equiv_{\mathcal{C}} Q$ and $P \xrightarrow{\alpha}_{\mathcal{C}} Q$:

- The cases for structural equivalence are obvious.
- The cases where π ends with a rule (cong), (sum), ($\mathcal{C}$-ext), or ($\mathcal{C}$-out) are direct by induction hypothesis.
- π ends with a rule ($\mathcal{C}$-rest)

$$\frac{P \xrightarrow{\alpha}_{\mathcal{C}} Q \quad x \notin \mathrm{fv}(\alpha)}{\exists x P \xrightarrow{\alpha}_{\mathcal{C}} \exists x Q}$$

Let y' be a fresh variable and $\rho = [yy' \backslash y'y]$. We have:

$$\begin{array}{rll}
P \xrightarrow{\alpha}_{\mathcal{C}} Q \Longrightarrow & P\rho\sigma \xrightarrow{\alpha\rho\sigma}_{\mathcal{C}} Q\rho\sigma & \text{by induction hypothesis} \\
\Longrightarrow & P\rho\sigma \xrightarrow{\alpha\sigma}_{\mathcal{C}} Q\rho\sigma & \text{because } y \notin \mathrm{fv}(\alpha) \\
\Longrightarrow & \exists y'.(P\rho\sigma) \xrightarrow{\alpha\sigma}_{\mathcal{C}} \exists y'.Q(\rho\sigma) & \text{by } (\mathcal{C}\text{-rest}) \text{ and } y' \notin \mathrm{fv}(\alpha\sigma) \\
\Longrightarrow & (\exists y'.P\rho)\sigma \xrightarrow{\alpha\sigma}_{\mathcal{C}} (\exists y'.Q\rho)\sigma & \text{because } y' \text{ is fresh} \\
\Longrightarrow & \exists y.(P)\sigma \xrightarrow{\alpha\sigma}_{\mathcal{C}} (\exists y.Q)\sigma & \text{by } \alpha\text{-renaming}
\end{array}$$

- π ends with a rule ($\mathcal{C}$-comp): Just note that if $\mathrm{ev}(\alpha) \cap \mathrm{fv}(Q) = \emptyset$ then $\mathrm{ev}(\alpha\sigma) \cap \mathrm{fv}(Q\sigma) = \emptyset$ and use induction hypothesis.
- π ends with a rule ($\mathcal{C}$-sync):

$$\frac{c \vdash_{\mathcal{C}} \exists \mathbf{y}(d[\mathbf{x} \backslash \mathbf{t}] \otimes e) \quad \mathbf{y} \cap (d, P) = \emptyset}{\overline{c} | \forall \mathbf{x}(d \to P) \xrightarrow{\tau}_{\mathcal{C}} \exists \mathbf{y}.(P[\mathbf{x} \backslash \mathbf{t}] | \overline{e})}$$

Let $\mathbf{x}'$ and $\mathbf{y}'$ two sequences of fresh variables of same length that $\mathbf{x}$ and $\mathbf{y}$ respectively. Let $\rho_x = [\mathbf{x}\mathbf{x}' \backslash \mathbf{x}'\mathbf{x}]$ and $\rho_y = [\mathbf{y}\mathbf{y}' \backslash \mathbf{y}'\mathbf{y}]$. We have:

$$\begin{array}{rll}
 & c \vdash_{\mathcal{C}} \exists \mathbf{y}(d[\mathbf{x} \backslash \mathbf{t}] \otimes e) & \\
\Longrightarrow & c \vdash_{\mathcal{C}} \exists \mathbf{y}(d\rho_x[\mathbf{x}' \backslash \mathbf{t}] \otimes e) & \\
\Longrightarrow & c \vdash_{\mathcal{C}} \exists \mathbf{y}'(d\rho_x[\mathbf{x}' \backslash \mathbf{t}]\rho_y \otimes e\rho_y) & \text{by } \alpha\text{-renaming} \\
\Longrightarrow & c \vdash_{\mathcal{C}} \exists \mathbf{y}'(d\rho_x[\mathbf{x}' \backslash \mathbf{t}\rho_y] \otimes e\rho_y) & \text{because } \mathbf{y} \cap \mathrm{fv}(d) = \emptyset \text{ and } y' \text{ are fresh} \\
\Longrightarrow & c\sigma \vdash_{\mathcal{C}} \exists \mathbf{y}'(d\rho_x[\mathbf{x}' \backslash \mathbf{t}\rho_y] \otimes e\rho_y)\sigma & \text{by stability of logical implication} \\
\Longrightarrow & c\sigma \vdash_{\mathcal{C}} \exists \mathbf{y}'(d\rho_x\sigma[\mathbf{x}' \backslash \mathbf{t}\rho_y\sigma] \otimes e\rho_y\sigma) & \text{since } x' \text{ are fresh}
\end{array}$$

$$\begin{array}{rll}
P \equiv_{\mathcal{C}} & (\overline{c} | \forall \mathbf{x}(d \to P))\sigma & \\
\equiv_{\mathcal{C}} & \overline{c}\sigma | \forall \mathbf{x}'(d\rho_x \to P\rho_x)\sigma \equiv_{\mathcal{C}} & \text{by } \alpha\text{-renaming} \\
\equiv_{\mathcal{C}} & \overline{c}\sigma | \forall \mathbf{x}'(d\rho_x\sigma \to P\rho_x\sigma) & \text{because } x' \text{ are fresh} \\
\xrightarrow{\tau}_{\mathcal{C}} & \exists y'(P\rho_x\sigma[x' \backslash \mathbf{t}\rho_y\sigma] \otimes \overline{e}\rho_y\sigma) & \text{because } c\sigma \vdash_{\mathcal{C}} \exists \mathbf{y}'(d\rho_x\sigma[\mathbf{x}' \backslash \mathbf{t}\rho_y\sigma] \otimes e\rho_y\sigma) \\
\equiv_{\mathcal{C}} & \exists y'(P\rho_x[x' \backslash \mathbf{t}\rho_y] \otimes \overline{e}\rho_y)\sigma & \text{because } x' \text{ are fresh} \\
\equiv_{\mathcal{C}} & \exists y'(P\rho_x[x' \backslash \mathbf{t}]\rho_y \otimes \overline{e}\rho_y)\sigma & \text{because } \mathbf{y} \cap \mathrm{fv}(P) = \emptyset \text{ and } y' \text{ are fresh} \\
\equiv_{\mathcal{C}} & \exists y(P\rho_x[x' \backslash \mathbf{t}] \otimes \overline{e})\sigma & \text{by } \alpha\text{-renaming} \\
\equiv_{\mathcal{C}} & \exists y(P[x \backslash \mathbf{t}] \otimes \overline{e})\sigma &
\end{array}$$

- π ends with a rule ($\mathcal{C}$-in): trivial. □

Lemma B.8. *Let $\mathcal{D}$ and $\mathcal{E}$ be a two sets of constraints.*

1. *If $P \approx_{\mathcal{DE}} Q$ holds, then for any renaming ρ $P\rho \approx_{\mathcal{DE}} Q\rho$ holds.*
2. *$\approx_{\mathcal{DE}}$ is an equivalence relation.*
3. *If $P_1 \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_2 \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_3$ and $P_1 \cong_{\mathcal{DE}} P_3$, holds then so do $P_1 \cong_{\mathcal{DE}} P_2$.*
4. *If $P \approx_{\mathcal{DE}} Q$ holds, then for any $c \in \mathcal{E}$, $P|c \approx_{\mathcal{DE}} Q|c$.*
5. *If $P \approx_{\mathcal{DE}} Q$ holds, then for any $x \in \mathcal{V}$, $\exists xP \approx_{\mathcal{DE}} \exists xQ$ holds.*
6. *If $P \approx_{\mathcal{DE}} Q$ holds, then for any $\mathcal{DE}$-guard G, $P|G \approx_{\mathcal{DE}} Q|G$ holds.*

Proof. **case 1.** Using Theorem B.7, it is straightforward to show that $\{(P\rho, Q\rho) \mid P \approx_{\mathcal{DE}} Q\}$ is a $\mathcal{DE}$-bisimulation.

case 2. The symmetry and reflexivity of $\approx_{\mathcal{DE}}$ are obvious. For transitivity we show that the following relation is a $\mathcal{DE}$-bisimulation.

$$\{(P,Q) \mid \text{ there exists } R \text{ such that } P \approx_{\mathcal{DE}} R \text{ and } R \approx_{\mathcal{DE}} Q\}$$

Let R such that $P \approx_{\mathcal{DE}} R$, $R \approx_{\mathcal{DE}} Q$ and $P \xrightarrow{\alpha}_{\mathcal{C}} P'$. The case of silent or input action is trivial. If $\alpha = (\mathbf{z})\overline{c}$, then, assuming that $\rho = [\mathbf{z}\backslash\mathbf{z}']$ is a renaming with fresh variables. We infer, using case *(i)*, there exists R' and Q' such that of $R\rho \overset{(\mathbf{z}')\overline{c\rho}}{\Longrightarrow}_{\mathcal{C}} R'\rho$ and $Q\rho \overset{(\mathbf{z}')\overline{c\rho}}{\Longrightarrow}_{\mathcal{C}} Q'\rho$ with $P\rho \approx_{\mathcal{DE}} R'\rho$ and $R'\rho \approx_{\mathcal{DE}} Q'\rho$. Using once more case *(i)* with ρ^{-1} we conclude that $Q \overset{(\mathbf{z})\overline{c}}{\Longrightarrow}_{\mathcal{C}} Q'$ with $P \approx_{\mathcal{DE}} R'$ and $R' \approx_{\mathcal{DE}} Q'$.

case 3. It is sufficient to show that the following relation is a $\mathcal{DE}$-bisimulation.

$$\Big\{(P_1, P_2) \mid P_1 \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_2 \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_3 \text{ and } P_1 \approx_{\mathcal{DE}} P_3\Big\}$$

The proof is straightforward using transitivity of $\approx_{\mathcal{DE}}$ (i.e.case 2).

case 4 Let $P \approx_{\mathcal{DE}} Q$. We have:

$$\begin{array}{lll}
 & P \xrightarrow{c}_{\mathcal{C}} (P|c) & \text{by def. of } \xrightarrow{c}_{\mathcal{C}} \\
\Longrightarrow & Q \overset{\tau}{\Rightarrow}_{\mathcal{C}} Q_1 \xrightarrow{c}_{\mathcal{C}} (Q_1|c) \overset{\tau}{\Rightarrow}_{\mathcal{C}} Q_2 \ \wedge\ (P|c) \approx_{\mathcal{DE}} Q_2 & \text{by def. of } \approx_{\mathcal{DE}} \\
\Longrightarrow & Q \xrightarrow{c}_{\mathcal{C}} (Q|c) \overset{\tau}{\Rightarrow}_{\mathcal{C}} (Q_1|c) \overset{\tau}{\Rightarrow}_{\mathcal{C}} Q_2 & \\
\Longrightarrow & P \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_1 \xrightarrow{c}_{\mathcal{C}} (P_1|c) \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_2 \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_3 \ \wedge\ (Q|c) \approx_{\mathcal{DE}} P_2 \ \wedge\ Q_2 \approx P_3 & \text{by def. of } \approx_{\mathcal{DE}} \\
\Longrightarrow & P \overset{\tau}{\Rightarrow}_{\mathcal{C}} (P|c) \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_2 \overset{\tau}{\Rightarrow}_{\mathcal{C}} P_3 \ \wedge\ P|c \approx_{\mathcal{DE}} P_3 & \text{by trans. of } \approx_{\mathcal{DE}} \\
\Longrightarrow & (P|c) \approx_{\mathcal{DE}} P_2 & \text{by case 3} \\
\Longrightarrow & (P|c) \approx (Q|c) & \text{by trans. of } \approx_{\mathcal{DE}}
\end{array}$$

case 5 We show that the following relation is a $\mathcal{DE}$-bisimulation up to $\approx_{\mathcal{DE}}$.

$$\{(\exists x P, \exists x Q) \mid P \approx_{\mathcal{DE}} Q\}$$

Let $\exists x P \xrightarrow{\alpha}_{\mathcal{C}} P'$. If α is an silent action or a output action with $x \notin \mathrm{ev}(\alpha)$, then the result is trivial.

If $\alpha = (x\mathbf{y})\overline{d}$, we can infer by Theorem B.3 that the proof tree of $\exists x P \xrightarrow{\alpha}_{\mathcal{C}} P'$ is of the form:

$$\begin{array}{cl}
c \vdash_{\mathcal{C}} \exists \mathbf{y}(d' \otimes e) \quad \exists \mathbf{y} d' \vdash_{\mathcal{C}} \exists \mathbf{y_2} d' & \\ \hline
\overline{c} \xrightarrow{(\mathbf{y_2})\overline{d}}_{\mathcal{C}} \overline{e} & \textit{($\mathcal{C}$-out)} \\ \hline
\overline{c}|P'' \xrightarrow{(\mathbf{y_2})\overline{d}}_{\mathcal{C}} \overline{e}|P'' & \textit{($\mathcal{C}$-comp$^{\equiv}$)} \\ \hline
\exists \mathbf{y}_1\,(\overline{c}|P'') \xrightarrow{(\mathbf{y_2})\overline{d}}_{\mathcal{C}} \overline{e}|P'' & \textit{($\mathcal{C}$-out$_{\mathrm{ex}}$)}^* \\ \hline
\exists \mathbf{z} x \mathbf{y}_1\,(\overline{e}|P'') \xrightarrow{(x\mathbf{y}_1\mathbf{y_2})\overline{d}}_{\mathcal{C}} \exists \mathbf{z}\,(\overline{e}|P'') & \textit{($\mathcal{C}$-rest)}^* \\ \hline
\exists x P \xrightarrow{(x\mathbf{y})\overline{d}}_{\mathcal{C}} P' & \textit{(cong)}
\end{array}$$

From this, it is easy to infer the tree:

$$\begin{array}{cl}
c \vdash_{\mathcal{C}} \exists \mathbf{y}(d' \otimes e) \quad \exists \mathbf{y} d' \vdash_{\mathcal{C}} \exists \mathbf{x}\mathbf{y_2} d' & \\ \hline
\overline{c} \xrightarrow{(x\mathbf{y_2})\overline{d}}_{\mathcal{C}} \overline{e} & \textit{($\mathcal{C}$-out)} \\ \hline
\overline{c}|P'' \xrightarrow{(x\mathbf{y_2})\overline{d}}_{\mathcal{C}} \overline{e}|P'' & \textit{($\mathcal{C}$-comp$^{\equiv}$)} \\ \hline
\exists \mathbf{y}_1\,(\overline{c}|P'') \xrightarrow{(x\mathbf{y_2})\overline{d}}_{\mathcal{C}} \overline{e}|P'' & \textit{($\mathcal{C}$-out$_{\mathrm{ex}}$)}^* \\ \hline
\exists \mathbf{z} \mathbf{y}_1\,(\overline{e}|P'') \xrightarrow{(\mathbf{y}_1 x\mathbf{y_2})\overline{d}}_{\mathcal{C}} \exists \mathbf{z}\,(\overline{e}|P'') & \textit{($\mathcal{C}$-rest)}^* \\ \hline
P \xrightarrow{(x\mathbf{y})\overline{d}}_{\mathcal{C}} P' & \textit{(cong)}
\end{array}$$

Hence, we get by hypotheses, the existence a process Q' such that $Q \xrightarrow{(x\mathbf{y})\overline{d}}_{\mathcal{C}} Q'$ i.e. $\exists x Q \xrightarrow{(x\mathbf{y})\overline{d}}_{\mathcal{C}} Q'$ with $P' \approx_{\mathcal{DE}} Q'$.

The case of input action is easily inferred thanks to case 1.

case 6. We show that the following relation is a $\mathcal{DE}$-bisimulation. By Theorem B.3, the proof of $\exists \mathbf{x}.(P|G) \xrightarrow{\tau}_{\mathcal{C}} P'$ is of the form:

$$\{(\exists \mathbf{x}(P|G), \exists \mathbf{x}(Q|G)) \mid P \approx_{\mathcal{DE}} Q\}$$

Let $P \xrightarrow{\alpha}_{\mathcal{C}} P'$. The only non trivial case is when $\alpha = \tau$ and the reduced ask is in G. By Theorem B.3, we get $P \xrightarrow{(\mathbf{x})\overline{d}[z\backslash \mathbf{t}]}_{\mathcal{C}} P''$, $G \xrightarrow{d[z\backslash \mathbf{t}]}_{\mathcal{C}} (G'|R[\mathbf{z}\backslash t])$ with $P' \equiv_{\mathcal{C}} \exists \mathbf{y}(P''|G)$ and $P' \equiv_{C} \exists \mathbf{x} \exists \mathbf{y}(P''|G'|R[\mathbf{z}\backslash t])$. By definition of $\approx_{\mathcal{DE}}$, we have $Q \xrightarrow{(\mathbf{x})\overline{c}}_{\mathcal{C}} Q''$ with $P'' \approx_{\mathcal{DE}} Q''$, that is $Q'' \exists \mathbf{x} \exists \mathbf{y}[Q''|G'|R[\mathbf{z}\backslash t])$ with $P' \equiv_{C} \exists \mathbf{x} \exists \mathbf{y}[Q''|G'|R[\mathbf{z}\backslash t])$. □

Proof of Theorem 3.7. From Theorem B.8, one can infer that $\mathcal{DE}$-bisimilarity is closed by evaluation $\mathcal{DE}$-context. □

We can related logical equivalence and bisimulation if the compared processes do contain neither asks nor choices.

Remark B.9. *Let P and Q two LCC-agent built form tell, parallel composition and scope restriction. $P \multimapboth_{\mathcal{C}} Q$ iff $P \approx_{\mathcal{CC}} Q$*

B.3.3 Proofs of Section 3.5. (Barbed congruence)

Proof of Theorem 3.9. We prove $\cong_{\mathcal{DE}} \subset \approx_{\mathcal{DE}}$ by establishing that $\cong_{\mathcal{DE}}$ is a $\mathcal{DE}$-bisimulation. Let $P \cong_{\mathcal{DE}} Q$ and $P \xrightarrow{\alpha}_{\mathcal{C}} P'$. The cases of silent or output action are direct. For input action just use the fact that $\cong_{\mathcal{DE}}$ is a $\mathcal{DE}$-congruence. For the other inclusion, is straightforward to prove that $\approx_{\mathcal{DE}}$ is a $\mathcal{DE}$-barbed bisimulation. By Theorem 3.7, we get that $\approx_{\mathcal{DE}}$ is a $\mathcal{DE}$-congruence. Hence we conclude that $\approx_{\mathcal{DE}}$ is barbed $\mathcal{DE}$-congruence. $\square$

B.4 Proofs of Section 4. (LCC a natural generalization of asynchronous calculi)

Proposition B.10 (Soundness)**.** *For all π-processes P and Q and any π-action α, we have:*

$$(i)\ \text{If } P \equiv_\pi Q \text{ then } [\![P]\!]_\pi \equiv_{\mathcal{C}_\pi} [\![Q]\!]_\pi. \qquad (ii)\ \text{If } P \xrightarrow{\alpha}_\pi Q \text{ then } [\![P]\!]_\pi \xrightarrow{[\![\alpha]\!]_\pi}_{\mathcal{C}_\pi} [\![Q]\!]_\pi.$$

Proof. *(i)* is immediate since each rule defining $\equiv_m$ has a counterpart in the definition of $\equiv_{\mathcal{C}_m}$. For *(ii)* we proceed by case on the transition $P \xrightarrow{\alpha}_\pi Q$. We focus only on the non trivial cases. For the rule (*cong*) we just use case *(i)*. For the rules *(π-comp)* and *(π-rest)*, we notice that $bv(\alpha) = bv([\![\alpha]\!]_\pi)$ and apply counterpart rules. For *(π-silent)* rule we notice that for any π-process R we have $[\![\tau.R]\!]_\pi = (\mathbf{1} \rightarrow [\![R]\!]_\pi) \equiv_{\mathcal{C}_m} (\mathbf{1} \rightarrow [\![R]\!]_\pi|\overline{\mathbf{1}}) \xrightarrow{\tau}_{\mathcal{C}_m} ([\![R]\!]_\pi|\overline{\mathbf{1}}) \equiv_{\mathcal{C}_m} [\![R]\!]_\pi$. For *($\pi$-sync)* we remark that for any π-process R we have $[\![x(z).R|\bar{x}y]\!]_\pi = (\gamma(x,y) \rightarrow [\![R]\!]_\pi|\overline{\gamma(x,z)}) \xrightarrow{\tau}_{\mathcal{C}_m} ([\![R]\!]_\pi[y\backslash z]|\overline{\mathbf{1}}) \equiv_{\mathcal{C}_m} [\![R[y\backslash z]]\!]_\pi$. $\square$

Proposition B.11 (Completeness)**.** *For any π-process P, any LCC process Q, any constraint c, and all variables $\mathbf{x}$, we have:*

(i) If $[\![P]\!]_\pi \equiv_{\mathcal{C}_\pi} \exists \mathbf{z}.(Q|c)$ then $P \equiv_\pi \nu\mathbf{z}(P'|\Pi_{i=1}^n \bar{x}_i y_i)$, $Q \equiv_{\mathcal{C}_\pi} [\![P']\!]_\pi$ and $c \dashv\vdash_{\mathcal{C}_\pi} \bigotimes_{i=1}^n \gamma(x_i, y_i)$

(ii) If $[\![P]\!]_\pi \xrightarrow{(\mathbf{z})\overline{c}}_{\mathcal{C}} Q$ then $P \equiv_\pi \nu\mathbf{z}'(P'|\Pi_{i=1}^n \bar{x}_i y_i)$, $Q \equiv_{\mathcal{C}_\pi} [\![P']\!]_\pi$, $\bigotimes_{i=1}^n \gamma(x_i, y_i) \vdash_{\mathcal{C}_\pi} c$

(iii) If $[\![P]\!]_\pi \xrightarrow{\tau}_{\mathcal{C}} Q'$ then $P \xrightarrow{\tau}_\pi Q$ with $[\![Q]\!]_\pi \equiv_{\mathcal{C}_\pi} Q'$.

Proof. (i) is by induction on P. (ii) are straightforward by case (i) and Theorem B.3.

For (iii), by case (i) and Theorem B.3, we infer

- $[\![P]\!]_\pi \equiv_{\mathcal{C}_\pi} \exists \mathbf{z}\,(\Pi_{i \in I}[\![G_i]\!]_\pi|\overline{c}|\forall \mathbf{x}(d \rightarrow [\![P']\!]_\pi))$
- $Q \equiv_{\mathcal{C}_\pi} \exists \mathbf{zy}\,(\Pi_{i \in I}[\![G_i]\!]_\pi|\overline{e}|[\![P']\!]_\pi[\mathbf{x}\backslash\mathbf{t}])$
- $c \dashv\vdash_{\mathcal{C}_\pi} \bigotimes_{j \in J} \gamma(x_j, y_j) \vdash_{\mathcal{C}_\pi} \exists \mathbf{y}\,(d[\mathbf{x}\backslash\mathbf{t}] \otimes e)$

There are two subcases:

- $\forall \mathbf{x}(d \rightarrow [\![P']\!]_\pi)$ is the translation of the silent prefix: We have $\mathbf{x} = \emptyset$ and $x = \mathbf{1}$, that is $P \equiv_m \nu\mathbf{z}\left(\Pi_{i=1}^m G_i|\Pi_{j=1}^n \bar{x}_j y_j|\tau.P'\right)$. Since $c \vdash_{\mathcal{C}_\pi} \exists y(\mathbf{1} \otimes e)$ is a most general choice, we infer $e \dashv\vdash_{\mathcal{C}_\pi} c$ and $\mathbf{y} = \emptyset$. It is straightforward to verify that $P \xrightarrow{\tau}_m Q' = \nu\mathbf{z}\left(\Pi_{i=1}^m G_i|\Pi_{j=1}^n \bar{x}_j y_j|P'\right)$ and $[\![Q']\!]_\pi \equiv_{\mathcal{C}_\pi} Q'$

- $\forall \mathbf{x}(d \rightarrow [\![P']\!]_\pi)$ is the translation of the input prefix: We have $\mathbf{x} = y'$, and $c = \gamma(x', y')$, that $P \equiv_m \nu\mathbf{z}\left(\Pi_{i=1}^m G_i | \Pi_{j=1}^n \bar{x}_j y_j | \bar{x}'(y').P'\right)$. We infer $(x', \mathbf{t}) = (x_j, y_j)$ for some $j \in 1, \ldots, n$; w.l.o.g. we choose $j = 1$. Since $c \vdash_{\mathcal{C}_\pi} \exists y(\gamma(x_1, y_1)[y' \backslash t] \otimes e)$ is a most general choice for, we infer $e \dashv\vdash_{\mathcal{C}_\pi} \Pi_{j=2}^n$ and $\mathbf{y} = \emptyset$. It is straightforward to verify that $P \xrightarrow{\tau}_m Q' = \nu\mathbf{z}\left(\Pi_{i=1}^m G_i | \Pi_{j=2}^n \bar{x}_j y_j | P'[y' \backslash y_j]\right)$ and $[\![Q']\!]_\pi \equiv_{\mathcal{C}_\pi} Q'$. □

Proof of Theorem 4.2. Direct by Theorems B.10 and B.11. □

We defined two relations, we will use in the following proofs:

$$\approx_{[\![\pi]\!]} \stackrel{def}{=} \{([\![P]\!]_\pi, [\![Q]\!]_\pi) \mid P \approx_\pi Q\}$$
$$\cong_{[\![\pi]\!]} \stackrel{def}{=} \{(P, Q) \mid P \approx_{\mathcal{C}_\pi} [\![P']\!]_\pi \wedge P' \cong_m Q' \wedge [\![Q']\!]_\pi \approx_{\mathcal{C}_\pi} Q\}$$

Let $\mathcal{D}_\pi = \{\exists y.\gamma(x, y) \mid x \in \mathcal{V} \setminus \{y\}\}$ and $\mathcal{E}_\pi = \cup\{\gamma(x, y) \mid xy \in \mathcal{V}\}$ and $\mathcal{D}_\pi^\star = \mathcal{D}_\pi \cup \mathcal{E}_\pi$. Form the observation that the co-domain of $[\![\ ]\!]_\pi$ is precisely the set of $\mathcal{D}_\pi\mathcal{E}_\pi$-processes, we deduce the following lemma.

Lemma B.12. *(i) If $\mathcal{R}$ is a congruence on π-processes then $\{([\![P']\!]_\pi, [\![Q]\!]_\pi) \mid P\mathcal{R}Q\}$ is a $\mathcal{D}_\pi\mathcal{E}_\pi$-congruence. (ii) If $\mathcal{R}'$ is a $\mathcal{D}_\pi\mathcal{E}_\pi$-congruence, then $\{(P, Q) \mid [\![P]\!]_\pi \mathcal{R}' [\![Q]\!]_\pi\}$ is a π-congruence.*

Proof of Theorem 4.3. We prove each case independently.

case *(i)* The "if" direction is straightforward by Theorem 4.2, case *(ii)* of Theorem B.12, and the fact that any $\mathcal{D}_\pi\mathcal{C}_\pi$-congruence is obviously a $\mathcal{D}_\pi\mathcal{E}_\pi$-congruence.

For the "only if" direction, by Theorem B.9, we infer, that for any $\mathcal{D}_\pi\mathcal{C}_\pi$-context $D[\ ]$, there exists a $\mathcal{D}_\pi\mathcal{C}_\pi$-context $D'[\ ]$ such that for any P, $D[P] \approx_{\mathcal{C}_\pi} D'[P]$, that is $\mathcal{O}^{\mathcal{D}_\pi}(D[P]) = \mathcal{O}^{\mathcal{D}_\pi}(D'[P])$. Hence it is sufficient to prove that if $P \simeq_\pi Q$ then $[\![P]\!]_\pi \simeq_{\mathcal{D}_\pi\mathcal{E}_\pi} [\![Q]\!]_\pi$. That can be infer straightforwardly form Theorem 4.2, case *(i)* of Theorem B.12.

case *(ii)* The "if" direction is corollary of Theorem 4.2. For the "only if" direction, it is sufficient to show that the relation $\approx_{[\![\pi]\!]}$ is a $\mathcal{D}_\pi^\star\mathcal{C}_\pi$-bisimulation up to $\equiv_{\mathcal{C}_\pi}$. Let assume that $[\![P]\!]_\pi \approx_{[\![\pi]\!]} [\![Q]\!]_\pi$ and $[\![P]\!]_\pi \xrightarrow{\alpha}_{\mathcal{C}_\pi} P'$. There are three cases according the type of the action α:

- (silent action) $\alpha = \tau$. Direct by case (iii) of Theorem B.11.
- (input action). $\alpha = c \in C_\pi$. By Theorem B.11, we infer $P' \equiv_{\mathcal{C}_\pi} [\![P]\!]_\pi | c$ with $c \dashv\vdash_{\mathcal{C}} \exists\mathbf{z}. \bigotimes_{i=1}^n \gamma(x_i, y_i)$. By Theorem B.9, Theorem 3.7, and Theorem B.11 (case (i)), we have $P' \equiv_{\mathcal{C}_\pi} [\![P | \nu\mathbf{z}\Pi_{i=1}^n \bar{x}_i y_i]\!]_\pi$. Since $\approx_\pi$ is stable by parallel composition (See the proof of Fournet and Gonthier (Fournet and Gonthier 2005)), we have $P | \nu\mathbf{z}\Pi_{i=1}^n \bar{x}_i y_i \approx_\pi Q | \nu\mathbf{z}\Pi_{i=1}^n \bar{x}_i y_i$. By Theorems 3.7, B.9 and B.10, (case (i)) we have $[\![P]\!]_\pi | c \approx_{[\![\pi]\!]} [\![Q]\!]_\pi | c$. To conclude, notice that obviously $[\![Q]\!]_\pi \xrightarrow{c}_{\mathcal{C}_\pi} [\![Q]\!]_\pi | c$.
- (output action). We have $\alpha = (z)\overline{\gamma(x, y)}$ with $x \neq z$. By Theorem B.11, we infer the existence of a π-process P'' such that $P \equiv_\pi \nu z(P'' | \bar{x}y)$, that is either $P \xrightarrow{\bar{x}y}_\pi \nu z(P'')$ (if $z \neq y$) or $P \xrightarrow{\bar{x}(y)}_\pi (P'')$ (if $z = y$). By definition of $\approx_\pi$ we infer the existence of Q' such that $Q \xrightarrow{\bar{x}y}_\pi Q'$ and $Q' \approx \exists z P''$ (if $z \neq y$) or $Q \xrightarrow{\bar{x}(y)}_\pi Q'$ and $Q' \approx P''$ (if $z = y$). The conclusion is then obvious by Theorem B.10.

case ***(iii)*** As for may-testing case, the "if" direction of the barbed congruence is direct by Theorem 4.2 and case *(ii)* of Theorem B.12. For the "only if" direction, it is sufficient to show that $\cong_{[\![\pi]\!]}$ is *(a)* a $\mathcal{D}_\pi\mathcal{C}_\pi$-barbed bisimulation and *(b)* a $\mathcal{D}_\pi\mathcal{C}_\pi$-congruence

(a) Let $P \approx_{\mathcal{C}_\pi} [\![P']\!]_\pi \wedge P' \cong_m Q' \wedge [\![Q']\!]_\pi \approx_{\mathcal{C}_\pi} Q$. We have to show that *(1)* if $P \xrightarrow{\tau}_{\mathcal{C}_\pi} P''$ then $Q \overset{\tau}{\Rightarrow}_{\mathcal{C}_\pi} Q''$ with $P'' \cong_{[\![\pi]\!]} Q''$, and *(2)* and $\mathcal{O}^{\mathcal{D}_\pi}(P) \subset \mathcal{O}^{\mathcal{D}_\pi}(Q)$. For *(1)*, notice that by $\approx_{\mathcal{C}_\pi}$ is more discriminative that $\approx_{\mathcal{D}_\pi\mathcal{C}_\pi}$ and use case *(ii)*, Theorem 4.2, and coinduction hypothesis. For *(2)*, let assume $\alpha \in \mathcal{O}^{\mathbb{A}}(P)$, that is $\alpha \in \mathcal{O}^{\mathcal{D}_m}([\![P']\!]_\pi)$. We have $\alpha = (y)\overline{\gamma}(x,y)$ for some distinct x, y. By Theorem B.11, we have either $P' \xrightarrow{\bar{x}y}_m P'''$, or $P \xrightarrow{\bar{x}(y)}_m P'''$ (for some π-process P'''), that is $P' \Downarrow_{\bar{x}}$, or by definition of $\cong_m$, $Q' \Downarrow_{\bar{x}}$. By Theorem B.11, we infer $[\![Q']\!]_\pi \xrightarrow{\overline{\gamma(x,y)}}_{\mathcal{C}_m} Q'''$ or $[\![Q']\!]_\pi \xrightarrow{(y)\overline{\gamma(x,y)}}_{\mathcal{C}_m} Q'''$ for some LCC-process Q''', that is, by definition of $\approx_{\mathcal{C}_\pi}$, $(\exists y.\gamma(x,y)) \in \mathcal{O}^{\mathcal{D}_\pi\mathcal{C}_\pi}(Q)$.

(b) Since $\cong_\pi$ is a by definition a π-congruence, we have, by Theorem B.12 that $\{([\![P]\!]_\pi, [\![Q]\!]_\pi) \mid P \cong_m Q\}$ is a $\mathcal{D}_\pi\mathcal{E}_\pi$-congruence. Since moreover $\approx_{\mathcal{C}_\pi}$ is clearly a $\mathcal{D}_\pi\mathcal{E}_\pi$-congruence, we infer $\cong_{[\![\pi]\!]}$ is a $\mathcal{D}_\pi\mathcal{E}_\pi$-congruence. By Theorem B.9 and Theorem 3.7, we conclude that $\cong_{[\![\pi]\!]}$ is a $\mathcal{D}_\pi\mathcal{C}_\pi$-congruence. □

B.5 Proofs of Section 5. (Observational equivalence relations for CC framework)

Proof of Theorem 5.4. We show the following relation is a $\mathcal{CC}$-bisimulation, the result follows by Theorem 3.9 up to $\equiv_{\mathcal{C}}$

$$\{((\sigma_1^\times|\mathcal{P}^\times), (\sigma_2^\times|\mathcal{P}^\times)) \mid \text{ there exist } \sigma_1' \text{ and } \sigma_2' \text{ such that } \sigma_1 \xrightarrow{\mathcal{P}}{}_a^* \sigma_1' \cong_{\mathcal{C}_c\mathcal{D}} \sigma_2' \xleftarrow{\mathcal{P}}{}_a^* \sigma_2\}$$

This is easily proved using the confluence of $\mathcal{P}$ and Theorem 5.2. □

Proof of Theorem 5.5. It is sufficient to show that the following relation is a $\mathcal{C}_c\mathcal{D}$-bisimulation up to $\equiv_{\mathcal{C}}$.

$$\{((\sigma^\times|\mathcal{P}^\times), (\sigma'^\times|\mathcal{P}'^\times)) \mid (\sigma^\times|\mathcal{P}^\times) \simeq_{\mathcal{C}_c\mathcal{D}} (\sigma'^\times|\mathcal{P}'^\times)$$

Let $(\sigma^\times|\mathcal{P}^\times) \xrightarrow{\alpha}_{\mathcal{C}} (\sigma_1^\times|\mathcal{P}^\times)$ and $((\sigma^\times|\mathcal{P}^\times), (\sigma'^\times|\mathcal{P}'^\times))$. The case of $alpha = \tau$ by Theorem 5.4. For the case $\alpha = c \in \mathcal{D}$, use the fact that $\simeq_{\mathcal{C}_c\mathcal{D}}$ is a $\mathcal{C}_c\mathcal{D}$-congruence. For the case of $\alpha = (\mathbf{x})\overline{c}$ with $(\exists \mathbf{x}c) \in \mathcal{C}$ use Theorem 5.2. □