Deriving Labels and Bisimilarity for Concurrent Constraint Programming*

Andrés Aristizábal[1], Filippo Bonchi[2], Catuscia Palamidessi[1], Luis Pino[1], and Frank Valencia[1]

[1] Comète, LIX, Laboratoire de l'École Polytechnique associé à l'INRIA
[2] CNRS - Laboratoire de l'Informatique du Parallélisme, ENS Lyon

Abstract. Concurrent constraint programming (ccp) is a well-established model for concurrency. Bisimilarity is one of the central reasoning techniques in concurrency. The standard definition of bisimilarity, however, is not completely satisfactory for ccp since it yields an equivalence that is too fine grained. By building upon recent foundational investigations, we introduce a labelled transition semantics and a novel notion of bisimilarity that is fully abstract w.r.t. the typical observational equivalence in ccp.

Introduction

Concurrency is concerned with systems of multiple computing agents, usually called *processes*, that interact with each other. *Process calculi* treat processes much like the λ-calculus treats computable functions. They provide a language in which processes are represented by terms, and computational steps are represented as transitions between them. These formalisms are equipped with equivalence relations that determine what processes are deemed indistinguishable. *Bisimilarity* is one of the main representative of these. It captures our intuitive notion of process equivalence; two processes are equivalent if they can match each other's moves. Furthermore, it provides an elegant co-inductive proof technique based on the notion of bisimulation.

Concurrent Constraint Programming (ccp) [26] is a well-established formalism that combines the traditional algebraic and operational view of process calculi with a declarative one based upon first-order logic. In ccp, processes interact by *adding* (or *telling*) and *asking* information (namely, constraints) in a medium (the *store*). Ccp is parametric in a *constraint system* indicating interdependencies (entailment) between constraints and providing for the specification of data types and other rich structures. The above features have recently attracted a renewed attention as witnessed by the works [23,9,5,4] on calculi exhibiting data-types, logic assertions as well as tell and ask operations.

There have been few attempts to define a notion of bisimilarity for ccp. The ones we are aware of are those in [26] and [19] upon which we build. These equivalences are not completely satisfactory: We shall see that the first one may tell apart processes with identical behaviour, while the second quantifies over all possible inputs from the environment, and hence it is not clear whether it can lead to a feasible proof technique.

* This work has been partially supported by the project ANR-09-BLAN-0169-01 PANDA and by the INRIA DRI Equipe Associée FORCES.

M. Hofmann (Ed.): FOSSACS 2011, LNCS 6604, pp. 138–152, 2011.

The goal of this paper is to define a notion of bisimilarity for ccp which will allow to benefit of the feasible proof and verification techniques typically associated with bisimilarity. Furthermore, we aim at studying the relationship between this equivalence and other existing semantic notions for ccp. In particular, its elegant denotational characterization based on closure operators [27] and the connection with logic [19].

Labels and Bisimilarity from Reductions. Bisimilarity relies on *labelled transitions*: each evolution step of a system is tagged by some information aimed at capturing the possible interactions of a process with the environment. Nowadays process calculi tend to adopt reduction semantics based on *unlabelled transitions* and *barbed congruence* [21]. The main drawback of this approach is that to verify barbed congruences it is often necessary to analyze the behaviour of processes under every context.

This scenario has motivated a novel stream of research [29,18,11,28,7,25,13,6] aimed at defining techniques for "deriving labels and bisimilarity" from unlabeled reduction semantics. The main intuition is that labels should represent the "minimal contexts allowing a process to reduce". The *theory of reactive systems* by Leifer and Milner [18] provides a formal characterization (by means of a categorical construction) of such "minimal contexts" and it focuses on the bisimilarity over transition systems labeled as: $P \xrightarrow{C} P'$ iff $C[P] \longrightarrow P'$ and C is the minimal context allowing such reduction.

In [7,6], it is argued that the above bisimilarity is often too fine grained and an alternative, coarser, notion of bisimilarity is provided. Intuitively, in the bisimulation game, each move (transition) $P \xrightarrow{C} P'$, has to be matched it with a move $C[Q] \longrightarrow Q'$.

Labels and Bisimilarity for ccp. The operational semantics of ccp is expressed by reductions between configurations of the form $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ meaning that the process P with store d may reduce to P' with store d'. From this semantics we shall derive a labeled transition system for ccp by exploiting the intuition of [29,18]. The transition $\langle P, d\rangle \xrightarrow{e} \langle P', d'\rangle$ means that e is a "minimal constraint" (from the environment) that needs to be added to d to reduce from $\langle P, d\rangle$ into $\langle P', d'\rangle$.

Similar ideas were already proposed in [26] but, the recent developments in [6] enlighten the way for obtaining a fully abstract equivalence. Indeed, the standard notion of bisimilarity defined on our labeled semantics can be seen as an instance of the one proposed in [18]. As for the bisimilarity in [26], it is too fine grained, i.e., it separates processes which are indistinguishable. Instead, the notion of bisimulation from [6] (instantiated to the case of ccp) is fully abstract with respect to the standard observational equivalence given in [27]. Our work can therefore be also regarded as a compelling application of the theory of reactive systems.

Contributions. We provide a labelled transition semantics and a novel notion of labelled bisimilarity for ccp by building upon the work in [26,6]. We also establish a strong correspondence with existing ccp notions by providing a fully-abstract characterization of a standard observable behaviour for *infinite* ccp processes: *The limits of fair computations*. From [27] this implies a fully-abstract correspondence with the closure operator denotational semantics of ccp. Therefore, this work provides ccp with a new co-inductive proof technique, coherent with the existing ones, for reasoning about process equivalence.

Missing proofs and additional examples are in [1].

1 Background

In this section we recall the syntax, the operational semantics and the observational equivalence of concurrent constraint programming (ccp). We begin with the notion of constraint system. We presuppose some basic knowledge of domain theory (see [2]).

1.1 Constraint Systems

The ccp model is parametric in a *constraint system* specifying the structure and interdependencies of the information that processes can ask and tell. Following [27,10], we regard a constraint system as a complete algebraic lattice structure in which the ordering $\sqsubseteq$ is the reverse of an entailment relation ($c \sqsubseteq d$ means that d contains "more information" than c, hence c can be derived from d). The top element $false$ represents inconsistency, the bottom element $true$ is the empty constraint, and the *least upper bound* (lub) $\sqcup$ represents the join of information.

Definition 1. *A* constraint system $\mathbf{C}$ *is a complete algebraic lattice* $(Con, Con_0, \sqsubseteq, \sqcup, true, false)$ *where* Con *(the set of constraints) is a partially ordered set w.r.t.* $\sqsubseteq$*,* Con_0 *is the subset of finite elements of* Con*,* $\sqcup$ *is the lub operation, and* $true$*,* $false$ *are the least and greatest elements of* Con*, respectively.*

Recall that $\mathbf{C}$ is a *complete lattice* iff every subset of Con has a least upper bound in Con. An element $c \in Con$ is *finite* iff for any directed subset D of Con, $c \sqsubseteq \bigsqcup D$ implies $c \sqsubseteq d$ for some $d \in D$. $\mathbf{C}$ is *algebraic* iff each element $c \in Con$ is the least upper bound of the finite elements below c.

In order to model *hiding* of local variables and *parameter passing*, in [27] the notion of constraint system is enriched with *cylindrification operators* and *diagonal elements*, concepts borrowed from the theory of cylindric algebras (see [14]).

Let us consider a (denumerable) set of variables Var with typical elements $x, y, z, \ldots$ Define $\exists_{Var}$ as the family of operators $\exists_{Var} = \{\exists_x \mid x \in Var\}$ (*cylindric operators*) and D_{Var} as the set $D_{Var} = \{d_{xy} \mid x, y \in Var\}$ (*diagonal elements*).

A *cylindric constraint system* over a set of variables Var is a constraint system whose support set $Con \supseteq D_{Var}$ is closed under the cylindric operators $\exists_{Var}$ and quotiented by Axioms $\mathrm{C}1 - \mathrm{C}4$, and whose ordering $\sqsubseteq$ satisfies Axioms $\mathrm{C}5 - \mathrm{C}7$:

$\mathbf{C1}$. $\exists_x \exists_y c = \exists_y \exists_x c$ $\quad$ $\mathbf{C2}$. $d_{xx} = true$

$\mathbf{C3}$. *if* $z \neq x, y$ *then* $d_{xy} = \exists_z (d_{xz} \sqcup d_{zy})$ $\mathbf{C4}$. $\exists_x (c \sqcup \exists_x d) = \exists_x c \sqcup \exists_x d$

$\mathbf{C5}$. $\exists_x c \sqsubseteq c$ $\mathbf{C6}$. *if* $c \sqsubseteq d$ *then* $\exists_x c \sqsubseteq \exists_x d$ $\mathbf{C7}$. *if* $x \neq y$ *then* $c \sqsubseteq d_{xy} \sqcup \exists_x (c \sqcup d_{xy})$

where c, c_i, d indicate finite constraints, and $\exists_x c \sqcup d$ stands for $(\exists_x c) \sqcup d$. For our purposes, it is enough to think the operator $\exists_x$ as *existential quantifier* and the constraint d_{xy} as the equality $x = y$.

We assume notions of *free variable* and of *substitution* that satisfy the following conditions, where $c[y/x]$ is the constraint obtained by substituting x by y in c and $fv(c)$ is the set of free variables of c: (1) if $y \notin fv(c)$ then $(c[y/x])[x/y] = c$; (2) $(c \sqcup d)[y/x] = c[y/x] \sqcup d[y/x]$; (3) $x \notin fv(c[y/x])$; (4) $fv(c \sqcup d) = fv(c) \cup fv(d)$.

We now define the cylindric constraint system that will be used in all the examples.

Example 1 (The $\mathcal{S}$ Constraint System). Let $S = (\omega + 1, 0, \infty, =, <, succ)$ be a first-order structure whose domain of interpretation is $\omega + 1 \stackrel{\text{def}}{=} \omega \cup \{\infty\}$, i.e., the natural numbers extended with a top element ∞. The constant symbols 0 and ∞ are interpreted as zero and infinity, respectively. The symbols $=$, $<$ and $succ$ are all binary predicates on $\omega + 1$. The symbol $=$ is interpreted as the identity relation. The symbol $<$ is interpreted as the set of pairs (n, m) s.t., $n \in \omega$, $m \in \omega + 1$ and n strictly smaller than m. The symbol $succ$ is interpreted as the set of pairs (n, m) s.t., $n, m \in \omega$ and $m = n + 1$.

Let Var be an infinite set of variables. Let $\mathcal{L}$ be the logic whose formulae ϕ are: $\phi ::= t \mid \phi_1 \wedge \phi_2 \mid \exists_x \phi$ and $t ::= e_1 = e_2 \mid e_1 < e_2 \mid succ(e_1, e_2)$ where e_1 and e_2 are either 0 or ∞ or variables in Var. Note that formulas like $x = n$ or $x < n$ (for $n = 1, 2, \ldots$) do not belong to $\mathcal{L}$. A useful abbreviation to express them is $succ^n(x, y) \stackrel{\text{def}}{=} \exists y_0 \ldots \exists y_n (\bigwedge_{0<i\leq n} succ(y_{i-1}, y_i) \wedge x = y_0 \wedge y = y_n)$. We use $x = n$ as shorthand for $succ^n(0, x)$ and $x < n$ as shorthand for $\exists_y (x < y \wedge y = n)$.

A variable assignment is a function $\mu : Var \longrightarrow \omega + 1$. We use $\mathcal{A}$ to denote the set of all assignments; $\mathcal{P}(X)$ to denote the powerset of a set X, $\emptyset$ the empty set and $\cap$ the intersection of sets. We use $\mathcal{M}(\phi)$ to denote the set of all assignments that *satisfy* the formula ϕ, where the definition of *satisfaction* is as expected.

We can now introduce a *constraint system* as follows: the set of constraints is $\mathcal{P}(\mathcal{A})$, and define $c \sqsubseteq d$ iff $c \supseteq d$. The constraint $false$ is $\emptyset$, while $true$ is $\mathcal{A}$. Given two constraints c and d, $c \sqcup d$ is the intersection $c \cap d$. By abusing the notation, we will often use a formula ϕ to denote the corresponding constraint, i.e., the set of all assignments satisfying ϕ. E.g. we use $1 < x \sqsubseteq 5 < x$ to mean $\mathcal{M}(1 < x) \sqsubseteq \mathcal{M}(5 < x)$.

From this structure, let us now define the *cylindric constraint system* $\mathcal{S}$ as follows. We say that an assignment μ' is *an x-variant* of μ if $\forall y \neq x$, $\mu(y) = \mu'(y)$. Given $x \in Var$ and $c \in \mathcal{P}(\mathcal{A})$, the constraint $\exists_x c$ is the set of assignments μ such that exists $\mu' \in c$ that is an x-variant of μ. The diagonal element d_{xy} is $x = y$. □

We make an assumption that will be pivotal in Section 3. Given a partial order $(C, \sqsubseteq)$, we say that c is strictly smaller than d (written $c \sqsubset d$) if $c \sqsubseteq d$ and $c \neq d$. We say that $(C, \sqsubseteq)$ is *well-founded* if there exists no infinite descending chains $\cdots \sqsubset c_n \sqsubset \cdots \sqsubset c_1 \sqsubset c_0$. For a set $A \subseteq C$, we say that an element $m \in A$ is *minimal* in A if for all $a \in A$, $a \not\sqsubset m$. We shall use $min(A)$ to denote the set of all minimal elements of A. Well-founded order and minimal elements are related by the following result.

Lemma 1. *Let $(C, \sqsubseteq)$ be a well-founded order and $A \subseteq C$. If $a \in A$, then $\exists m \in min(A)$ s.t., $m \sqsubseteq a$.*

In spite of its being a reasonable assumption, well-foundedness of $(Con, \sqsubseteq)$ is not usually required in the standard theory of ccp. We require it because the above lemma is fundamental for proving the completeness of labeled semantics (Lemma 5).

1.2 Syntax

Concurrent constraint programming (ccp) was proposed in [30] and then refined in [26,27]. We restrict ourselves to the summation-free fragment of ccp. The distinctive

confluent nature of this fragment is necessary for showing that our notion of bisimilarity coincides with the observational equivalence for infinite ccp processes given in [27].

Definition 2. *Assume a cylindric constraint system* $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ *over a set of variables* Var*. The ccp processes are given by the following syntax:*

$$P, Q \ldots ::= \mathbf{tell}(c) \mid \mathbf{ask}(c) \rightarrow P \mid P \parallel Q \mid \exists_x P \mid p(\boldsymbol{z})$$

where $c \in Con_0, x \in Var, \boldsymbol{z} \in Var^*$*. We use Proc to denote the set of all processes.*

Finite processes. Intuitively, the tell process $\mathbf{tell}(c)$ adds c to the global store. The addition is performed regardless the generation of inconsistent information. The ask process $\mathbf{ask}(c) \rightarrow P$ may execute P if c is entailed from the information in the store. The process $P \parallel Q$ stands for the *parallel execution* of P and Q; $\exists_x$ is a *hiding operator*, namely it indicates that in $\exists_x P$ the variable x is *local* to P. The occurrences of x in $\exists_x P$ are said to be bound. The bound variables of P, $bv(P)$, are those with a bound occurrence in P, and its free variables, $fv(P)$, are those with an unbound occurrence.

Infinite processes. To specify infinite behaviour, ccp provides parametric process definitions. A process $p(\boldsymbol{z})$ is said to be a *procedure call* with identifier p and actual parameters $\boldsymbol{z}$. We presuppose that for each procedure call $p(z_1 \ldots z_m)$ there exists a unique *procedure definition* possibly *recursive*, of the form $p(x_1 \ldots x_m) \stackrel{\text{def}}{=} P$ where $fv(P) \subseteq \{x_1, \ldots, x_m\}$. Furthermore we require recursion to be *guarded*: I.e., each procedure call within P must occur within an ask process. The behaviour of $p(z_1 \ldots z_m)$ is that of $P[z_1 \ldots z_m / x_1 \ldots x_m]$, i.e., P with each x_i replaced with z_i (applying α-conversion to avoid clashes). We shall use D to denote the set of all process definitions.

Although we have not defined yet the semantics of processes, we find it instructive to illustrate the above operators with the following example. Recall that we shall use $\mathcal{S}$ in Ex. 1 as the underlying constraint system in all examples.

Example 2. Consider the following (family of) process definitions.

$$up_n(x) \stackrel{\text{def}}{=} \exists_y(\mathbf{tell}(y = n) \parallel \mathbf{ask}\ (y = n)\ \rightarrow\ up(x, y))$$

$$up(x, y) \stackrel{\text{def}}{=} \exists_{y'}(\mathbf{tell}(y < x \wedge succ^2(y, y')) \parallel \mathbf{ask}(y < x \wedge succ^2(y, y')) \rightarrow up(x, y'))$$

Intuitively, $up_n(x)$, where n is a natural number, specifies that x should be greater than any natural number (i.e., $x = \infty$ since $x \in \omega + 1$) by telling (adding to the global store) the constraints $y_{i+1} = y_i + 2$ and $y_i < x$ for some $y_0, y_1, \ldots$ with $y_0 = n$. The process $up_0(x) \parallel \mathbf{ask}(42 < x) \rightarrow \mathbf{tell}(z = 0)$, can set $z = 0$ when it infers from the global store that $42 < x$. (This inference is only possible after the 22^{nd} call to *up*.) □

1.3 Reduction Semantics

To describe the evolution of processes, we extend the syntax by introducing a process $\mathbf{stop}$ representing successful termination, and a process $\exists_x^e P$ representing the evolution of a process of the form $\exists_x P$, where e is the local information (*local store*) produced during this evolution. The process $\exists_x P$ can be seen as a particular case of $\exists_x^e P$: it represents the situation in which the local store is empty. Namely, $\exists_x P = \exists_x^{true} P$.

Table 1. Reduction semantics for ccp (The symmetric Rule for R3 is omitted)

$$\text{R1}\quad \langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{stop}, d \sqcup c\rangle \qquad \text{R2}\quad \frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \longrightarrow \langle P, d\rangle}$$

$$\text{R3}\quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow \langle P' \parallel Q, d'\rangle} \qquad \text{R4}\quad \frac{\langle P, e \sqcup \exists_x d\rangle \longrightarrow \langle P', e' \sqcup \exists_x d\rangle}{\langle \exists^e_x P, d\rangle \longrightarrow \langle \exists^{e'}_x P', d \sqcup \exists_x e'\rangle}$$

$$\text{R5}\quad \frac{\langle P[\boldsymbol{z}/\boldsymbol{x}], d\rangle \longrightarrow \gamma'}{\langle p(\boldsymbol{z}), d\rangle \longrightarrow \gamma'} \quad \text{where } p(\boldsymbol{x}) \stackrel{\text{def}}{=} P \text{ is a process definition in } D$$

A configuration is a pair $\langle P, d\rangle$ representing the state of a system; d is a constraint representing the global store, and P is a process in the extended syntax. We use $Conf$ with typical elements $\gamma, \gamma', \ldots$ to denote the set of configurations. The operational model of ccp can be described formally in the SOS style by means of the transition relation between configurations $\longrightarrow \subseteq Conf \times Conf$ defined in Table 1.

Rules R1-R3 and R5 are easily seen to realize the above process intuitions. Rule R4 is somewhat more involved. Here, we show an instructive example of its use.

Example 3. We have the below reduction of $P = \exists^e_x(\mathbf{ask}\ (y > 1)\ \rightarrow\ Q)$ where the local store is $e = x < 1$, and the global store $d' = d \sqcup \alpha$ with $d = y > x$, $\alpha = x > 1$.

$$\begin{array}{ll} & \dfrac{(y > 1) \sqsubseteq e \sqcup \exists_x d'}{\langle \mathbf{ask}\ (y > 1)\ \rightarrow\ Q, e \sqcup \exists_x d'\rangle \longrightarrow \langle Q, e \sqcup \exists_x d'\rangle} \\ \text{R2} & \\ \text{R4} & \dfrac{}{\langle P, d'\rangle \longrightarrow \langle \exists^e_x Q, d' \sqcup \exists_x e\rangle} \end{array}$$

Note that the x in d' is hidden, by using existential quantification in the reduction obtained by Rule R2. This expresses that the x in d' is different from the one bound by the local process. Otherwise an inconsistency would be generated (i.e., $(e \sqcup d') = \mathit{false}$). Rule R2 applies since $(y > 1) \sqsubseteq e \sqcup \exists_x d'$. Note that the free x in $e \sqcup \exists_x d'$ is hidden in the global store to indicate that is different from the global x. □

1.4 Observational Equivalence

The notion of fairness is central to the definition of observational equivalence for ccp. To define fair computations, we introduce the notions of *enabled* and *active* processes, following [12]. Observe that any transition is generated either by a process $\mathbf{tell}(c)$ or by a process $\mathbf{ask}\ (c)\ \rightarrow\ Q$. We say that a process P is *active* in a transition $t = \gamma \longrightarrow \gamma'$ if it generates such transition; i.e if there exist a derivation of t where R1 or R2 are used

to produce a transition of the form $\langle P, d\rangle \longrightarrow \gamma''$. Moreover, we say that a process P is *enabled* in a configuration γ if there exists γ' such that P is active in $\gamma \longrightarrow \gamma'$.

Definition 3. *A computation* $\gamma_0 \longrightarrow \gamma_1 \longrightarrow \gamma_2 \longrightarrow \ldots$ *is said to be* fair *if for each process enabled in some* γ_i *there exists* $j \geq i$ *such that the process is active in* γ_j.

Note that a finite fair computation is guaranteed to be *maximal*, namely no outgoing transitions are possible from its last configuration.

The standard notion of observables for ccp are the *results* computed by a process for a given initial store. The result of a computation is defined as the least upper bound of all the stores occurring in the computation, which, due to the monotonic properties of ccp, form an increasing chain. More formally, given a finite or infinite computation ξ of the form $\langle Q_0, d_0\rangle \longrightarrow \langle Q_1, d_1\rangle \longrightarrow \langle Q_2, d_2\rangle \longrightarrow \ldots$ the result of ξ, denoted by $Result(\xi)$, is the constraint $\bigsqcup_i d_i$. Note that for a finite computation the result coincides with the store of the last configuration.

The following theorem from [27] states that all the fair computations of a configuration have the same result (due to fact that summation-free ccp is confluent).

Theorem 1 (from [27]). *Let* γ *be a configuration and let* ξ_1 *and* ξ_2 *be two computations of* γ. *If* ξ_1 *and* ξ_2 *are fair, then* $Result(\xi_1) = Result(\xi_2)$.

This allows us to set $Result(\gamma) \stackrel{\texttt{def}}{=} Result(\xi)$ for any fair computation ξ of γ.

Definition 4. (Observational equivalence) *Let* $\mathcal{O} : Proc \rightarrow Con_0 \rightarrow Con$ *be given by* $\mathcal{O}(P)(d) = Result(\langle P, d\rangle)$. *We say that* P *and* Q *are observational equivalent, written* $P \sim_o Q$, *iff* $\mathcal{O}(P) = \mathcal{O}(Q)$.

Example 4. Consider the processes $P = up_0(x) \parallel up_1(y)$ and $Q = \exists_z(\mathbf{tell}(z = 0) \parallel \mathbf{ask}(z = 0) \rightarrow fairup(x, y, z))$ with up_0 and up_1 as in Ex. 2 and $fairup(x, y, z) \stackrel{\texttt{def}}{=}$

$$\exists_{z'}(\mathbf{tell}(z < x \wedge succ(z, z')) \parallel \mathbf{ask}\ ((z < x) \wedge succ(z, z')) \rightarrow fairup(y, x, z')))$$

Let $s(\gamma)$ denote the store in the configuration γ. For every infinite computation $\xi : \langle P, true\rangle = \gamma_0 \longrightarrow \gamma_1 \longrightarrow \ldots$ with $(1 < y) \not\sqsubseteq s(\gamma_i)$ for each $i \geq 0$, ξ is not fair and $Result(\xi) = (x = \infty)$. In contrast, every infinite computation $\xi : \langle Q, true\rangle = \gamma_0 \longrightarrow \gamma_1 \longrightarrow \ldots$ is fair and $Result(\xi) = (x = \infty \wedge y = \infty)$. Nevertheless, under our fair observations, P and Q are indistinguishable, i.e., $\mathcal{O}(P) = \mathcal{O}(Q)$. □

2 Saturated Bisimilarity for ccp

We introduce a notion of bisimilarity in terms of (unlabelled) reductions and *barbs* and we prove that this equivalence is fully abstract w.r.t. observational equivalence.

2.1 Saturated Barbed Bisimilarity

Barbed equivalences have been introduced in [21] for CCS, and have become the standard behavioural equivalences for formalisms equipped with unlabeled reduction semantics. Intuitively, *barbs* are basic observations (predicates) on the states of a system.

The choice of the "right" barbs is a crucial step in the barbed approach, and it is usually not a trivial task. For example, in synchronous languages like CCS or π-calculus both the inputs and the outputs are considered as barbs, (see e.g. [21,20]), while in the asynchronous variants only the outputs (see e.g. [3]). Even several works (e.g. [24,15]) have proposed abstract criteria for defining "good" barbs.

We shall take as barbs all the finite constraints in Con_0. This choice allows us to introduce a barbed equivalence (Def. 7) that coincides with the standard observational equivalence (Def. 4). It is worth to note that in $\sim_o$, the observables are all the constraints in Con and not just the finite ones.

We say that $\gamma = \langle P, d\rangle$ *satisfies* the barb c, written $\gamma \downarrow_c$, iff $c \sqsubseteq d$; γ *weakly satisfies* the barb c, written $\gamma \Downarrow_c$, iff $\gamma \longrightarrow^* \gamma'$ and $\gamma' \downarrow_c$[1].

Definition 5. *(Barbed bisimilarity) A barbed bisimulation is a symmetric relation $\mathcal{R}$ on configurations such that whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$:*

(i) if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$,
(ii) if $\gamma_1 \longrightarrow \gamma_1'$ then there exists γ_2' such that $\gamma_2 \longrightarrow \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$.

We say that γ_1 and γ_2 are barbed bisimilar, written $\gamma_1 \dot{\sim}_b \gamma_2$, if there exists a barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$. We write $P \dot{\sim}_b Q$ iff $\langle P, true\rangle \dot{\sim}_b \langle Q, true\rangle$.

Congruence characterization. One can verify that $\dot{\sim}_b$ is an equivalence. However, it is not a *congruence*; i.e., it is not preserved under arbitrary contexts. A context C is a term with a hole $[-]$ s.t., replacing it with a process P yields a process term $C[P]$. E.g., $C = \mathbf{tell}(c) \parallel [-]$ and $C[\mathbf{tell}(d)] = \mathbf{tell}(c) \parallel \mathbf{tell}(d)$.

Example 5. Let us consider the context $C = \mathbf{tell}(a) \parallel [-]$ and the processes $P = \mathbf{ask}\ (b) \rightarrow \mathbf{tell}(d)$ and $Q = \mathbf{ask}\ (c) \rightarrow \mathbf{tell}(d)$ with $a, b, c, d \neq true, b \sqsubseteq a$ and $c \not\sqsubseteq a$. We have $\langle P, true\rangle \dot{\sim}_b \langle Q, true\rangle$ because both configurations cannot move and they only satisfy the barb $true$. But $\langle C[P], true\rangle \not\dot{\sim}_b \langle C[Q], true\rangle$, because the former can perform three transitions (in sequence), while the latter only one. □

An elegant solution to modify bisimilarity for obtaining a congruence has been introduced in [22] for the case of weak bisimilarity in CCS. This work has inspired the introduction of *saturated bisimilarity* [7] (and its extension to the barbed approach [6]). The basic idea is simple: saturated bisimulations are closed w.r.t. all the possible contexts of the language. In the case of ccp, it is enough to require that bisimulations are *upward closed* as in condition (iii) below.

Definition 6. *(Saturated barbed bisimilarity). A saturated barbed bisimulation is a symmetric relation $\mathcal{R}$ on configurations such that whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$ with $\gamma_1 = \langle P, d\rangle$ and $\gamma_2 = \langle Q, e\rangle$:*

(i) if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$,
(ii) if $\gamma_1 \longrightarrow \gamma_1'$ then there exists γ_2' such that $\gamma_2 \longrightarrow \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$,
(iii) for every $a \in Con_0$, $(\langle P, d \sqcup a\rangle, \langle Q, e \sqcup a\rangle) \in \mathcal{R}$.

We say that γ_1 and γ_2 are saturated barbed bisimilar, written $\gamma_1 \dot{\sim}_{sb} \gamma_2$, if there exists a saturated barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$. We write $P \dot{\sim}_{sb} Q$ iff $\langle P, true\rangle \dot{\sim}_{sb} \langle Q, true\rangle$.

[1] As usual, $\longrightarrow^*$ denotes the reflexive and transitive closure of $\longrightarrow$.

Definition 7. *(Weak saturated barbed bisimilarity).* Weak saturated barbed bisimilarity *($\dot{\approx}_{sb}$) is obtained from Def. 6 by replacing $\longrightarrow$ with $\longrightarrow^*$ and $\downarrow_c$ with $\Downarrow_c$.*

Since $\dot{\sim}_{sb}$ is itself a saturated barbed bisimulation, it is obvious that it is upward closed. This fact also guarantees that it is a congruence w.r.t. all the contexts of ccp: a context C can modify the behaviour of a configuration γ only by adding constraints to its store. The same holds for $\dot{\approx}_{sb}$.

2.2 Correspondence with Observational Equivalence

We now show that $\dot{\approx}_{sb}$ coincides with the observational equivalence $\sim_o$. From [27] it follows that $\dot{\approx}_{sb}$ coincides with the standard denotational semantics for ccp.

First, we recall some basic facts from domain theory central to our proof. Two (possibly infinite) chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ and $e_0 \sqsubseteq e_1 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$ are said to be *cofinal* if for all d_i there exists an e_j such that $d_i \sqsubseteq e_j$ and, viceversa, for all e_i there exists a d_j such that $e_i \sqsubseteq d_j$.

Lemma 2. *Let $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ and $e_0 \sqsubseteq e_1 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$ be two chains. (1) If they are cofinal, then they have the same limit, i.e., $\bigsqcup d_i = \bigsqcup e_i$. (2) If the elements of the chains are* finite *and $\bigsqcup d_i = \bigsqcup e_i$, then the two chains are cofinal.*

In the proof, we will show that the stores of any pairs of *fair* computations of equivalent processes form pairs of cofinal chains. First, the following result relates weak barbs and fair computations.

Lemma 3. *Let $\langle P_0, d_0\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \ldots$ be a (possibly infinite) fair computation. If $\langle P_0, d_0\rangle \Downarrow_c$ then there exist a store d_i (in the above computation) such that $c \sqsubseteq d_i$.*

Theorem 2. *$P \sim_o Q$ if and only if $P \dot{\approx}_{sb} Q$.*

Proof. The proof proceeds as follows:

- *From $\dot{\approx}_{sb}$ to $\sim_o$.* Suppose that $\langle P, true\rangle \;\dot{\approx}_{sb}\; \langle Q, true\rangle$ and take a finite input $b \in Con_0$. Let

$$\langle P, b\rangle \longrightarrow \langle P_0, d_0\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \ldots$$

$$\langle Q, b\rangle \longrightarrow \langle Q_0, e_0\rangle \longrightarrow \langle Q_1, e_1\rangle \longrightarrow \ldots \longrightarrow \langle Q_n, e_n\rangle \longrightarrow \ldots$$

  be two fair computations. Since $\dot{\approx}_{sb}$ is upward closed, $\langle P, b\rangle \;\dot{\approx}_{sb}\; \langle Q, b\rangle$ and thus, for all d_i, $\langle Q, b\rangle \Downarrow_{d_i}$. By Lemma 3, it follows that there exists an e_j (in the above computation) such that $d_i \sqsubseteq e_j$. Analogously, for all e_i there exists a d_j such that $e_i \sqsubseteq d_j$. Then the two chains are cofinal and by Lemma 2.1, it holds that $\bigsqcup d_i = \bigsqcup e_i$, that means $\mathcal{O}(P)(b) = \mathcal{O}(Q)(b)$.
- From $\sim_o$ to $\dot{\approx}_{sb}$. Suppose that $P \sim_o Q$. We first show that for all $b \in Con_0$, $\langle P, b\rangle$ and $\langle Q, b\rangle$ satisfy the same weak barbs. Let

$$\langle P, b\rangle \longrightarrow \langle P_0, d_0\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \ldots$$

$$\langle Q, b\rangle \longrightarrow \langle Q_0, e_0\rangle \longrightarrow \langle Q_1, e_1\rangle \longrightarrow \ldots \longrightarrow \langle Q_n, e_n\rangle \longrightarrow \ldots$$

  be two (possibly infinite) fair computations. Since $P \sim_o Q$, then $\bigsqcup d_i = \bigsqcup e_i$. Since all the stores of computations are finite constraints (only finite constraints

can be added to the store), then by Lemma 2.2, it holds that for all d_i there exists an e_j such that $d_i \sqsubseteq e_j$. Now suppose that $\langle P, b\rangle \Downarrow_c$. By Lemma 3, it holds that there exists a d_i (in the above computation) such that $c \sqsubseteq d_i$. Thus $c \sqsubseteq d_i \sqsubseteq e_j$ that means $\langle Q, b\rangle \Downarrow_c$.

With this observation it is easy to prove that

$$\mathcal{R} = \{(\gamma_1, \gamma_2) \mid \exists b \text{ s.t. } \langle P, b\rangle \longrightarrow^* \gamma_1,\ \langle Q, b\rangle \longrightarrow^* \gamma_2\}$$

is a weak saturated barbed bisimulation (Def. 7). Take $(\gamma_1, \gamma_2) \in \mathcal{R}$.

If $\gamma_1 \Downarrow_c$ then $\langle P, b\rangle \Downarrow_c$ and, by the above observation, $\langle Q, b\rangle \Downarrow_c$. Since ccp is confluent, also $\gamma_2 \Downarrow_c$.

The fact that $\mathcal{R}$ is closed under $\longrightarrow^*$ is evident from the definition of $\mathcal{R}$. While for proving that $\mathcal{R}$ is upward-closed take $\gamma_1 = \langle P', d'\rangle$ and $\gamma_2 = \langle Q', e'\rangle$. It is easy to see that for all $a \in Con_0$, $\langle P, b \sqcup a\rangle \longrightarrow^* \langle P', d' \sqcup a\rangle$ and $\langle Q, b \sqcup a\rangle \longrightarrow^* \langle Q', e' \sqcup a\rangle$. Thus, by definition of $\mathcal{R}$, $(\langle P', d' \sqcup a\rangle, \langle Q', e' \sqcup a\rangle) \in \mathcal{R}$. □

3 Labeled Semantics

Although $\dot{\sim}_{sb}$ is fully abstract, it is at some extent unsatisfactory because of the upward-closure (namely, the quantification over all possible $a \in Con_0$ in condition (iii)) of Def. 6. We shall deal with this by refining the notion of transition by adding to it a label that carries additional information about the constraints that cause the reduction.

Labelled Transitions. Intuitively, we will use transitions of the form

$$\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$$

where label α represents a *minimal* information (from the environment) that needs to be added to the store d to evolve from $\langle P, d\rangle$ into $\langle P', d'\rangle$, i.e., $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$. From a more abstract perspective, our labeled semantic accords with the proposal of [29,18] of looking at "labels as the minimal contexts allowing a reduction". In our setting we take as contexts only the constraints that can be added to the store.

The Rules. The labelled transition $\longrightarrow \subseteq Conf \times Con_0 \times Conf$ is defined by the rules in Table 3. We shall only explain rules LR2 and LR4 as the other rules are easily seen to realize the above intuition and follow closely the corresponding ones in Table 1.

The rule LR2 says that $\langle \mathbf{ask}\ (c) \rightarrow P, d\rangle$ can evolve to $\langle P, d \sqcup \alpha\rangle$ if the environment provides a minimal constraint α that added to the store d entails c, i.e., $\alpha \in \min\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}$. Note that assuming that $(Con, \sqsubseteq)$ is well-founded (Sec. 1.1) is necessary to guarantee that α exists whenever $\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}$ is not empty.

To give an intuition about LR4, it may be convenient to first explain why a naive adaptation of the analogous reduction rule R4 in Table 1 would not work. One may be tempted to define the rule for the local case, by analogy to the labelled local rules in other process calculi (e.g., the π-calculus) and R4, as follows:

$$(*)\ \frac{\langle P, e \sqcup \exists_x d\rangle \xrightarrow{\alpha} \langle Q, e' \sqcup \exists_x d\rangle}{\langle \exists_x^e P, d\rangle \xrightarrow{\alpha} \langle \exists_x^{e'} Q, d \sqcup \exists_x e'\rangle} \quad \text{where } x \notin fv(\alpha)$$

Table 2. Labelled Transitions (The symmetric Rule for LR3 is omitted)

$$\text{LR1}\quad \langle \mathbf{tell}(c), d\rangle \xrightarrow{true} \langle \mathbf{stop}, d \sqcup c\rangle$$

$$\text{LR2}\quad \frac{\alpha \in \min\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \xrightarrow{\alpha} \langle P, d \sqcup \alpha\rangle} \qquad \text{LR3}\quad \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \xrightarrow{\alpha} \langle P' \parallel Q, d'\rangle}$$

$$\text{LR4}\ \frac{\langle P[z/x], e[z/x] \sqcup d\rangle \xrightarrow{\alpha} \langle P', e' \sqcup d \sqcup \alpha\rangle}{\langle \exists^{e}_{x} P, d\rangle \xrightarrow{\alpha} \langle \exists_{x}^{e'[x/z]} P'[x/z], \exists_x(e'[x/z]) \sqcup d \sqcup \alpha\rangle}\ x \notin fv(e'), z \notin fv(P) \cup fv(e \sqcup d \sqcup \alpha)$$

$$\text{LR5}\quad \frac{\langle P[\boldsymbol{z}/\boldsymbol{x}], d\rangle \xrightarrow{\alpha} \gamma'}{\langle p(\boldsymbol{z}), d\rangle \xrightarrow{\alpha} \gamma'}\quad \text{where } p(\boldsymbol{x}) \stackrel{\text{def}}{=} P \text{ is a process definition in } D$$

This rule however is not "complete" (in the sense of Lemma 5 below) as it does not derive all the transitions we wish to have.

Example 6. Let P as in Ex. 3, i.e., $P = \exists_x^{x<1}(\mathbf{ask}\ (y > 1)\ \rightarrow\ Q)$ and $d = y > x$. Note that $\alpha = x > 1$ is a minimal constraint that added to d enables a reduction from P. In Ex. 3 we obtained the transition: $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle \exists_x^{x<1} Q, d \sqcup \alpha \sqcup \exists_x(x < 1)\rangle$ Thus, we would like to have a transition from $\langle P, d\rangle$ labelled with α. But such a transition cannot be derived with Rule (*) above since $x \in fv(\alpha)$. □

Now, besides the side condition, another related problem with Rule (*) arises from the existential quantification $\exists_x d$ in the antecedent transition $\langle P, e \sqcup \exists_x d\rangle \xrightarrow{\alpha} \langle Q, e' \sqcup \exists_x d\rangle$. This quantification hides the effect of d on x and thus is not possible to identify the x in α with the x in d. The information from the environment α needs to be added to the global store d, hence the occurrences of x in both d and α must be identified. Notice that dropping the existential quantification of x in d in the antecedent transition does identify the occurrences of x in d with those in α but also with those in the local store e thus possibly generating variable clashes.

The rule LR4 in Table 2 solves the above-mentioned issues by using in the antecedent derivation a fresh variable z that acts as a substitute for the free occurrences of x in P and its local store e. (Recall that $T[z/x]$ represents T with x replaced with z). This way we identify with z the free occurrences of x in P and e and avoid clashes with those in α and d. E.g., for the process defined in the Ex.6, using LR4 (and LR2) one can derive

$$\frac{\langle \mathbf{ask}\ (y > 1)\ \rightarrow\ Q[z/x], z < 1 \sqcup y > x\rangle \xrightarrow{x>1} \langle Q[z/x], z < 1 \sqcup y > x \sqcup x > 1\rangle}{\langle \exists_x^{x<1}(\mathbf{ask}\ (y > 1)\ \rightarrow\ Q), y > x\rangle \xrightarrow{x>1} \langle \exists_x^{x<1} Q, \exists_x(x < 1) \sqcup y > x \sqcup x > 1\rangle}$$

The labeled semantics is *sound* and *complete* w.r.t. the unlabeled one. Soundness states that $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$ corresponds to our intuition that if α is added to d, P can

reach $\langle P', d' \rangle$. Completeness states that if we add a to (the store in) $\langle P, d \rangle$ and reduce to $\langle P', d' \rangle$, it exists a minimal information $\alpha \sqsubseteq a$ such that $\langle P, d \rangle \xrightarrow{\alpha} \langle P', d'' \rangle$ with $d'' \sqsubseteq d'$.

Lemma 4. *(Soundness). If* $\langle P, d \rangle \xrightarrow{\alpha} \langle P', d' \rangle$ *then* $\langle P, d \sqcup \alpha \rangle \longrightarrow \langle P', d' \rangle$.

Lemma 5. *(Completeness). If* $\langle P, d \sqcup a \rangle \longrightarrow \langle P', d' \rangle$ *then* $\exists \alpha, b$ *s.t.* $\langle P, d \rangle \xrightarrow{\alpha} \langle P', d'' \rangle$ *and* $\alpha \sqcup b = a$, $d'' \sqcup b = d'$.

Corollary 1. $\langle P, d \rangle \xrightarrow{true} \langle P', d' \rangle$ *if and only if* $\langle P, d \rangle \longrightarrow \langle P', d' \rangle$.

By virtue of the above, we will write $\longrightarrow$ to mean $\xrightarrow{true}$.

4 Strong and Weak Bisimilarity

Having defined our labelled transitions for ccp, we now proceed to define an equivalence that characterizes $\dot{\sim}_{sb}$ without the upward closure condition.

When defining bisimilarity over a labeled transition system, barbs are not usually needed because they can be somehow inferred by the labels of the transitions. For example in CCS, $P \downarrow_a$ iff $P \xrightarrow{a}$. The case of ccp is different: barbs cannot be removed from the definition of bisimilarity because they cannot be inferred by the transitions. In order to remove barbs from ccp, we could have inserted labels showing the store of processes (as in [26]) but this would have betrayed the philosophy of "labels as minimal constraints". Then, we have to define bisimilarity as follows.

Definition 8. *(Syntactic bisimilarity). A syntactic bisimulation is a symmetric relation* $\mathcal{R}$ *on configurations such that whenever* $(\gamma_1, \gamma_2) \in \mathcal{R}$*:*

(i) if $\gamma_1 \downarrow_c$ *then* $\gamma_2 \downarrow_c$,
(ii) if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ *then* $\exists \gamma_2'$ *such that* $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ *and* $(\gamma_1', \gamma_2') \in \mathcal{R}$.

We say that γ_1 *and* γ_2 *are syntactically bisimilar, written* $\gamma_1 \sim_S \gamma_2$*, if there exists a syntactic bisimulation* $\mathcal{R}$ *such that* $(\gamma_1, \gamma_2) \in \mathcal{R}$.

We called the above bisimilarity "syntactic", because it does not take into account the "real meaning" of the labels. This equivalence coincides with the one in [26] (apart from the fact that in the latter, barbs are implicitly observed by the transitions) and, from a more general point of view can be seen as an instance of bisimilarity in [18] (by identifying contexts with constraints). In [7], it is argued that the equivalence in [18] is often over-discriminating. This is also the case of ccp, as illustrated in the following.

Example 7. Let $P = \mathbf{ask}\ (x < 10) \rightarrow \mathbf{tell}(y = 0)$ and $Q = \mathbf{ask}\ (x < 5) \rightarrow \mathbf{tell}(y = 0)$. The configurations $\gamma_1 = \langle P \parallel Q, true \rangle$ and $\gamma_2 = \langle P \parallel P, true \rangle$ are not equivalent according to $\sim_S$. Indeed $\gamma_1 \xrightarrow{x<10} \gamma_1' \xrightarrow{x<5} \gamma_1''$, while γ_2 after performing $\gamma_2 \xrightarrow{x<10} \gamma_2'$ can only perform $\gamma_2' \xrightarrow{true} \gamma_2''$. However $\gamma_1 \dot{\sim}_{sb} \gamma_2$. □

To obtain coarser equivalence (coinciding with $\dot{\sim}_{sb}$), we define the following.

Definition 9. *(Strong bisimilarity). A strong bisimulation is a symmetric relation* $\mathcal{R}$ *on configurations such that whenever* $(\gamma_1, \gamma_2) \in \mathcal{R}$ *with* $\gamma_1 = \langle P, d \rangle$ *and* $\gamma_2 = \langle Q, e \rangle$:

(i) if $\gamma_1 \downarrow_c$ *then* $\gamma_2 \downarrow_c$,
(ii) if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ *then* $\exists \gamma_2'$ *s.t.* $\langle Q, e \sqcup \alpha \rangle \longrightarrow \gamma_2'$ *and* $(\gamma_1', \gamma_2') \in \mathcal{R}$.

We say that γ_1 *and* γ_2 *are strongly bisimilar, written* $\gamma_1 \dot{\sim} \gamma_2$, *if there exists a strong bisimulation* $\mathcal{R}$ *such that* $(\gamma_1, \gamma_2) \in \mathcal{R}$.

To give some intuition about the above definition, let us recall that in $\langle P, d \rangle \xrightarrow{\alpha} \gamma'$ the label α represents *minimal* information from the environment that needs to be added to the store d to evolve from $\langle P, d \rangle$ into γ'. We do not require the transitions from $\langle Q, e \rangle$ to match α. Instead (ii) requires something weaker: If α is added to the store e, it should be possible to reduce into some γ'' that it is in bisimulation with γ'. This condition is weaker because α may not be a minimal information allowing a transition from $\langle Q, e \rangle$ into a γ'' in the bisimulation, as shown in the previous example.

Definition 10. *(Weak bisimilarity). A weak bisimulation is a symmetric relation* $\mathcal{R}$ *on configurations such that whenever* $(\gamma_1, \gamma_2) \in \mathcal{R}$ *with* $\gamma_1 = \langle P, d \rangle$ *and* $\gamma_2 = \langle Q, e \rangle$:

(i) if $\gamma_1 \downarrow_c$ *then* $\gamma_2 \Downarrow_c$,
(ii) if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ *then* $\exists \gamma_2'$ *s.t.* $\langle Q, e \sqcup \alpha \rangle \longrightarrow^* \gamma_2'$ *and* $(\gamma_1', \gamma_2') \in \mathcal{R}$.

We say that γ_1 *and* γ_2 *are weakly bisimilar, written* $\gamma_1 \dot{\approx} \gamma_2$, *if there exists a weak bisimulation* $\mathcal{R}$ *such that* $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Example 8. We can show that $\mathbf{tell}(true) \dot{\approx} \mathbf{ask}(c) \rightarrow \mathbf{tell}(d)$ when $d \sqsubseteq c$. Intuitively, this corresponds to the fact that the implication $c \Rightarrow d$ is equivalent to $true$ when c entails d. Let us take $\gamma_1 = \langle \mathbf{tell}(true), true \rangle$ and $\gamma_2 = \langle \mathbf{ask}(c) \rightarrow \mathbf{tell}(d), true \rangle$. Their labeled transition systems are the following: $\gamma_1 \xrightarrow{true} \langle \mathbf{stop}, true \rangle$ and $\gamma_2 \xrightarrow{c} \langle \mathbf{tell}(d), c \rangle \xrightarrow{true} \langle \mathbf{stop}, c \rangle$. It is now easy to see that the symmetric closure of the relation $\mathcal{R}$ given below is a weak bisimulation.

$\mathcal{R} = \{(\gamma_2, \gamma_1), (\gamma_2, \langle \mathbf{stop}, true \rangle), (\langle \mathbf{tell}(d), c \rangle, \langle \mathbf{stop}, c \rangle), (\langle \mathbf{stop}, c \rangle, \langle \mathbf{stop}, c \rangle)\}$ □

The following theorem states that strong and weak bisimilarity coincide, resp., with $\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$. Hence γ_1 and γ_2 in the above example are also in $\dot{\approx}_{sb}$ (and, by Thm 2, also in $\sim_o$). It is worth noticing that any saturated barbed bisimulation (Def. 7) relating γ_1 and γ_2 is infinite in dimension, since it has to relate $\langle \mathbf{tell}(true), a \rangle$ and $\langle \mathbf{ask}(c) \rightarrow \mathbf{tell}(d), a \rangle$ for all constraints $a \in Con_0$. Instead, the relation $\mathcal{R}$ above is finite and it represents (by virtue of the following theorem) a proof also for $\gamma_1 \dot{\approx}_{sb} \gamma_2$.

Theorem 3. $\dot{\sim}_{sb} = \dot{\sim}$ *and* $\dot{\approx}_{sb} = \dot{\approx}$.

5 Conclusions, Related and Future Work

In this paper we introduced labeled semantics and bisimilarity for ccp. Our equivalence characterizes the observational semantics introduced in [27] based on limits of infinite computations, by means of a co-inductive definition. It follows from [27] that our bisimilarity coincides with the equivalence induced by the standard closure operators

semantics of ccp. Therefore, our weak bisimulation approach represents a novel sound and complete proof technique for observational equivalence in ccp.

Our work is also interesting for the research programme on "labels derivation". Our labeled semantics can be regarded as an instance of the one introduced at an abstract level in [18]. Syntactical bisimulation (Def. 8) as an instance of the one in [18], while strong and weak bisimulations (Def. 9 and Def. 10) as instances of those in [6]. Furthermore, syntactical bisimulation intuitively coincides with the one in [26], while saturated barbed bisimulation (Def. 6) with the one in [19]. Recall that syntactical bisimilarity is too fine grained, while saturated barbed bisimulation requires the relation to be upward closed (and thus, infinite in dimension). Our weak bisimulation instead is fully abstract and avoid the upward closure. Summarizing, the framework in [6] provides us an abstract approach for deriving a novel interesting notion of bisimulation.

It is worth noticing that the restriction to the summation-free fragment is only needed for proving the coincidence with [27]. The theorem in Section 2.1 still holds in the presence of summation. Analogously, we could extend all the definitions to infinite constraints without invalidating these theorems.

Some recent works [9,17,16] have defined bisimilarity for novel languages featuring the interaction paradigms of both ccp and the π-calculus. In these works, bisimilarity is defined starting from transition systems whose labels represent communications in the style of the π-calculus. Instead we employ barbs on a purely unlabeled semantics. Preliminary attempts have shown that defining a correspondence with our semantics is not trivial. We left this for an extended version of the paper.

As shown e.g. in [19] there are strong connections between ccp processes and logic formulae. As future work we would like to investigate whether our present results can be adapted to provide a novel characterization of logic equivalence in terms of bisimilarity. Preliminary results show that at least the propositional fragment, without negation, can be characterized in terms of bisimilarity.

Finally, we are implementing a checker for our equivalence by employing [8].

References

1. Extended version. Technical report, `http://www.lix.polytechnique.fr/ luis.pino/files/ FOSSACS11-extended.pdf`
2. Abramsky, S., Jung, A.: Domain theory. In: Handbook of Logic in Computer Science, pp. 1–168. Clarendon Press, Oxford (1994)
3. Amadio, R.M., Castellani, I., Sangiorgi, D.: On bisimulations for the asynchronous pi-calculus. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 147–162. Springer, Heidelberg (1996)
4. Bartoletti, M., Zunino, R.: A calculus of contracting processes. In: LICS, pp. 332–341. IEEE Computer Society, Los Alamitos (2010)
5. Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: Mobile processes, nominal data, and logic. In: LICS, pp. 39–48 (2009)
6. Bonchi, F., Gadducci, F., Monreale, G.V.: Reactive systems, barbed semantics, and the mobile ambients. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 272–287. Springer, Heidelberg (2009)

7. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: LICS, pp. 69–80 (2006)
8. Bonchi, F., Montanari, U.: Minimization algorithm for symbolic bisimilarity. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 267–284. Springer, Heidelberg (2009)
9. Buscemi, M.G., Montanari, U.: Open bisimulation for the concurrent constraint pi-calculus. In: Gairing, M. (ed.) ESOP 2008. LNCS, vol. 4960, pp. 254–268. Springer, Heidelberg (2008)
10. de Boer, F.S., Pierro, A.D., Palamidessi, C.: Nondeterminism and infinite computations in constraint programming. Theor. Comput. Sci. 151(1), 37–78 (1995)
11. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 151–166. Springer, Heidelberg (2004)
12. Falaschi, M., Gabbrielli, M., Marriott, K., Palamidessi, C.: Confluence in concurrent constraint programming. Theor. Comput. Sci. 183(2), 281–315 (1997)
13. Di Gianantonio, P., Honsell, F., Lenisa, M.: Rpo, second-order contexts, and lambda-calculus. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 334–349. Springer, Heidelberg (2008)
14. Henkin, J.M.L., Tarski, A.: Cylindric Algebras (Part I). North-Holland, Amsterdam (1971)
15. Honda, K., Yoshida, N.: On reduction-based process semantics. Theor. Comput. Sci. 151(2), 437–486 (1995)
16. Johansson, M., Bengtson, J., Parrow, J., Victor, B.: Weak equivalences in psi-calculi. In: LICS, pp. 322–331 (2010)
17. Johansson, M., Victor, B., Parrow, J.: A fully abstract symbolic semantics for psi-calculi. CoRR, abs/1002.2867 (2010)
18. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
19. Mendler, N.P., Panangaden, P., Scott, P.J., Seely, R.A.G.: A logical view of concurrent constraint programming. Nord. J. Comput. 2(2), 181–220 (1995)
20. Milner, R.: Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge (1999)
21. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 685–695. Springer, Heidelberg (1992)
22. Montanari, U., Sassone, V.: Dynamic congruence vs. progressing bisimulation for ccs. FI 16(1), 171–199 (1992)
23. Palamidessi, C., Saraswat, V.A., Valencia, F.D., Victor, B.: On the expressiveness of linearity vs persistence in the asychronous pi-calculus. In: LICS, pp. 59–68 (2006)
24. Rathke, J., Sassone, V., Sobociński, P.: Semantic barbs and biorthogonality. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 302–316. Springer, Heidelberg (2007)
25. Rathke, J., Sobocinski, P.: Deconstructing behavioural theories of mobility. In: IFIP TCS, vol. 273, pp. 507–520. Springer, Heidelberg (2008)
26. Saraswat, V.A., Rinard, M.C.: Concurrent constraint programming. In: POPL, pp. 232–245 (1990)
27. Saraswat, V.A., Rinard, M.C., Panangaden, P.: Semantic foundations of concurrent constraint programming. In: POPL, pp. 333–352 (1991)
28. Sassone, V., Sobocinski, P.: Reactive systems over cospans. In: LICS, pp. 311–320 (2005)
29. Sewell, P.: From rewrite rules to bisimulation congruences. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 269–284. Springer, Heidelberg (1998)
30. Saraswat, V.A.: Concurrent Constraint Programming. PhD thesis, Carnegie-Mellon University (1989)