Bisimulation Techniques and Algorithms for Concurrent Constraint Programming

Andrés Aristizábal

ÉCOLE POLYTECHNIQUE
Thèse de Doctorat
Spécialité Informatique

BISIMULATION TECHNIQUES AND ALGORITHMS FOR CONCURRENT CONSTRAINT PROGRAMMING

Présentée et soutenue publiquement par

ANDRÉS ALBERTO ARISTIZÁBAL PINZÓN

17 octobre 2012

devant le jury composé de

Rapporteurs:	María ALPUENTE
	Fabio GADDUCCI
Directeurs de thèse:	Catuscia PALAMIDESSI
	Frank D. VALENCIA
Examinateurs:	Filippo BONCHI
	Carlos AGÓN
	François FAGES
	Eva CRÜCK

To God and my beloved and supportive parents, Hersilia and Alvaro

Abstract

Concurrent constraint programming (ccp) is a well-established model for concurrency that builds upon operational and algebraic notions from process calculi and first-order logic. It singles out the fundamental aspects of asynchronous systems whose agents (or processes) evolve by posting and querying (partial) information in a global medium. Bisimilarity is one of the central reasoning techniques in concurrency. It is the main representative of the so called behavioral equivalences, i.e., equivalence relations that determine when two processes (e.g., the specification and the implementation) behave in the same way. 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, in this dissertation we introduce a labeled transition semantics and a novel notion of bisimilarity that is fully abstract wrt the typical observational equivalence in ccp. This way we provide ccp with a new proof technique coherent with existing ones. Remarkably, our co-inductive characterization of observation equivalence for ccp in terms of bisimilarity avoids complicated concepts used in previous work such as fairness and infinite computations.

When the state space of a system is finite, the ordinary notion of bisimilarity can be computed via the well-known *partition refinement algorithm*, unfortunately, this algorithm does not work for ccp bisimilarity. In this dissertation we propose a variation of the standard partition refinement algorithm for verifying ccp strong bisimilarity. To the best of our knowledge this is the first work providing for the automatic verification of concurrent constraint programme.

Weak bisimiliarity is a central behavioural equivalence in process calculi and it is obtained from the strong case by taking into account only the actions that are observable in the system. Typically, the standard partition refinement can also be used for deciding weak bisimilarity simply by using Milner's reduction from weak to strong bisimilarity; a technique referred to as saturation. In this dissertation we demonstrate that, because of its involved labeled transitions, the above-mentioned saturation technique does not work for ccp. We give an alternative reduction from weak ccp bisimilarity to the strong one that allows us to use the ccp partition refinement algorithm that we introduced for deciding this equivalence.

Acknowledgments

First of all I want to express my deepest gratitude to my supervisors Frank D. Valencia and Catuscia Palamidessi for their guidance, encouragement, dedication and faith in my work. I am really indebted with them for inspiring me and for enlightening my professional path through their hard work, dedication and enthusiasm for Computer Science. Right now, I can truly say that I have become a better researcher and most relevantly, a better person. Certainly, I have nourished myself from my two supervisors whom have always cared for me in any aspect of my life, always guiding me into the right direction and giving me one of the best environments any student could ever desire. I hope, one day, I can be just like them, professionally and personally.

Special thanks to Camilo Rueda, the person that lighted in me the torch of research in theoretical computer science. The first one that believed in my abilities, giving me the opportunity to make my first steps in research, in our group, AVISPA. Also my thanks to the members of this group and specially to Carlos Olarte, for all the invaluable help he gave me during my studies.

I also owe a lot to my mentor and friend Filippo Bonchi who believed in me more than myself, who placed me on the right track regarding my research work with his brilliant ideas, his excellent advices and his motivation. Although he only stayed for 6 months at LIX, he still worked with me during the remaining 2 years and a half of my PhD studies, teaching me a great variety of things and as I have already stated, providing me with the basics and guidances for this dissertation to be successfully concluded.

My gratitude also goes to Luis Pino a colleague and great friend at LIX who worked side by side with me during most of my studies.

Thanks to the DGA and the CNRS for funding my doctoral studies and to

INRIA and the Laboratoire d'Informatique de l'École Polytechnique for allowing me to develop my PhD thesis there.

Many thanks to Alejandro Arbeláez, Filippo Bonchi, Sophia Knight, Jorge A. Pérez and Luis Pino to proofread some of the chapters of this dissertation. I am also grateful to María Alpuente, Fabio Gadducci, François Fages, Filippo Bonchi, Carlos Agón and Eva Crück to be part of my jury.

I also want to thank my colleagues and friends at LIX, Miguel, Nico, Kostas, Ehab, Sophia, Matteo, Luis, Hamadou, Marco and Lili. My gratitude also goes to Marie-Jeanne Gaffard (thanks indeed for all your help, understanding and advices), Christelle Lievin, Valérie Lecompte, Valérie Berthou, Audrey Lemarechal, Christine Ferret, Lydie Fontaine, Docteur Martine Bettane-Caperan, Anne Delaigue and Alexandra Bélus.

I want express my affection to my friends around the world: María del Mar Caldas, Gloria Correa, Alejandro Arbeláez, Vero Giraldo, Conie Medina, Lina Palacio, Jorge A. Pérez, Fernando Matiz, Anita Serrano, Ferenc Figueroa, Paola Bianchi, Kate Puzakova, Dasha Kovaleva, Magda Olesińska, Alexandra Silva, Viktoria Nasyrova, Kat, Lara, Luis Alfredo Céspedes, Jérémy Dubreil, Andrea Mairone, Justin Dean, Lorena Montes and many others.

I am grateful to the group of ex-magistrates and all my relatives for giving me their support and love in one way or the other. Elenita and Beatriz Elena, you mean a lot to me, I am lucky to have you as my aunt and cousin. Alejandro Almario, you were a second father to me, your memory will not vanish from my heart.

I will never forget those families that took me as one of them. Thanks a lot to Sara, Felipe and Frank, and Madame Passani and her friends.

I am deeply indebted with people such as Blanca Estelia, Jaime, my dad's nurses and therapists, etc.

Comète team, what can I say?, I will remember all the parties, games, meetings, chats, trips, even the sleepless nights working. Those memories will never be erased, this group will be forever in my heart.

I have no words to describe the last months of my PhD, I have been through really tough but enriching times. I have grown as a person. I have found my true friends. But mainly I thank God for finding my girlfriend Krystyna Węgrzyniak, she has supported and helped me during the most difficult times. Thanks for loving and pampering me during my personal problems and the last and most stressful phase of my studies.

Last but not least, I thank God, my mother and father from the bottom of my heart. Without them my goals would have never been fulfilled. My appropriate gratitude to my father, Alvaro Aristizábal, a real fighter of life, who taught me how to behave myself and gave me his constant support. To my mother, Hersilia Pinzón, my deepest appreciation for her love, help, understanding, tenderness, for her advices and obviously for her hard work carrying me through almost all life. All in all, my parents deserve this PhD title as much or even more than I do.

Andrés A. Aristizábal P.
September the 23rd, 2012

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An ounce of practice is worth more than tons of preaching.
– Mahatma Gandhi.

If I can introduce someone to something new, as is constantly
happening to me, then I am elated.
– Trevor Dunn.

Concurrent constraint programming (ccp) [SR90] is a model for concurrency that combines operational and algebraic notions from process calculi and first-order logic. It was designed to give programmers explicit access to the concept of partial information. A distinctive aspect of ccp is that, unlike other process calculi, is not concerned about point-to-point channel communication but rather about systems of agents posting and querying information (traditionally referred to as *constraints*) in some medium.

This dissertation is devoted to the development of a co-inductive reasoning technique, namely *bisimilarity*, for ccp. We shall demonstrate that the distinctive nature of ccp makes this development a non-trivial, novel and relevant task for the modelling of concurrent systems. In the following chapters we shall present the foundations of ccp bisimilarity as well as the techniques and algorithms to decide this relation. In what follows we motivate and briefly describe the contributions of this work.

1.1 Bisimilarity for CCP

Concurrency is concerned with the fundamental aspects of 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.

Co-induction [JR97, San09, San11] is the most natural and powerful approach to define and reason about infinite or circular structures and computations, bringing in tools and methods that are dual and complementary to those of induction. In particular, coinduction is central in concurrency, where processes are naturally seen as entities that continuously interact with their environment. Coinduction, in the form of bisimulation, is used to define equality between processes, and the coinduction proof method, in the form of the bisimulation proof method, is employed to prove such equalities. The largest bisimulation, *bisimilarity* [Mil80, Mil99], captures an intuitive notion of process equivalence; two processes are equivalent if they can match each other's moves.

Concurrent Constraint Programming (ccp) [SR90] 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 *posting* (or *telling*) and *asking* information (traditionally referred to as *constraints*) in a medium (referred to as *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 [PSVV06, BM08, BJPV09, BZ10] on calculi exhibiting data-types, logic assertions as well as tell and ask operations.

Surprisingly, the development of co-inductive techniques, in particular, of bisimulation for ccp has been so far too little considered. There have been few attempts to define a notion of bisimilarity for ccp. The ones we are aware of are those in [SR90] and [MPSS95] upon which we build. These equivalences are not completely satisfactory: We shall demonstrate in this thesis that the first one may tell apart processes with identical observable behavior, 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.

In this dissertation we shall introduce a notion of bisimilarity for ccp that allows us 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 [SRP91] and the connection with logic [MPSS95].

1.2 Labeled Semantics

Bisimilarity relies on *labeled 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 *unlabeled transitions* and *barbed congruence* [MS92a]. The main drawback of this approach is that to verify barbed congruences it is often necessary to analyze the behavior of processes under every context.

This scenario has motivated a novel stream of research [Sew98, LM00, EK04, SS05, BKM06, RS08, GHL08, BGM09] 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 [LM00] 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:

$$P \xrightarrow{C} P' \text{ iff } C[P] \longrightarrow P'$$

where C is the minimal context allowing such reduction.

In [BKM06, BGM09], 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 with a move $C[Q] \longrightarrow Q'$, where $C[-]$ is not necessarily the minimal.

Labeled Semantics 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 [Sew98, LM00]. 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 [SR90], but the recent developments in [BGM09] 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 [LM00]. As for the bisimilarity in [SR90], it is too fine grained, i.e., it separates processes which are behaviorally indistinguishable. Instead, the notion of bisimulation from [BGM09] (instantiated to the case of ccp) is fully abstract with respect to the standard observational equivalence given in [SRP91]. Our work can therefore be also regarded as a compelling application of the theory of reactive systems.

1.3 Algorithms

A fundamental issue in concurrency concerns the analysis techniques. That is, methods and tools to decide equivalence between behaviours, or more generally to prove behavioural properties of a concurrent system. The tractability of a technique is important: we need techniques which make proofs shorter, and reduce the number of cases to be examined when analysing a behaviour. Tools may be derived out of the techniques, to permit automatic or semi-automatic analysis.

Many efficient algorithms and tools for bisimilarity checking have been developed [VM94, Fer89, FGM^{+}98]. Among these, the *partition refinement algorithm*

[KS83] is the best known: first it generates the state space of a labeled transition system (LTS), i.e., the set of states reachable through the transitions; then, it creates a partition equating all states and afterwards, iteratively, refines these partitions by splitting non equivalent states. At the end, the resulting partition equates all and only bisimilar states.

The ccp formalism has been widely investigated and tested in terms of theoretical studies and the implementation of several ccp programming languages. From the *applied computing point of view*, however, ccp lacks algorithms and tools to automatically verify program equivalence. Furthermore, the absence of a well-behaved notion of bisimilarity for ccp made the development of tools for automatic verification in ccp somewhat a pointless effort. Therefore, after introducing a suitable definition of bisimilarity for ccp, we shall give the first step towards automatic verification of ccp program equivalences by presenting an algorithm based on an alternative notion of bisimilarity, namely *saturated barbed bisimilarity* ($\dot{\sim}_{sb}$) for ccp.

Two configurations are equivalent according to $\dot{\sim}_{sb}$ if (i) they have the same store, (ii) their transitions go into equivalent states and (iii) they are still equivalent when adding an arbitrary constraint to the store. We shall also prove that the weak variant of $\dot{\sim}_{sb}$ is shown to be *fully abstract* w.r.t. the standard observational equivalence of [SRP91].

Unfortunately, the standard partition refinement algorithm does not work for $\dot{\sim}_{sb}$ because condition (iii) requires to check all possible constraints that might be added to the store. In this dissertation we shall introduce a modified partition refinement algorithm for $\dot{\sim}_{sb}$.

We closely follow the approach in [BM09] that studies the notion of saturated bisimilarity from a more general perspective and proposes an abstract checking procedure. We first define a *derivation relation* $\vdash_D$ amongst the transitions of ccp processes: $\gamma \xrightarrow{\alpha_1} \gamma_1 \vdash_D \gamma \xrightarrow{\alpha_2} \gamma_2$ which intuitively means that the latter transition is a logical consequence of the former. Then we introduce the notion of *domination* ($\succ_D$), which means that if there exist two transitions $\gamma \xrightarrow{\alpha_1} \gamma_1$ and $\gamma \xrightarrow{\alpha_2} \gamma_2$, $\gamma \xrightarrow{\alpha_1} \gamma_1 \succ_D \gamma \xrightarrow{\alpha_2} \gamma_2'$ if and only if $\gamma \xrightarrow{\alpha_1} \gamma_1 \vdash_D \gamma \xrightarrow{\alpha_2} \gamma_2'$ and $\alpha_1 \neq \alpha_2$. Finally we give the definition of a *redundant transition*. Intuitively $\gamma \xrightarrow{\alpha_2} \gamma_2$ is redundant if $\gamma \xrightarrow{\alpha_1} \gamma_1$ dominates it, that is $\gamma \xrightarrow{\alpha_1} \gamma_1 \succ_D \gamma \xrightarrow{\alpha_2} \gamma_2'$ and

$\gamma_2 \mathrel{\dot{\sim}_{sb}} \gamma_2'$. Now, if we consider the LTS having only non-redundant transitions, the ordinary notion of bisimilarity coincides with $\dot{\sim}_{sb}$. Thus, in principle, we could remove all the redundant transitions and then check bisimilarity with the standard partition refinement algorithm. But how can we decide which transitions are redundant, if redundancy itself depends on $\dot{\sim}_{sb}$?

Our solution consists in computing $\dot{\sim}_{sb}$ and redundancy *at the same time*. In the first step, the algorithm considers all the states as equivalent and all the transitions (potentially redundant) as redundant. At any iteration, states are discerned according to (the current estimation of) non-redundant transitions and then non-redundant transitions are updated according to the new computed partition.

A distinctive aspect of our algorithm is that in the initial partition we insert not only the reachable states, but also extra ones which are needed to check for redundancy. We prove that these additional states are finitely many and thus the termination of the algorithm is guaranteed whenever the original LTS is finite (as it is the case of the standard partition refinement). Unfortunately, the number of these states might be exponential w.r.t. the size of the original LTS, consequently the worst-case running time is exponential.

1.4 From Weak to Strong Bisimilarity

A major dichotomy among behavioural equivalences concerns *strong* and *weak* equivalences. In strong equivalences, all the transitions performed by a system are deemed observable. In weak equivalences, instead, internal transitions (usually denoted by τ) are unobservable. On the one hand, weak equivalences are more abstract (and thus closer to the intuitive notion of behaviour); on the other hand, strong equivalences are usually much easier to be checked (for instance, in [LPSS11] it is introduced a strong equivalence which is computable for a Turing complete formalism).

In Section 1.3 we develop a partition refinement algorithm to verify strong bisimilarity for ccp.

Weak bisimilarity can be computed by reducing it to strong bisimilarity. Given an LTS $\xrightarrow{a}$, one can build $\overset{a}{\Longrightarrow}$ as follows.

$$\frac{P \xrightarrow{a} Q}{P \overset{a}{\Longrightarrow} Q} \qquad \frac{}{P \overset{\tau}{\Longrightarrow} P} \qquad \frac{P \overset{\tau}{\Longrightarrow} P_1 \overset{a}{\Longrightarrow} Q_1 \overset{\tau}{\Longrightarrow} Q}{P \overset{a}{\Longrightarrow} Q}$$

Since weak bisimilarity on $\xrightarrow{a}$ coincides with strong bisimilarity on $\overset{a}{\Longrightarrow}$, then one can check weak bisimilarity with the algorithms for strong bisimilarity on the new LTS $\overset{a}{\Longrightarrow}$.

Nevertheless, this standard reduction method cannot entirely be regarded as the general solution since it works with labels resembling actions, but not, as in the case of ccp, as constraints.

1.5 Summary of Contributions and Organization

Here we briefly describe the structure of this dissertation and its contributions. Each chapter begins with a brief introduction and relevant background and ends with a summary of its contents and a discussion about its related work. Frequently used notational conventions and terminology are summarized in the Index.

Chapter 2 (Deriving Labels and Bisimilarity for CCP) In this chapter we provide a labeled transition semantics and a novel notion of labeled bisimilarity for ccp by building upon the work in [SR90, BGM09]. We also establish a strong correspondence with existing ccp notions by providing a fully-abstract characterization of a standard observable behavior for *infinite* ccp processes (*the limits of fair, possibly infinite, computations*). Remarkably, complex notions such as fairness and infinite computations are avoided in our characterization of the observable behavior via bisimilarity. From [SRP91] 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.

Chapter 3 (Partition Refinement for Bisimilarity in CCP) In this chapter we provide an algorithm that allows us to verify saturated barbed bisimilarity for ccp. To the best of our knowledge, this is the first algorithm for the automatic verification of ccp program equivalence. This is done by building upon the results of [BM09]. We also show the termination and the complexity of the algorithm. We have implemented the algorithm in c++ and the code is available at http://www.lix.polytechnique.fr/~andresaristi/strong/.

Chapter 4 (Reducing Weak to Strong Bisimilarity in CCP) In this chapter we first show that the standard method for reducing weak to strong bisimilarity does not work for ccp and then we provide a way out of the impasse. Our solution can be readily explained by observing that the labels in the LTS of a ccp agent are constraints (actually, they are "the minimal constraints" that the store should satisfy in order to make the agent progress). These constraints form a lattice where the least upper bound (denoted by $\sqcup$) intuitively corresponds to conjunction and the bottom element is the constraint $true$. (As expected, transitions labeled by $true$ are internal transitions, corresponding to the τ moves in standard process calculi.) Now, rather than closing the transitions just with respect to $true$, we need to close them w.r.t. all the constraints. Formally we build the new LTS with the following rules.

$$\frac{P \xrightarrow{a} Q}{P \overset{a}{\Longrightarrow} Q} \qquad \frac{}{P \overset{true}{\Longrightarrow} P} \qquad \frac{P \overset{a}{\Longrightarrow} Q \overset{b}{\Longrightarrow} R}{P \overset{a \sqcup b}{\Longrightarrow} R}$$

Note that, since $\sqcup$ is idempotent, if the original LTS $\xrightarrow{a}$ has finitely many transitions, then also $\overset{a}{\Longrightarrow}$ is finite. This allows us to use the algorithm in [ABPV12a] to check weak bisimilarity on (the finite fragment) of concurrent constraint programming. We have implemented this procedure in a tool that is available at `http://www.lix.polytechnique.fr/~andresaristi/checkers/`. At the best of our knowledge, this is the first tool for checking weak equivalence of ccp programs.

Chapter 5 (Conclusions) This Chapter offers some concluding remarks, discusses related works, and proposes some future work.

Appendix (A Tool for Verifying Bisimilarity in CCP) In the appendix we briefly describe a tool for verifying strong and weak bisimilarity for ccp using as a basis all the theoretical knowledge we gathered. We highlight the importance of the selected programming language, the abstract data types used to represent the components to implement the partition refinement algorithm for ccp, the analysis of the auxiliary functions or procedures, a study of several graphs that can show the relation between the way a ccp process is composed, and how much time it takes for the algorithm to give an answer regarding the strong or the weak bisimilarity.

1.6 Publications from this Dissertation

Most of the material of this dissertation has been previously reported in the following papers.

- **Proceedings of conferences.**
  - Andres Aristizabal, Filippo Bonchi, Luis Pino and Frank D. Valencia. *Partition Refinement for Bisimilarity in CCP*. In Proc of SAC 2012: 88-93. ACM Press. 2012 [ABPV12b].

    The main contributions of this paper are included in Chapter 3 and 4, and Appendix A.
  - Andres Aristizabal, Filippo Bonchi, Catuscia Palamidessi, Luis Pino and Frank D. Valencia. *Deriving Labels and Bisimilarity for Concurrent Constraint Programming*. In Proc of FOSSACS 2011: 138-152. Springer 2011 [ABP$^+$11b].

    The main contributions of this paper are included in Chapter 2.

- **Proceedings of Workshops.**
  - Andres Aristizabal, Filippo Bonchi, Luis Pino and Frank D. Valencia. *Reducing Weak to Strong Bisimilarity in CCP*. In Proc of ICE 2012 [ABPV12c].

    The main contributions of this paper are included in Chapter 4 and Appendix A.
- **Short Papers.**
  - Andres Aristizabal. *Bisimilarity in Concurrent Constraint Programming*. (Short Paper) In Proceedings ICLP 2010. Springer, 2010. [Ari10a]
  - Andres Aristizabal. *Bisimilarity in Concurrent Constraint Programming*. (Short Paper) YR-CONCUR 2010, a satellite workshop of CONCUR 2010. [Ari10b].
- **Technical Reports.**
  - Andres Aristizabal, Filippo Bonchi, Luis Pino and Frank D. Valencia. *Partition Refinement for Bisimilarity in CCP (Extended Version)*. Technical Report. INRIA-CNRS, 2012 [ABPV12a].
  - Andres Aristizabal, Filippo Bonchi, Catuscia Palamidessi, Luis Pino and Frank D. Valencia. *Deriving Labels and Bisimilarity for Concurrent Constraint Programming (Extended Version)*. Technical Report. INRIA-CNRS, 2011 [ABP$^+$11a].

Chapter 2

Deriving Labels and Bisimilarity for CCP

Labeled semantics does two things: it gives the semantics of the raw processes, the computational steps and it helps you reason about the congruences. Reduction semantics just does the first thing and then you have some horrid rules about congruences and you need some nice tools to reason about this.
– Martin Berger.

In this chapter we introduce a labeled semantics for ccp [SR90] and a well-behaved equivalence for this language, namely, a new coarse grained bisimilarity notion for ccp. We begin by presenting the notion of constraint systems (cs), then we show the basic insights of ccp by recalling its syntax, its operational semantics and its observational equivalence. We show that the only existing notion of standard bisimilarity for ccp, one of the central reasoning techniques in concurrency, is not completely satisfactory for ccp since it yields an equivalence that is too fine grained. Therefore, by building upon recent foundational investigations, we give a labeled transition semantics and a novel notion of bisimilarity that is fully abstract w.r.t. the typical observational equivalence in ccp. This way we are able to provide ccp with a new proof technique coherent with the existing ones.

2.1 Background

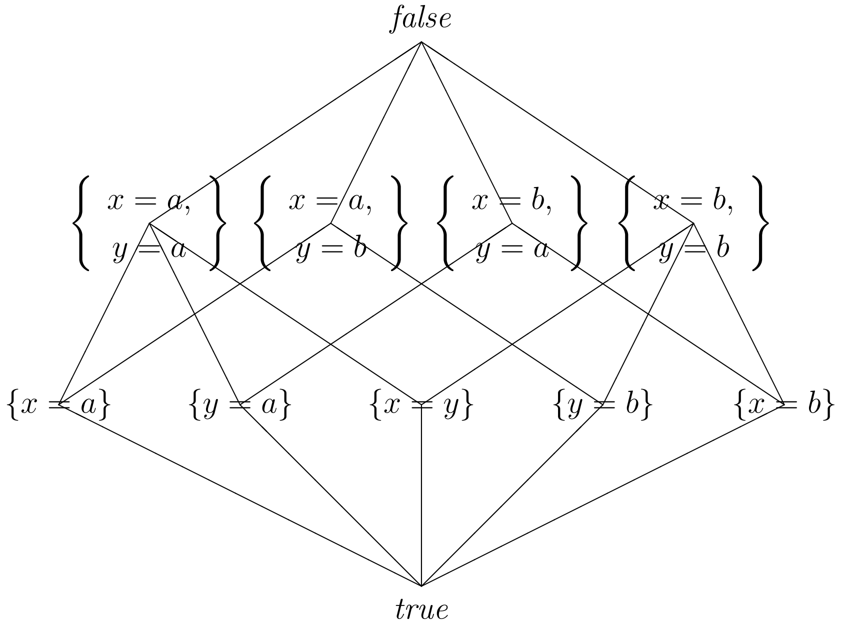
2.1.1 Constraint Systems

The ccp model is parametric in a constraint system specifying the structure and interdependencies of the information that processes can ask and tell. We presuppose a basic knowledge of domain theory (see [AJ94]). Following [SRP91, dBPP95], 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 2.1.1 *A* constraint system **C** *is a complete algebraic lattice* $(\mathit{Con}, \mathit{Con}_0, \sqsubseteq, \sqcup, \mathit{true}, \mathit{false})$ *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 *and* false *are the least and greatest elements of* Con*, respectively.*

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

Example 2.1.1 *We briefly explain the* Herbrand *cs from [V.A89, SRP91]. This cs captures* syntactic *equality between terms* $t, t', \ldots$ *built from a first-order alphabet* $\mathcal{L}$ *with countably many variables* $x, y, \ldots$*, function symbols, and equality* $=$*. The constraints are sets of equalities over the terms of* $\mathcal{L}$ *(e.g.,* $\{x = t, y = t\}$ *is a constraint). The relation* $c \sqsubseteq d$ *holds if the equalities in* c *follow from those in* d *(e.g.,* $\{x = y\} \sqsubseteq \{x = t, y = t\}$*). The constraint* false *is the set of all term equalities in* $\mathcal{L}$ *and* true *is (the equivalence class of) the empty set. The finite elements are the (equivalence classes of) finite sets of equalities. The lub is (the equivalence class of) set union. (See [V.A89, SRP91] for full details). Figure 2.1 represents this kind of constraint lattice for* x*,* y*,* a*,* b*. Let us stress that the only symbols of* $\mathcal{L}$ *are the constants* a *and* b*.*



$false$
$\left\{ \begin{array}{c} x = a, \\ y = a \end{array} \right\}$
$\left\{ \begin{array}{c} x = a, \\ y = b \end{array} \right\}$
$\left\{ \begin{array}{c} x = b, \\ y = a \end{array} \right\}$
$\left\{ \begin{array}{c} x = b, \\ y = b \end{array} \right\}$
$\{x = a\}$
$\{y = a\}$
$\{x = y\}$
$\{y = b\}$
$\{x = b\}$
$true$

Note that the constraints are made out of a variable and a term, therefore $a = b$ cannot be in the lattice.

Figure 2.1: The Herbrand Constraint Lattice for x, y, a, b.

In order to model *hiding* of local variables and *parameter passing*, in [SRP91] the notion of constraint system is enriched with *cylindrification operators* and *diagonal elements*, concepts borrowed from the theory of cylindric algebras (see [HT71]).

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

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

$\mathrm{C5} - \mathrm{C7}:$

$\mathbf{C1}.\ \exists_x\exists_y c = \exists_y\exists_x c$

$\mathbf{C2}.\ d_{xx} = true$

$\mathbf{C3}.\ \textit{if } z \neq x, y \textit{ then } d_{xy} = \exists_z(d_{xz} \sqcup d_{zy})$

$\mathbf{C4}.\ \exists_x(c \sqcup \exists_x d) = \exists_x c \sqcup \exists_x d$

$\mathbf{C5}.\ \exists_x c \sqsubseteq c$

$\mathbf{C6}.\ \textit{if } c \sqsubseteq d \textit{ then } \exists_x c \sqsubseteq \exists_x d$

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

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 $x = y$.

Cylindrification and diagonal elements allow us to model the variable renaming of a formula ϕ; in fact, by the aforementioned axioms, the formula $\exists_x(d_{xy} \sqcup \phi) = true$ can be depicted as the formula $\phi[y/x]$, i.e., the formula obtained from ϕ by replacing all free occurrences of x by 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) $fv(c \sqcup d) = fv(c) \cup fv(d)$.

Example 2.1.2 *The Herbrand constraint system in Example 2.1.1 can be extended to be a cylindric constraint system, where $\exists_x$ just represents the standard existential quantifier. For instance, consider the alphabet which contains a constant symbol a and a monadic function symbol f. Figure 2.2 represents the part of the cylindric constraint system in which x (and only x) is free. For simplicity we have indicated a set $\{t = u\}$ by $t = u$.*

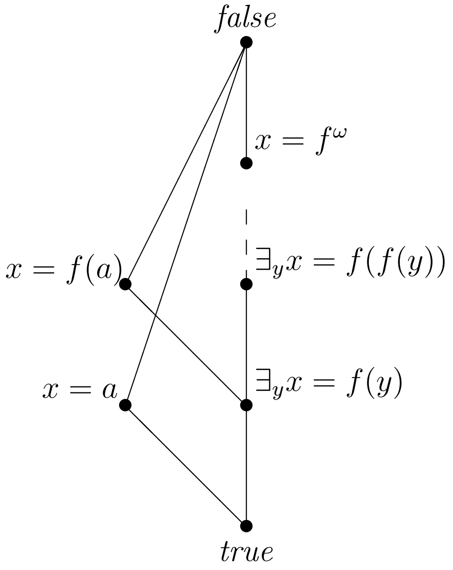


Figure 2.2: The Cylindric Herbrand Constraint System for x, a and f.

The constraint $x = f^\omega$ stands for the limit of the chain $\{\exists_y x = f^i(y)\}_i$.

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

Example 2.1.3 (The 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: $\phi ::= \quad t \mid \phi_1 \wedge \phi_2 \mid \exists_x \phi$ and $t ::= \quad e_1 = e_2 \mid e_1 < e_2 \mid succ(e_1, e_2)$ where e_1 and e_2 are either 0 or ∞ or variables in Var. Note that formulas like $x = n$ or $x < n$ (for $n = 1, 2, \dots$) do not belong to $\mathcal{L}$. A useful abbreviation to express them is $succ^n(x, y) \stackrel{\text{def}}{=} \exists y_0 \dots \exists y_n (\bigwedge_{0<i\leq n} succ(y_{i-1}, y_i) \wedge x = y_0 \wedge y = y_n)$. 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 there 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 2.3. 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 chain $\cdots \sqsubset c_n \sqsubset \cdots \sqsubset c_1 \sqsubset c_0$. 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 orders and minimal elements are related by the following result.

Proposition 2.1.1 *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 proposition is fundamental for proving the completeness of labeled semantics (Lemma 2.3.2).

2.1.2 Syntax

Ccp was proposed in [V.A89] and then refined in [SR90, SRP91]. In this chapter we restrict ourselves to the summation-free fragment of this formalism. The distinctive confluent (for more details about confluence in ccp see [FGMP94]) nature of this fragment is necessary for showing that our notion of bisimilarity coincides with the observational equivalence for infinite ccp processes given in [SRP91].

Remark 2.1.1 *A ccp summation-free fragment is the one in which the nondeterministic choice is not included in the syntax of the ccp language.*

Definition 2.1.2 *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:*

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

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

Remark 2.1.2 *The notions and relations in this and the following chapters assume an underlying cylindric constraint system* $\mathbf{C}$.

Finite processes. Intuitively, the tell process $\mathbf{tell}(c)$ adds c to the global store. This addition is performed regardless of the generation of inconsistent information. The ask process $\mathbf{ask}(c) \rightarrow P$ may execute P if c is entailed by the information in the store. The process $P \mid Q$ 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 behavior, ccp provides parametric process definitions. A process $p(\vec{z})$ is said to be a *procedure call* with identifier p and actual parameters $\vec{z}$. We presuppose that for each procedure call $p(z_1 \ldots z_m)$ there exists a unique (possibly recursive) procedure definition of the form $p(x_1 \ldots x_m) \stackrel{\texttt{def}}{=} P$ 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 behavior 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 procedure definitions.

Although we have not yet defined 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 Example 2.1.3 as the underlying constraint system in all examples.

Example 2.1.4 *Consider the following (family of) process definitions.*

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

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

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 $up_0(x) \mid \mathbf{ask}(42 < x) \rightarrow \mathbf{tell}(z = 0)$ 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.) □

2.1.3 Reduction Semantics

To describe the evolution of processes, we extend the syntax by introducing a process $\mathtt{stop}$ representing successful termination, and a process $\exists^e_x P$ 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 $\exists^e_x P$: it represents the situation in which the local store is empty. Namely, $\exists_x P = \exists^{true}_x P$.

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 relation between configurations $\longrightarrow \subseteq \mathit{Conf} \times \mathit{Conf}$ defined in Table 2.1.

Rules R1-R3 and R5 are easily seen to realize the above process intuitions. Rule R4 is somewhat more involved. Intuitively, $\exists_x^e P$ behaves like P, except that the variable x possibly present in P must be considered local, and that the information present in e has to be taken into account. It is convenient to distinguish between the *external* and the *internal* points of view. From the internal point of view, the variable x, possibly occurring in the global store d, is hidden.

This corresponds to the usual scoping rules: the x in d is *global*, hence "covered" by the local x. Therefore, P has no access to the information on x in d, and this is achieved by filtering d with $\exists_x$. Furthermore, P can use the information (which may also concern the local x) that has been produced locally and accumulated in e. In conclusion, if the visible store at the external level is d[1], then the store that is visible internally by P is $e \sqcup \exists_x d$. Now, if P is able to make a step, thus reducing to P' and transforming the local store into e', what we see from the external point of view is that the process is transformed into $\exists_x^{e'} P'$, and that the information $\exists_x e$ present in the global store is transformed into $\exists_x e'$. To show how this works we show an instructive example.

[1] Operationally $\exists_x^e P$ can only derive from a ccp process of the form $\exists_x P''$, which has produced the local information e while evolving into $\exists_x^e P$. This local information is externally seen as $\exists_x e$, that is, $\exists_x e \sqsubseteq d$

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

$$\text{R3 (a)}\quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P \mid Q, d\rangle \longrightarrow \langle P' \mid Q, d'\rangle} \qquad \text{R3 (b)}\quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle Q \mid P, d\rangle \longrightarrow \langle Q \mid P', d'\rangle}$$

$$\text{R4}\quad \frac{\langle P, e \sqcup \exists_x d\rangle \longrightarrow \langle P', e' \sqcup \exists_x d\rangle}{\langle \exists_x^e P, d\rangle \longrightarrow \langle \exists_x^{e'} P', d \sqcup \exists_x e'\rangle}$$

$$\text{R5}\quad \frac{\langle P[\vec{z}/\vec{x}], d\rangle \longrightarrow \gamma'}{\langle p(\vec{z}), d\rangle \longrightarrow \gamma'}$$

where $p(\vec{x}) \overset{\texttt{def}}{=} P$ is a process definition in D

Table 2.1: Reduction Semantics for CCP

Example 2.1.5 *We exhibit below the reduction of* $P = \exists_x^e(\mathbf{ask}\ (y > 1)\ \rightarrow\ Q)$ *where the local store is* $e = x < 1$*, and the global store* $d' = d \sqcup \alpha$ *with* $d = y > x$*,* $\alpha = x > 1$*.*

$$\textit{R4}\quad \dfrac{\textit{R2}\quad \dfrac{(y > 1) \sqsubseteq e \sqcup \exists_x d'}{\langle \mathbf{ask}\ (y > 1)\ \rightarrow\ Q, e \sqcup \exists_x d'\rangle \longrightarrow \langle Q, e \sqcup \exists_x d'\rangle}}{\langle P, d'\rangle \longrightarrow \langle \exists_x^e Q, d' \sqcup \exists_x e\rangle}$$

Note that the x *in* d' *is hidden, by using existential quantification in the reduc-*

tion 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., $(e \sqcup d') = false$). 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. □

2.1.4 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 [FGMP97]. Observe that any transition is generated either by a process $\mathbf{tell}(c)$ or by a process $\mathbf{ask}\ (c) \ \to\ Q$. We say that a process P is *active* in a transition $t = \gamma \longrightarrow \gamma'$ if it generates such transition; i.e. if there exists 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 2.1.3 *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,

Definition 2.1.4 *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 is the constraint $\bigsqcup_i d_i$ and is denoted as $Result(\xi)$.*

Note that for a finite computation the result coincides with the store of the last configuration.

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

Theorem 2.1.1 (from [SRP91]) *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 $Result(\gamma) \overset{\texttt{def}}{=} Result(\xi)$ for any fair computation ξ of γ.

Definition 2.1.5 (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 $P \sim_o Q$ if $\mathcal{O}(P) = \mathcal{O}(Q)$.*

Example 2.1.6 *Consider the processes $P = up_0(x) \mid up_1(y)$ and $Q = \exists_z(\mathbf{tell}(z = 0) \mid \mathbf{ask}(z = 0) \rightarrow fairup(x, y, z))$ with up_0 and up_1 as in Example 2.1.4 and $fairup(x, y, z) \overset{\texttt{def}}{=}$*

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

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 (the results of their fair computations are the same), i.e., $\mathcal{O}(P) = \mathcal{O}(Q)$. □

2.2 Saturated Bisimilarity for CCP

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

2.2.1 Saturated Barbed Bisimilarity

Barbed equivalences have been introduced in [MS92a] 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. [MS92a, Mil99]), while in the asynchronous variants only the outputs (see e.g. [ACS96]). Even several works (e.g. [RSS07, HY95]) 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 (Definition 2.2.3) that coincides with the standard observational equivalence (Definition 2.1.5). 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 $\gamma \downarrow_c$, if $c \sqsubseteq d$; γ *weakly satisfies* the barb c, written $\gamma \Downarrow_c$, if $\gamma \longrightarrow^* \gamma'$ and $\gamma' \downarrow_c$. As usual, $\longrightarrow^*$ denotes the reflexive and transitive closure of $\longrightarrow$.

Definition 2.2.1 (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 $P \dot{\sim}_b Q$ if $\langle P, true \rangle \dot{\sim}_b \langle Q, true \rangle$.

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., $C = \mathbf{tell}(c) \mid [-]$ and $C[\mathbf{tell}(d)] = \mathbf{tell}(c) \mid \mathbf{tell}(d)$.

Example 2.2.1 *Let us consider the context $C = \mathbf{tell}(a) \mid [-]$ 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 [MS92b] for the case of weak bisimilarity in CCS. This work has inspired the introduction of *saturated bisimilarity* [BKM06] (and its extension to the barbed approach [BGM09]). 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 2.2.2 (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 $P \dot{\sim}_{sb} Q$ if $\langle P, true\rangle \dot{\sim}_{sb} \langle Q, true\rangle$.

The weak notion of the aforementioned definition uses $\Downarrow_c$ and $\longrightarrow^*$ instead of $\downarrow_c$ and $\longrightarrow$ respectively.

Definition 2.2.3 (Weak saturated barbed bisimilarity) *A weak 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 weak saturated barbed bisimilar, written $\gamma_1 \dot{\approx}_{sb} \gamma_2$, if there exists a weak saturated barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$. We write $P \dot{\approx}_{sb} Q$ if $\langle P, true\rangle \dot{\approx}_{sb} \langle Q, true\rangle$.

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 behavior of a configuration γ only by adding constraints to its store. The same holds for $\dot{\approx}_{sb}$ in the summation-free fragment.

2.2.2 Correspondence with Observational Equivalence

We now show that $\dot{\approx}_{sb}$ coincides with the observational equivalence $\sim_o$. This is remarkable since $\dot{\approx}_{sb}$ avoids complex concepts such as fairness and infinite elements used in the definition of $\sim_o$. Furthermore, from [SRP91] it follows that $\dot{\approx}_{sb}$ coincides with the standard denotational semantics for ccp.

First, we recall some basic facts from domain theory that are 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 2.2.1 *Let $D = d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ and $E = e_0 \sqsubseteq e_1 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$ be two chains. (1) If they are cofinal, then they have the same limit, i.e., $\bigsqcup D = \bigsqcup E$. (2) If the elements of the chains are* finite *and $\bigsqcup D = \bigsqcup E$, then the two chains are cofinal.*

Proof Recall that D and E are cofinal if for every d_i there exists e_j such that $d_i \sqsubseteq e_j$ and for every e_i there exists d_j such that $e_i \sqsubseteq d_j$.

1. For each $d_i \in D$, there exists $e_{k_i} \in E$ such that $d_i \sqsubseteq e_{k_i}$. Define $f_l = \bigsqcup_{i=0}^{l} e_{k_i}$. Then for all l, $d_l \sqsubseteq f_l$ and $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq$ .... So the chain $d_0, d_1, d_2, ...$ is dominated by the chain $f_0, f_1, f_2, ...$ and therefore $\bigsqcup D \sqsubseteq \bigsqcup_{i \in \mathbb{N}} f_i$. However $\bigsqcup_{i \in \mathbb{N}} f_i = \bigsqcup_{i \in \mathbb{N}} e_{k_i} = \bigsqcup E$, proving that $\bigsqcup D \sqsubseteq \bigsqcup E$. But we can prove in exactly the same way that $\bigsqcup E \sqsubseteq \bigsqcup D$, so we have that $\bigsqcup D = \bigsqcup E$.

2. If $\bigsqcup D = \bigsqcup E$ then for arbitrary d_i, since $d_i \sqsubseteq \bigsqcup D$, $d_i \sqsubseteq \bigsqcup E$, and since d_i is finite, by definition there must be e_j such that $d_i \sqsubseteq e_j$. The same reasoning can be used to prove that for every e_i, there exists d_j with $e_i \sqsubseteq d_j$. Therefore D and E are cofinal.

Lemma 2.2.2 *Let $\langle P_0, d_0\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \dots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \dots$ 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 $c \sqsubseteq d_i$.*

Proof If $\langle P_0, d_0\rangle \Downarrow_c$, then $\langle P_0, d_0\rangle \longrightarrow^* \langle P', d'\rangle$ with $c \sqsubseteq d'$ (by definition of barb). If $\langle P', d'\rangle$ belongs to the above computation (i.e., there exists an i such that $P_i = P'$ and $d_i = d'$) then the result follows immediately. If $\langle P', d'\rangle$ does not belong to the computation, it holds that there exists $\langle P_i, d_i\rangle$ (in the computation above) such that $\langle P', d'\rangle \longrightarrow^* \langle P_i, d_i\rangle$, because (summation-free) ccp is confluent and the computation is fair. Since the store is preserved, $d' \sqsubseteq d_i$ and then $c \sqsubseteq d_i$.□

Theorem 2.2.1 *$P {\sim_o} Q$ if and only if $P \dot{\approx}_{sb} Q$.*

Proof The proof proceeds as follows:

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

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

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

  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 2.2.2, 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 2.2.1.1, 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

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

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

  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, then by

Lemma 2.2.1.2, 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 2.2.2, 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 (Definition 2.2.3). 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 $\mathcal{R}$, $(\langle P', d' \sqcup a \rangle, \langle Q', e' \sqcup a \rangle) \in \mathcal{R}$.□

2.3 Labeled Semantics

Although $\dot{\approx}_{sb}$ 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 Definition 2.2.2. 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.

Labeled 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 [Sew98, LM00] 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.

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

$$\text{LR2} \quad \frac{\alpha \in \min\{a \in \mathit{Con}_0 \mid c \sqsubseteq d \sqcup a\ \}}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \xrightarrow{\alpha} \langle P, d \sqcup \alpha\rangle}$$

$$\text{LR3 (a)} \quad \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P \mid Q, d\rangle \xrightarrow{\alpha} \langle P' \mid Q, d'\rangle}$$

$$\text{LR3 (b)} \quad \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle Q \mid P, d\rangle \xrightarrow{\alpha} \langle Q \mid P', d'\rangle}$$

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

$$x \notin \mathit{fv}(e'), z \notin \mathit{fv}(P) \cup \mathit{fv}(e \sqcup d \sqcup \alpha)$$

$$\text{LR5} \quad \frac{\langle P[\vec{z}/\vec{x}], d\rangle \xrightarrow{\alpha} \gamma'}{\langle p(\vec{z}), d\rangle \xrightarrow{\alpha} \gamma'}$$

where $p(\vec{x}) \stackrel{\mathtt{def}}{=} P$ is a process definition in D

Table 2.2: Labeled Transitions.

The Rules. The labeled transition $\xrightarrow{\alpha} \subseteq \mathit{Conf} \times \mathit{Con}_0 \times \mathit{Conf}$ is defined by the rules in Table 2.2. 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 2.1.

The rule LR2 says that $\langle \textbf{ask}\ (c) \rightarrow P, d\rangle$ 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., $\alpha \in \min\{a \in Con_0 \,|\, c \sqsubseteq d \sqcup a\ \}$. Note that assuming that $(Con, \sqsubseteq)$ is well-founded (Sec. 2.1.1) is necessary to guarantee that α exists whenever $\{a \in Con_0 \,|\, c \sqsubseteq d \sqcup a\ \}$ 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 2.1 would not work. One may be tempted to define the rule for the local case, by analogy to the labeled local rules in other process calculi (e.g., the π-calculus) and R4, as follows:

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

This rule however is not "complete" (in the sense of Lemma 2.3.2 below) as it does not derive all the transitions we wish to have.

Example 2.3.1 *Let P as in Example 2.1.5, i.e., $P = \exists_x^{x<1}(\textbf{ask}\ (y > 1) \rightarrow Q)$ and $d = y > x$. Note that $\alpha = x > 1$ is a minimal constraint that added to d enables a reduction from P. In Example 2.1.5 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 $\langle P, d\rangle$ labeled 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 2.2 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 Example 2.3.1, using LR4 (and LR2) one can derive

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

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 $\langle P', d'\rangle$, there exists a minimal information $\alpha \sqsubseteq a$ such that $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d''\rangle$ with $d'' \sqsubseteq d'$.

For technical reasons we shall use an equivalent formulation of Rule R4. In the new rule (R4'), instead of using existential quantification to hide the global x, we rename it to match the renaming used in its corresponding labeled transition rule (LR4).

$$\text{R4'}\ \frac{\langle P, e \sqcup d[z/x]\rangle \longrightarrow \langle P', e' \sqcup d[z/x]\rangle}{\langle \exists_x^e P, d\rangle \longrightarrow \langle \exists_x^{e'} P', d \sqcup \exists_x e'\rangle} \quad \text{with } z \notin fv(P) \cup fv(e) \cup fv(d)$$

We also use an equivalent formulation of LR4, in which we choose not to rename the local x (with a fresh name z). Instead, we rename the global one thus we check when the environment is giving information about z and rename it (therefore $\alpha[x/z]$). Notice that in LR4' we do not use renaming on the process but we must rename on the label.

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

Lemma 2.3.1 *(Soundness). If $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$ then $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$.*

Proof *By induction on (the depth) of the inference of $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$ and a case analysis on the last transition rule used.*

- *Using LR1 then $P = \mathbf{tell}(c)$ with $d' = d \sqcup c$. Now the transition $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d \sqcup \alpha \sqcup c\rangle = \langle P', d'\rangle$ follows from the fact that $\alpha = true$ and by applying Rule R1.*

- *Using LR2 then $P = \mathbf{ask}\,(c) \rightarrow P'$, $\alpha \in \min\{a \mid c \sqsubseteq d \sqcup a\}$ and $d' = d \sqcup \alpha$. Now the transition $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d \sqcup \alpha\rangle = \langle P', d'\rangle$ follows from the fact that $c \sqsubseteq d \sqcup \alpha$ and by applying Rule R2.*

- *Using LR3 (a) then $P = Q \mid R$ and $P' = Q' \mid R$, which leads us to $\langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$, by a shorter inference. By appeal to induction then $\langle Q, d \sqcup \alpha\rangle \longrightarrow \langle Q', d'\rangle$. Applying Rule R3(a) to the previous reduction we get $\langle Q \mid R, d \sqcup \alpha\rangle \longrightarrow \langle Q' \mid R, d'\rangle$.*[2]

- *Using LR4' then $P = \exists_x^e Q$, $P' = \exists_x^{e'} Q'$, $\alpha = \alpha'[x/z]$ and $d' = d \sqcup (\exists_x e') \sqcup \alpha'[x/z]$ with $\langle Q, e \sqcup d[z/x]\rangle \xrightarrow{\alpha'} \langle Q', e' \sqcup d[z/x] \sqcup \alpha'\rangle$ by a shorter inference. By appeal to induction then $\langle Q, e \sqcup d[z/x] \sqcup \alpha'\rangle \longrightarrow \langle Q', e' \sqcup d[z/x] \sqcup \alpha'\rangle$. Note that $\alpha' = (\alpha'[x/z])[z/x] = \alpha[z/x]$. Thus, the previous transition is equivalent to $\langle Q, e \sqcup (d \sqcup \alpha)[z/x]\rangle \longrightarrow \langle Q', e' \sqcup (d \sqcup \alpha)[z/x]\rangle$. Using this reduction, the transition $\langle \exists_x^e Q, d \sqcup \alpha\rangle \longrightarrow \langle \exists_x^{e'} Q', d \sqcup (\exists_x e') \sqcup \alpha\rangle$ follows from Rule R4'. Hence $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$.$\square$*

Lemma 2.3.2 *(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 $\alpha \sqcup b = a$, $d'' \sqcup b = d'$.*

[2] Rules LR3 (b) and R3 (b) can be also used instead too.

Proof *The proof proceeds by induction on (the depth) of the inference of $\langle P, d \sqcup a\rangle \longrightarrow \langle P', d'\rangle$ and a case analysis on the last transition Rule used.*

- *Using the Rule R1. Then $P = \mathbf{tell}(c)$, $d' = d \sqcup a \sqcup c$. By Rule LR1 $\langle P, d\rangle \xrightarrow{true} \langle P', d \sqcup c\rangle$. Let $d'' = d \sqcup c$ and $a = b$. We have that $a = \alpha \sqcup b$ and that $d' = d \sqcup a \sqcup c = d \sqcup b \sqcup c = d'' \sqcup b$.*

- *Using the Rule R2. Then $P = \mathbf{ask}\ (c) \rightarrow P'$, $d' = d \sqcup a$ and $c \sqsubseteq d \sqcup a$. Note that $a \in \{a' \in Con_0 | c \sqsubseteq d \sqcup a'\}$ and then, by Proposition 2.1.1, there exists $\alpha \in \min\{a' \in Con_0 | c \sqsubseteq d \sqcup a'\}$ such that $\alpha \sqsubseteq a$. By Rule LR2, $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d \sqcup \alpha\rangle$. Let $d'' = d \sqcup \alpha$ and take $b = a$. Since $a = \alpha \sqcup b$ we have that $\alpha = true$ and then $d' = d \sqcup b = d \sqcup \alpha \sqcup b = d'' \sqcup b$.*

- *Using the Rule R3 (a). Then $P = Q \mid R$, and $P' = Q' \mid R$, which leads us to $\langle Q, d \sqcup a\rangle \longrightarrow \langle Q', d'\rangle$, by a shorter inference. Note that the active process generating this transition could be either an* **ask** *or a* **tell**. *Let suppose that the constraint that has been either asked or told is c. If it is generated by an* **ask** *then $d' = d \sqcup a$ and if it comes from a* **tell** *$d' = d \sqcup a \sqcup c$. Thereafter, by inductive hypothesis, we have that there exist α and b such that*

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

  s.t. $a = \alpha \sqcup b$ and $d' = d'' \sqcup b$. Now by Rule LR3 (a), we have that

  $$\langle Q \mid R, d\rangle \xrightarrow{\alpha} \langle Q' \mid R, d''\rangle^{3}.$$

- *Using the Rule R4'. Then $P = \exists_x^e Q, P' = \exists_x^{e'} Q'$, and $d' = d \sqcup a \sqcup \exists_x e'$ with $\langle Q, e \sqcup (d \sqcup a)[z/x]\rangle \longrightarrow \langle Q', e' \sqcup (d \sqcup a)[z/x]\rangle$ where $z \notin fv(Q) \cup fv(e) \cup fv(d) \cup fv(a)$, by a shorter inference. This transition is equivalent to $\langle Q, (e \sqcup d[z/x]) \sqcup a[z/x]\rangle \longrightarrow \langle Q', (e' \sqcup d[z/x]) \sqcup a[z/x]\rangle$. By induction hypothesis, we have that there exist α and b such that*

  $$\langle Q, e \sqcup d[z/x]\rangle \xrightarrow{\alpha} \langle Q', d''_1\rangle$$

[3] R3 (b) and LR3 (b) can be also used instead.

with $a[z/x] = \alpha \sqcup b$ *and* $e' \sqcup d[z/x] \sqcup a[z/x] = d_1'' \sqcup b$*.*

Note that the active process generating this transition could be either an **ask** *or a* **tell***. If it is generated by an* **ask** *then* $d_1'' = d[z/x] \sqcup e \sqcup \alpha$*. If it is generated by a* **tell***, then* $\alpha = \mathit{true}$ *and* $d_1'' = d[z/x] \sqcup e' \sqcup \alpha$*. Thus in both cases it is safe to assume that* $d_1'' = d[z/x] \sqcup e' \sqcup \alpha$*. Now, note that* $x \notin \mathit{fv}(a[z/x]) = \mathit{fv}(\alpha \sqcup b)$*, and thus* $x \notin \mathit{fv}(\alpha) \cup \mathit{fv}(b)$*. By Rule LR4', we have that*

$$\langle \exists_x^e Q, d\rangle \xrightarrow{\alpha[x/z]} \langle \exists_x^{e'} Q', d \sqcup \exists_x e' \sqcup \alpha[x/z]\rangle.$$

From $a[z/x] = \alpha \sqcup b$*, we have that* $(a[z/x])[x/z] = (\alpha \sqcup b)[x/z]$ *that is* $a = \alpha[x/z] \sqcup b[x/z]$*. Now, take* $d'' = d \sqcup \exists_x e' \sqcup \alpha[x/z]$*. We have that* $d'' \sqcup b[x/z] = d \sqcup \exists_x e' \sqcup \alpha[x/z] \sqcup b[x/z]$ *that by the previous equivalence is equal to* $d \sqcup \exists_x e' \sqcup a$*, that is* d'*.*□

Note that Proposition 2.1.1 is needed for the above proof in the case of Rule R2. This is the reason why in Sec. 2.1.1 we have assumed $(\mathit{Con}, \sqsubseteq)$ to be well-founded.

Corollary 2.3.1 $\langle P, d\rangle \xrightarrow{\mathit{true}} \langle P', d'\rangle$ *if and only if* $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$*.*

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

2.4 Strong and Weak Bisimilarity

Having defined our labeled 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* (LTS), 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 [SR90]) but this would have betrayed the philosophy of "labels as minimal constraints". Then, we have to define bisimilarity as follows.

Definition 2.4.1 (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 [SR90] (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 [LM00] (by identifying contexts with constraints). In [BKM06], it is argued that the equivalence in [LM00] is often over-discriminating. This is also the case of ccp, as illustrated in the following.

Example 2.4.1 *Let $P = \textbf{ask}\ (x < 5)\ \rightarrow\ Q$ and $Q = \textbf{tell}(x < 5)$. The configurations $\gamma_1 = \langle P \mid Q, true \rangle$ and $\gamma_2 = \langle Q \mid Q \mid Q, true \rangle$ are not equivalent according to $\sim_S$. Indeed $\gamma_1 \xrightarrow{x<5} \gamma_1'$, while γ_2 can only perform $\gamma_2 \xrightarrow{true} \gamma_2'$. However $\gamma_1 \sim_o \gamma_2$.* □

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

Definition 2.4.2 (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 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.

After the definition of this new notion of strong bisimilarity we can easily find out a bisimulation relation $\mathcal{R}$ in example 2.4.1 fulfilling our new conditions, i.e., $\mathcal{R}$, a symmetric relation where $\mathcal{R} = \{(\langle P \mid Q, \mathit{true}\rangle, \langle Q \mid Q \mid Q, \mathit{true}\rangle), (\langle P, x < 5\rangle, \langle Q \mid Q, x < 5\rangle), (\langle Q, x < 5\rangle, \langle Q, x < 5\rangle), (\langle \mathbf{stop}, x < 5\rangle, \langle \mathbf{stop}, x < 5\rangle)\}$.

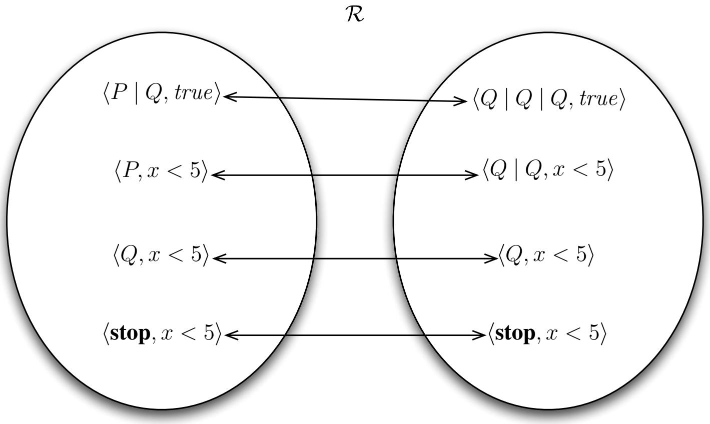

$\mathcal{R}$
$\langle P \mid Q, \mathit{true}\rangle$
$\langle Q \mid Q \mid Q, \mathit{true}\rangle$
$\langle P, x < 5\rangle$
$\langle Q \mid Q, x < 5\rangle$
$\langle Q, x < 5\rangle$
$\langle Q, x < 5\rangle$
$\langle \mathbf{stop}, x < 5\rangle$
$\langle \mathbf{stop}, x < 5\rangle$

Figure 2.3: Strong Bisimulation for Example 2.4.1

Definition 2.4.3 (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 2.4.2 *We can show that* $\mathbf{tell}(\mathit{true}) \dot{\approx} \mathbf{ask}(c) \to \mathbf{tell}(d)$ *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 $\gamma_1 = \langle \mathbf{tell}(\mathit{true}), \mathit{true}\rangle$ and $\gamma_2 = \langle \mathbf{ask}(c) \to \mathbf{tell}(d), \mathit{true}\rangle$. Their labeled transition systems are the following: $\gamma_1 \xrightarrow{\mathit{true}} \langle \mathbf{stop}, \mathit{true}\rangle$ and $\gamma_2 \xrightarrow{c} \langle \mathbf{tell}(d), c\rangle \xrightarrow{\mathit{true}} \langle \mathbf{stop}, c\rangle$. It is now easy to see that the symmetric closure of the relation $\mathcal{R}$ given below is a weak bisimulation.*

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

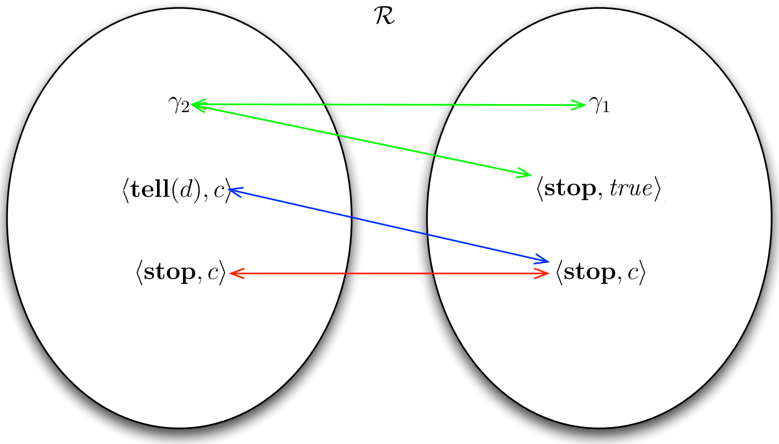

$\mathcal{R}$
γ_2
γ_1
$\langle \mathbf{tell}(d), c\rangle$
$\langle \mathbf{stop}, \mathit{true}\rangle$
$\langle \mathbf{stop}, c\rangle$
$\langle \mathbf{stop}, c\rangle$

Figure 2.4: Weak Bisimulation for Example 2.4.2

The following lemmata will lead us to conclude that strong and weak bisimilarity coincide, respectively, 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 Theorem 2.2.1, also in $\sim_o$). It is worth noticing that any saturated barbed bisimulation (Definition 2.2.3) relating γ_1 and γ_2 is infinite in dimension, since it has to relate $\langle \mathbf{tell}(\mathit{true}), a\rangle$ and $\langle \mathbf{ask}(c) \to \mathbf{tell}(d), a\rangle$ for all constraints $a \in \mathit{Con}_0$. Instead, the relation $\mathcal{R}$ above is finite and it represents (by virtue of the following theorem) a proof also for $\gamma_1 \dot{\approx}_{sb} \gamma_2$.

We start by showing that strong bisimulation ($\dot{\sim}$) is preserved under parallel context.

Lemma 2.4.1 *If $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle$, then $\forall a \in Con_0$, $\langle P, d \sqcup a\rangle \dot{\sim} \langle Q, e \sqcup a\rangle$.*

Proof *Let $\mathcal{R} = \{(\langle P, d \sqcup a\rangle, \langle Q, e \sqcup a\rangle)$ s.t. $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle\}$. We show that $\mathcal{R}$ is a strong bisimulation. We take $(\langle P, d \sqcup a\rangle, \langle Q, e \sqcup a\rangle) \in \mathcal{R}$ and we prove that satisfy conditions (i) and (ii) of Definition 2.4.2.*

(i) By hypothesis $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle$. Since $\langle P, d\rangle \downarrow_d$ then $\langle Q, e\rangle \downarrow_d$, that is, $d \sqsubseteq e$. For the same reason, $e \sqsubseteq d$ and thus $d = e$. So, trivially, $\langle P, d \sqcup a\rangle$ and $\langle Q, e \sqcup a\rangle$ satisfy the same barbs.

(ii) Suppose that $\langle P, d \sqcup a\rangle \xrightarrow{\alpha} \langle P', d'\rangle$. We need to prove that there exist Q' and e' such that $\langle Q, e \sqcup a \sqcup \alpha\rangle \longrightarrow \langle Q', e'\rangle$ and $(\langle P', d'\rangle, \langle Q', e'\rangle) \in \mathcal{R}$.

By Lemma 2.3.1, we have that $\langle P, d \sqcup a \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$. From this, we can obtain a labeled transition of $\langle P, d\rangle$ by using Lemma 2.3.2: $\langle P, d\rangle \xrightarrow{\alpha'} \langle P', d''\rangle$ and there exists b' such that (1) $\alpha' \sqcup b' = a \sqcup \alpha$ and (2) $d'' \sqcup b' = d'$.

From the labeled transition of $\langle P, d\rangle$ and the hypothesis $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle$, we have that $\langle Q, e \sqcup \alpha'\rangle \longrightarrow \langle Q', e''\rangle$ (matching the transition) with $\langle P', d''\rangle \dot{\sim} \langle Q, e''\rangle$(3). Note that by (1) $\langle Q, e \sqcup a \sqcup \alpha\rangle = \langle Q, e \sqcup \alpha' \sqcup b'\rangle$ and that $\langle Q, e \sqcup \alpha' \sqcup b'\rangle \longrightarrow \langle Q, e'' \sqcup b'\rangle$, by monotonicity of the store. Finally, by the definition of $\mathcal{R}$ and (3) we can conclude that $(\langle P', d'' \sqcup b'\rangle, \langle Q', e'' \sqcup b'\rangle) \in \mathcal{R}$ and, by (2), $\langle P', d'' \sqcup b'\rangle = \langle P', d'\rangle$.$\square$

Combining the Lemma above, Soundness (Lemma 2.3.1) and Completeness (Lemma 2.3.2) we can proceed to prove that $\dot{\sim}_{sb} = \dot{\sim}$. We split the two directions of the proof in two lemmas.

Lemma 2.4.2 $\dot{\sim} \subseteq \dot{\sim}_{sb}$

Proof *Let $\mathcal{R} = \{(\langle P, d\rangle, \langle Q, e\rangle)$ s.t $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle\}$. We show that $\mathcal{R}$ is a saturated barbed bisimulation. We take $(\langle P, d\rangle, \langle Q, e\rangle) \in \mathcal{R}$ and we prove that they satisfy the three conditions of Definition 2.2.2.*

(i) Suppose $\langle P, d\rangle \downarrow_c$. Since $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle$ then $\langle Q, e\rangle \downarrow_c$.

(ii) Suppose that $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$. By Corollary 2.3.1 $\langle P, d\rangle \xrightarrow{true} \langle P', d'\rangle$. Since $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle$ then $\langle Q, e \sqcup true\rangle \longrightarrow \langle Q', e'\rangle$ with $\langle P', d'\rangle \dot{\sim} \langle Q', e'\rangle$. Since $e = e \sqcup true$ we have $\langle Q, e\rangle \longrightarrow \langle Q', e'\rangle$ and $(\langle P', d'\rangle, \langle Q', e'\rangle) \in \mathcal{R}$.

(iii) By $\langle P, d\rangle \dot{\sim} \langle Q, e\rangle$ *and Lemma 2.4.1, we have that* $\forall c' \in Con_0$, $(\langle P, d \sqcup c'\rangle, \langle Q, e \sqcup c'\rangle) \in \mathcal{R}$.$\Box$

Lemma 2.4.3 $\dot{\sim}_{sb} \subseteq \dot{\sim}$

Proof *Let* $\mathcal{R} = \{(\langle P, d\rangle, \langle Q, e\rangle)$ *s.t.* $\langle P, d\rangle \dot{\sim}_{sb} \langle Q, e\rangle\}$. *We show that* R *is a strong bisimulation. We take* $(\langle P, d\rangle, \langle Q, e\rangle) \in \mathcal{R}$ *and we prove that they satisfy the two conditions of Definition 2.4.2.*

(i) Suppose $\langle P, d\rangle \downarrow_c$. *Since* $\langle P, d\rangle \dot{\sim}_{sb} \langle Q, e\rangle$ *then* $\langle Q, e\rangle \downarrow_c$.

(ii) Suppose that $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$. *Then by Lemma 2.3.1* $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$. *Since* $\langle P, d\rangle \dot{\sim}_{sb} \langle Q, e\rangle$ *then* $\langle Q, e{\sqcup}\alpha\rangle \longrightarrow \langle Q', e'\rangle$ *with* $\langle P', d'\rangle \dot{\sim}_{sb} \langle Q', e'\rangle$. *Then* $(\langle P', d'\rangle, \langle Q', e'\rangle) \in \mathcal{R}$.$\Box$

Consequently, as a corollary we have shown that strong bisimilarity coincides with the strong saturated barbed bisimilarity

Theorem 2.4.1 $\dot{\sim}_{sb} = \dot{\sim}$.

Proof *Using Lemma 2.4.2 and Lemma 2.4.3*

In order to prove that $\dot{\approx} = \dot{\approx}_{sb}$, we essentially use the same proof-scheme of the strong case ($\dot{\sim} = \dot{\sim}_{sb}$). The main difference concerns two technical lemmata (namely Lemma 2.4.4 and Lemma 2.4.6) stating that weak barbs are preserved by the addition of constraints to the store (this was trivial for the strong case).

Lemma 2.4.4 *Given* $\langle P, d\rangle$ *and* $\langle Q, e\rangle$ *such that* $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, *if* $\forall a \in Con_0$ $\langle P, d \sqcup a\rangle \downarrow_c$ *then* $\langle Q, e \sqcup a\rangle \Downarrow_c$.

Proof If $\langle P, d \sqcup a\rangle \downarrow_c$, then $c \sqsubseteq d \sqcup a$. Since $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, then there exists a $\langle Q', e'\rangle$ such that $\langle Q, e\rangle \longrightarrow^* \langle Q', e'\rangle$ and $d \sqsubseteq e'$. Moreover $\langle Q, e \sqcup a\rangle \longrightarrow^* \langle Q', e' \sqcup a\rangle$, because all reductions are preserved by the addition of constraints. Finally $c \sqsubseteq d \sqcup a \sqsubseteq e' \sqcup a$, that means $\langle Q', e' \sqcup a\rangle \downarrow_c$, i.e., $\langle Q, e \sqcup a\rangle \Downarrow_c$.

With the above lemma, we can use the same technique of Lemma 2.4.1 to prove that $\dot{\approx}$ is a congruence.

Lemma 2.4.5 *If* $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$ *then* $\forall a \in Con_0,\ \langle P, d \sqcup a\rangle \dot{\approx} \langle Q, e \sqcup a\rangle$.

Proof *We take the relation* $\mathcal{R} = \{(\langle P, d \sqcup a\rangle, \langle Q, e \sqcup a\rangle)$ *s.t.* $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle\}$ *and we prove that it is a weak bisimulation.*

(i) Suppose $\langle P, d \sqcup a\rangle \downarrow_c$. *Since* $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, *by Lemma 2.4.4, then* $\langle Q, e \sqcup a\rangle \Downarrow_c$.

(ii) Suppose $\langle P, d \sqcup a\rangle \xrightarrow{\alpha} \langle P', d'\rangle$.
By Lemma 2.3.1 $\langle P, d \sqcup a \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$.
By Lemma 2.3.2 $\langle P, d\rangle \xrightarrow{\beta} \langle P', d''\rangle$ *and there exists* b *such that* $\beta \sqcup b = a \sqcup \alpha$ *and* $d'' \sqcup b = d'$. *Since* $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, *then* $\langle Q, e \sqcup \beta\rangle \longrightarrow^* \langle Q', e''\rangle$ *with* $\langle P', d''\rangle \dot{\approx} \langle Q', e''\rangle$. *Note that all reductions are preserved when adding constraints to the store, therefore from* $\langle Q, e \sqcup \beta\rangle \longrightarrow^* \langle Q', e''\rangle$ *we can derive that* $\langle Q, e \sqcup \beta \sqcup b\rangle \longrightarrow^* \langle Q', e'' \sqcup b\rangle$. *This means that* $\langle Q, e \sqcup a \sqcup \alpha\rangle \longrightarrow^* \langle Q', e'' \sqcup b\rangle$. *Now we have* $\langle P', d'\rangle = \langle P', d'' \sqcup b\rangle$ *and* $(\langle P', d'' \sqcup b\rangle, \langle Q', e'' \sqcup b\rangle) \in \mathcal{R}$, *because* $\langle P', d''\rangle \dot{\approx} \langle Q', e''\rangle$.

The following lemma extends Lemma 2.4.4 to the case of weak barbs.

Lemma 2.4.6 *Given* $\langle P, d\rangle$ *and* $\langle Q, e\rangle$ *such that* $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, *if* $\forall a \in Con_0$ $\langle P, d \sqcup a\rangle \Downarrow_c$ *then* $\langle Q, e \sqcup a\rangle \Downarrow_c$.

Proof If $\langle P, d \sqcup a\rangle \Downarrow_c$, then there are two possibilities:

(i) $\langle P, d \sqcup a\rangle \downarrow_c$. The result follows by Lemma 2.4.4.

(ii) $\langle P, d \sqcup a\rangle \not\downarrow_c$ and $\langle P, d \sqcup a\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow^* \langle P_n, d_n\rangle \downarrow_c$. From $\langle P, d \sqcup a\rangle \longrightarrow \langle P_1, d_1\rangle$ and by Lemma 2.3.2 we have $a = \beta \sqcup b$ such that $\langle P, d\rangle \xrightarrow{\beta} \langle P_1, d_1'\rangle$ and $d_1' \sqcup b = d_1$. Since $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, then $\langle Q, e \sqcup \beta\rangle \longrightarrow^* \langle Q_1, e_1'\rangle$ with $\langle P_1, d_1'\rangle \dot{\approx} \langle Q_1, e_1'\rangle$. By Lemma 2.4.5, $\langle P_1, d_1\rangle = \langle P_1, d_1' \sqcup b\rangle \dot{\approx} \langle Q_1, e_1' \sqcup b\rangle$ and thus $\langle Q_1, e_1' \sqcup b\rangle \longrightarrow^* \langle Q_n, e_n\rangle \downarrow_c$. By putting all our pieces together, we have $\langle Q, e \sqcup a\rangle = \langle Q, e \sqcup \beta \sqcup b\rangle \longrightarrow^* \langle Q_1, e_1' \sqcup b\rangle \longrightarrow^* \langle Q_n, e_n\rangle \downarrow_c$, i.e., $\langle Q, e \sqcup a\rangle \Downarrow_c$.

We have now all the ingredients to prove that $\dot{\approx} = \dot{\approx}_{sb}$.

Lemma 2.4.7 *If $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, then $\langle P, d\rangle \dot{\approx}_{sb} \langle Q, e\rangle$.*

Proof *We take the relation $\mathcal{R} = \{(\langle P, d\rangle, \langle Q, e\rangle) \mid \langle P, d\rangle \dot{\approx} \langle Q, e\rangle\}$ and we prove that $\mathcal{R}$ is a weak saturated barbed bisimulation (Definition 2.2.3).*

(i) Suppose $\langle P, d\rangle \Downarrow_c$. Since $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$ then by Lemma 2.4.6, $\langle Q, e\rangle \Downarrow_c$.

(ii) Suppose $\langle P, d\rangle \longrightarrow^* \langle P', d'\rangle$. By definition of $\longrightarrow^*$, there exist $\langle P_1, d_1\rangle$, $\langle P_2, d_2\rangle$, ... , $\langle P_n, d_n\rangle$ such that

$$\langle P, d\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \langle P_2, d_2\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \langle P', d'\rangle$$

which means that

$$\langle P, d\rangle \xrightarrow{true} \langle P_1, d_1\rangle \xrightarrow{true} \langle P_2, d_2\rangle \xrightarrow{true} \ldots \xrightarrow{true} \langle P_n, d_n\rangle \xrightarrow{true} \langle P', d'\rangle.$$

Now, since $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$, then $\langle Q, e\rangle = \langle Q, e \sqcup true\rangle \longrightarrow^* \langle Q_1, e_1\rangle$ and $\langle P_1, d_1\rangle \dot{\approx} \langle Q_1, e_1\rangle$. By iterating this reasoning one have that

$$\langle Q, e\rangle \longrightarrow^* \langle Q_1, e_1\rangle \longrightarrow^* \langle Q_2, e_2\rangle \longrightarrow^* \ldots \longrightarrow^* \langle Q_n, e_n\rangle \longrightarrow^* \langle Q', e'\rangle$$

with $\langle P', d'\rangle \dot{\approx} \langle Q', e'\rangle$.

Summarizing $\langle Q, e\rangle \longrightarrow^* \langle Q', e'\rangle$ and $(\langle P', d'\rangle, \langle Q', e'\rangle) \in \mathcal{R}$.

(iii) $\forall a \in Con_0(\langle P, d \sqcup a\rangle, \langle Q, e \sqcup a\rangle) \in \mathcal{R}$, by Lemma 2.4.5.

Lemma 2.4.8 *If $\langle P, d\rangle \dot{\approx}_{sb} \langle Q, e\rangle$ then $\langle P, d\rangle \dot{\approx} \langle Q, e\rangle$.*

Proof *We take the relation $\mathcal{R} = \{(\langle P, d\rangle, \langle Q, e\rangle)$ s.t. $\langle P, d\rangle \dot{\approx}_{sb} \langle Q, e\rangle\}$ and we prove that it is a weak bisimulation (Definiton 2.4.3).*

(i) Suppose $\langle P, d\rangle \downarrow_c$. Then $\langle P, d\rangle \Downarrow_c$. Since $\langle P, d\rangle \dot{\approx}_{sb} \langle Q, e\rangle$, then $\langle Q, e\rangle \Downarrow_c$.

(ii) Suppose that $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$. By Lemma 2.3.1 $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$. By Definition of $\longrightarrow^*$, we can say that $\langle P, d \sqcup \alpha\rangle \longrightarrow^* \langle P', d'\rangle$. Since $\langle P, d\rangle \dot{\approx}_{sb} \langle Q, e\rangle$ we have $\langle Q, e \sqcup \alpha\rangle \longrightarrow^* \langle Q', e'\rangle$ with $\langle P', d'\rangle \dot{\approx}_{sb} \langle Q', e'\rangle$.

Theorem 2.4.2 $\dot{\approx}_{sb} = \dot{\approx}$.

Proof *Using Lemma 2.4.7 and Lemma 2.4.8*

2.5 Summary of Contributions and Related Work

In this chapter we provided a labeled transition semantics and a novel notion of labeled bisimilarity for ccp by building upon the work in [SR90, BGM09]. 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 [SRP91] 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. Furthermore, our characterization in terms of bisimilarity of the standard observable behaviour of ccp processes avoids complicated notions such as fairness and infinite elements.

In [BKM06, BGM09], it is argued that the standard notion of bisimilarity is often too fine grained and an alternative, coarser, definition of bisimilarity is provided. Intuitively, in the bisimulation game, each move (transition) $P \xrightarrow{C} P'$, has to be matched it with a move $C[Q] \longrightarrow Q'$.

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 derived a labeled transition system for ccp by exploiting the intuition of [Sew98, LM00]. 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 [SR90] but the recent developments in [BGM09] 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 [LM00]. As for the bisimilarity in [SR90], it is too fine grained, i.e., it separates processes which are indistinguishable. Instead, the notion of bisimulation from [BGM09] (instantiated to the case of ccp) is fully abstract with respect to the standard observational equivalence given in [SRP91]. Our work can therefore be also regarded as a compelling application of the theory of reactive systems.

Chapter 3

Partition Refinement for Bisimilarity in CCP

Divide et impera.
– Gaius Julius Caesar.

Right, ok, let's take out our scissors and refashion this.
– Paris C. Kanellakis.

In the previous chapter we gave an overview of concurrency and present the main insights of ccp, its operational semantics, and standard notion of bisimilarity for this language. Furthermore, we introduced labeled semantics and a new notion of labeled bisimilarity for ccp which is not over discriminative. In this chapter, we implement an algorithm which automatically calculates the well-behaved notion of bisimilarity for ccp presented in Definition 2.4.2 in Section 2.4.

When the state space of a system is finite, the ordinary notion of (strong) bisimilarity can be computed via the well-known partition refinement algorithm, but unfortunately, this algorithm does not work for our new definition of (strong) bisimilarity for ccp. Thus, in this chapter we propose a variation of the partition refinement algorithm for verifying ccp (strong) bisimilarity by borrowing some concepts involved in the Minimization Algorithm for Symbolic Bisimilarity [BM09].

3.1 Background

3.1.1 Partition Refinement

In this section we recall the partition refinement algorithm introduced in [KS83] for checking bisimilarity over the states of an LTS. Recall that an LTS can be intuitively seen as a graph where nodes represent states (of a computation) and arcs represent transitions between states. A transition $P \xrightarrow{a} Q$ between P and Q labeled with a can be typically thought of as an evolution from P to Q provided that a condition a is met. Transition systems can be used to represent the evolution of processes in calculi such as CCS and the π-calculus [Mil80, Mil99]. In this case states correspond to processes and transitions are given by the operational semantics of the respective calculus.

Let us now introduce some notation. Given a set S, a *partition* of S is a set of *blocks*, i.e., subsets of S, that are all disjoint and whose union is S. We write $\{B_1\} \dots \{B_n\}$ to denote a partition consisting of non empty blocks $B_1, \dots, B_n$. A partition represents an equivalence relation where equivalent elements belong to the same block. We write $P\mathcal{P}Q$ to mean that P and Q are equivalent in the partition $\mathcal{P}$.

The partition refinement algorithm (see Algorithm 3.1.1) checks the bisimilarity of a set of initial states IS as follows. First, it computes $IS^\star$, that is the set of all states that are reachable from IS. Then it creates the partition $\mathcal{P}^0$ where all the elements of $IS^\star$ belong to the same block (i.e., they are all equivalent). After the initialization, it iteratively refines the partitions by employing the function $\mathbf{F}$, defined as follows: for all partitions $\mathcal{P}$, $P \, \mathbf{F}(\mathcal{P}) \, Q$ if

- if $P \xrightarrow{a} P'$ then exists Q' s.t. $Q \xrightarrow{a} Q'$ and $P' \mathcal{P} Q'$.

The algorithm terminates whenever two consecutive partitions are equivalent. In such partition two states belong to the same block iff they are bisimilar.

Note that any iteration splits blocks and never fuses them. For this reason if $IS^\star$ is finite, the algorithm terminates in at most $|IS^\star|$ iterations.

Proposition 3.1.1 *If $IS^\star$ is finite, then the algorithm terminates and the resulting partition equates all and only the bisimilar states.*

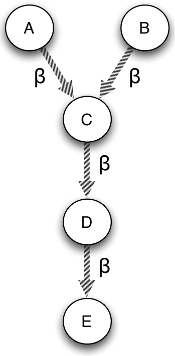
Algorithm 3.1.1 `Partition-Refinement(`IS`)`

Initialization

1. $IS^\star$ is the set of all processes reachable from IS,
2. $\mathcal{P}^0 := \{IS^\star\}$,

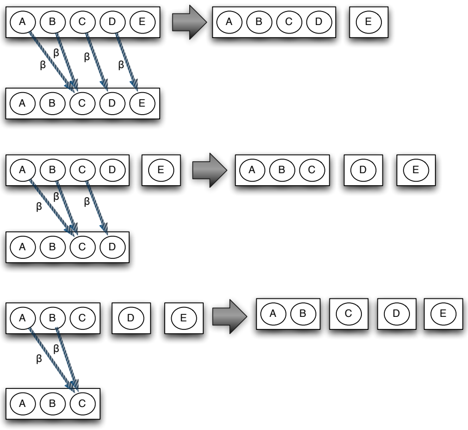
Iteration $\mathcal{P}^{n+1} := \mathbf{F}(\mathcal{P}^n)$,
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.

Example 3.1.1 *Let us take the following LTS:*



A Labelled Transition System

According to Algorithm 3.1.1 after the initialization process $IS^\star = \{A, B, C, D, E\}$ *and the first partition* $\mathcal{P}^0 = \{A, B, C, D, E\}$. *Afterwards, it iteratively refines partitions by employing the function* $\mathbf{F}(\mathcal{P})$ *where the iteration proceeds as follows:*



Splitting the Blocks

Therefore, the algorithm terminates whenever two consecutive partitions are equivalent, so finally $\mathcal{P}^n = \{\{A, B\}, \{C\}, \{D\}, \{E\}\}$.

3.2 Irredundant Bisimilarity

For purposes related to the clarity of examples concerned with the way the modified partition refinement for ccp works (in particular, finding out redundant transitions), from now on we will consider a non-determinist fragment of ccp. (with this operator we can find redundant transitions and at the same time avoid complicated automata made from a parallel composition operator). Therefore we shall take into account the following syntax and reduction and labeled rules for ccp processes in this fragment:

Syntax for non-deterministic ccp. We extend the syntax in Definition 2.1.2 in Section 2.1.2 with the summation process of the form $P + Q$, i.e., the syntax is now given as $P, Q ::= ... \mid P + Q$

3.2.1 Reduction Semantics for Non-deterministic CCP

We use all rules in Table 2.1 plus the following rules:

$$\text{R6 (a)} \quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P + Q, d\rangle \longrightarrow \langle P' + Q, d'\rangle} \qquad \text{R6 (b)} \quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle Q + P, d\rangle \longrightarrow \langle Q + P', d'\rangle}$$

3.2.2 Labeled Semantics for Non-deterministic CCP

We recall all rules in Table 2.2 and we add the following rules:

$$\text{LR6 (a)} \quad \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P + Q, d\rangle \xrightarrow{\alpha} \langle P' + Q, d'\rangle} \qquad \text{LR6 (b)} \quad \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle Q + P, d\rangle \xrightarrow{\alpha} \langle Q + P', d'\rangle}$$

3.2.3 Saturated Bisimilarity in a Non-deterministic CCP Fragment

As said in Section 2.2 an elegant solution to modify bisimilarity for obtaining a congruence consists in saturated bisimilarity (see Definition 2.2.2 in the aforementioned section). 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. Thus, now by introducing a non-deterministic fragment of ccp we can present some new interesting, nicer and much more simpler examples which are important for describing this particular notion in the case of this language. We shall emphasize that there is no standard semantics. In this case, we must also punctuate that the closure operators described in Section 2 only hold for the deterministic fragment of ccp.

Example 3.2.1 *Take $T = \mathbf{tell}(true)$, $P = \mathbf{ask}\ (x < 7) \rightarrow T$ and $Q = \mathbf{ask}\ (x < 5) \rightarrow T$. You can see that $\langle P, true\rangle \not\dot{\sim}_{sb} \langle Q, true\rangle$, since $\langle P, x < 7\rangle \longrightarrow$, while $\langle Q, x < 7\rangle \not\longrightarrow$. Consider now the configuration $\langle P + Q, true\rangle$ and observe that $\langle P + Q, true\rangle \dot{\sim}_{sb} \langle P, true\rangle$. Indeed, for all constraints e s.t. $x < 7 \sqsubseteq e$, both configurations evolve into $\langle T, e\rangle$, while for all e s.t. $x < 7 \not\sqsubseteq e$, both configurations cannot proceed. Since $x < 7 \sqsubseteq x < 5$, the behaviour of Q is somehow absorbed by the behaviour of P.*

Example 3.2.2 *Since $\dot{\sim}_{sb}$ is upward closed, $\langle P + Q, z < 5\rangle \dot{\sim}_{sb} \langle P, z < 5\rangle$ follows immediately by the previous example. Now take $R = \mathbf{ask}\ (z < 5) \rightarrow (P + Q)$ and $S = \mathbf{ask}\ (z < 7) \rightarrow P$. By analogous arguments of the previous example, one can show that $\langle R + S, true\rangle \dot{\sim}_{sb} \langle S, true\rangle$.*

Example 3.2.3 *Take* $T' = \mathbf{tell}(y = 1)$*,* $Q' = \mathbf{ask}\ (x < 5) \rightarrow T'$ *and* $R' = \mathbf{ask}\ (z < 5) \rightarrow P + Q'$*. Observe that* $\langle P + Q', z < 5\rangle \not\sim_{sb} \langle P, z < 5\rangle$ *and that* $\langle R' + S, true\rangle \dot{\not\sim}_{sb} \langle S, true\rangle$*, since* $\langle P + Q', x < 5\rangle$ *and* $\langle R' + S, true\rangle$ *can reach a store containing the constraint* $y = 1$*.*

3.2.4 Soundness and Completeness

The labeled semantics can be related to the reduction semantics via the two following lemmata, namely soundness and completeness. The first one deals with the fact that if a process is able to perform an action using some (minimal) information from the environment, then providing such information to the process (in its store) will allow to perform a reduction instead.

Lemma 3.2.1 (Soundness of $\longrightarrow$) *If* $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ *then* $\langle P, c \sqcup \alpha\rangle \longrightarrow \langle P', c'\rangle$*.*

Proof *By induction on (the depth) of the inference of* $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$*. Here we consider just the additional case for the non-deterministic choice and refer the reader to Lemma 2.3.1 in Section 2.3 for further cases.*

- *Using rule LR6 (a) then* $P = Q + R$ *and* $P' = Q'$ *which lead us to* $\langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ *by a shorter inference. By appeal to induction then* $\langle Q, d \sqcup a\rangle \longrightarrow \langle Q', d'\rangle$*. Applying Rule R6 (a) to the previous reduction we get* $\langle Q + R, d\rangle \longrightarrow \langle Q', d'\rangle$*.*[1]

Now completeness tries to do the opposite, if a process can perform a reduction then the idea is that one could use a piece of the store as a label, and the process should be able to arrive to the same result by means of a labeled transition. In this case, such label might not be exactly the piece of information we took from the store but something smaller, thus the result could also be smaller.

Lemma 3.2.2 (Completeness of $\longrightarrow$) *If* $\langle P, c \sqcup a\rangle \longrightarrow \langle P', c'\rangle$ *then there exists* α *and* b *s.t.* $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c''\rangle$ *where* $\alpha \sqcup b = a$ *and* $c'' \sqcup b = c'$

[1] LR6 (b) and R6 (b) can be used instead.

Proof *The proof proceeds by induction on (the depth) of the inference of $\langle P, d \sqcup a\rangle \longrightarrow \langle P', d'\rangle$. As in Lemma 3.2.1 we consider just the case for the non-deterministic choice.*

- *Using rule R6 (a) then $P = Q + R$ and $P' = Q'$ which lead us to $\langle Q, d \sqcup a\rangle \longrightarrow \langle Q', d'\rangle$ by a shorter inference. Note that the active process generating the transition could be either an* $\mathbf{ask}$ *or a* $\mathbf{tell}$. *Let suppose that the constraint that has been either asked or told is c. If it is generated by an* $\mathbf{ask}$ *then $d' = d \sqcup a$ and if it comes from a* $\mathbf{tell}$ *$d' = d \sqcup a \sqcup c$. Thereafter, by inductive hypothesis, we have that there exist α and b such that*

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

  st $a = \alpha \sqcup b$ and $d' = d'' \sqcup b$. Now by rule LR6 (a), we have that

  $$\langle Q + R, d\rangle \xrightarrow{\alpha} \langle Q' + R, d''\rangle^{2}.$$

3.2.5 Syntactic Bisimilarity and Redundancy

Here, we first consider the notion of Syntactic Bisimilarity (see Definition 2.4.1 in Section 2.4) since it is the obvious adaptation of the ordinary notion of bisimilarity (see [Mil80]) for the labeled semantics in ccp, which itself can be computed via the standard partition refinement algorithm [KS83].
Unfortunately syntactic bisimilarity, as we have already said in Section 2.4 in Chapter 2, is over-discriminating. This definition is too fine-grained because of some *redundant transitions*. Therefore we then try taking $\dot{\sim}_{sb}$. But in this case the main problem for verifying this notion is the quantification over all contexts. This problem is addressed following the abstract approach in [BM09]. More precisely, we use an equivalent notion, namely *irredundant bisimilarity* $\dot{\sim}_{I}$, which can be verified with the ccp partition refinement. As its name suggests, $\dot{\sim}_{I}$ only takes into account those transitions deemed irredundant. However, technically speaking, going from $\dot{\sim}_{sb}$ to $\dot{\sim}_{I}$ requires one intermediate notion, so-called *symbolic bisimilarity*. These three notions are shown to be equivalent, i.e., $\dot{\sim}_{sb} = \dot{\sim}_{sym} = \dot{\sim}_{I}$. In

[2] Both rules R6 (b) and LR6 (b) can be used instead in this proof

the following we recall all of them.

Let us first give some auxiliary definitions. The first concept is that of *derivation*. Consider the following transitions (taken from Figure 3.1):

(a) $\langle P{+}Q, z < 5\rangle \xrightarrow{x<7} \langle T, z < 5{\sqcup}x < 7\rangle$ (b) $\langle P{+}Q, z < 5\rangle \xrightarrow{x<5} \langle T, z < 5{\sqcup}x < 5\rangle$

Transition (a) means that for all constraints e s.t. $x < 7$ is entailed by e (formally $x < 7 \sqsubseteq e$), the transition (c) $\langle P + Q, z < 5 \sqcup e\rangle \longrightarrow \langle T, z < 5 \sqcup e\rangle$ can be performed, while transition (b) means that the reduction (c) is possible for all e s.t. $x < 5 \sqsubseteq e$. Since $x < 7 \sqsubseteq x < 5$, transition (b) is "redundant", in the sense that its meaning is "logically derived" by transition (a).

The following notion captures the above intuition:

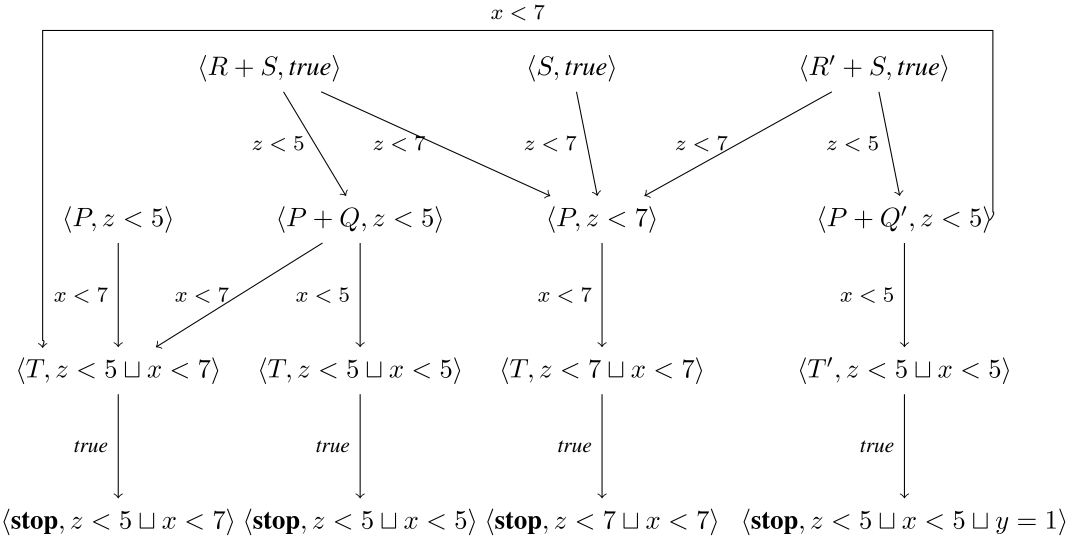
Definition 3.2.1 (Derivation) *We say that the transition* $t = \langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle$ *derives* $t' = \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$*, written* $\langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle \vdash_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$ *(*$t \vdash_D t'$ *), if there exists* e *s.t. the following conditions hold:*

(i) $\beta = \alpha \sqcup e$ *(ii)* $c'' = c' \sqcup e$

Figure 3.1 illustrates the LTSs of our running example 3.2.2.

$$\begin{array}{rcl} T = \mathbf{tell}(\mathit{true}) & \quad & P = \mathbf{ask}\ (x < 7)\ \rightarrow\ T \\ T' = \mathbf{tell}(y = 1) & \quad & S = \mathbf{ask}\ (z < 7)\ \rightarrow\ P \end{array}$$

$$\begin{array}{rr} Q = \mathbf{ask}\ (x < 5)\ \rightarrow\ T & R = \mathbf{ask}\ (z < 5)\ \rightarrow\ (P + Q) \\ Q' = \mathbf{ask}\ (x < 5)\ \rightarrow\ T' & R' = \mathbf{ask}\ (z < 5)\ \rightarrow\ (P + Q') \end{array}$$



$x < 7$
$\langle R + S, \mathit{true}\rangle$
$\langle S, \mathit{true}\rangle$
$\langle R' + S, \mathit{true}\rangle$
$z < 5$
$z < 7$
$z < 7$
$z < 7$
$z < 5$
$\langle P, z < 5\rangle$
$\langle P + Q, z < 5\rangle$
$\langle P, z < 7\rangle$
$\langle P + Q', z < 5\rangle$
$x < 7$
$x < 7$
$x < 5$
$x < 7$
$x < 5$
$\langle T, z < 5 \sqcup x < 7\rangle$
$\langle T, z < 5 \sqcup x < 5\rangle$
$\langle T, z < 7 \sqcup x < 7\rangle$
$\langle T', z < 5 \sqcup x < 5\rangle$
true
true
true
true
$\langle \mathbf{stop}, z < 5 \sqcup x < 7\rangle$
$\langle \mathbf{stop}, z < 5 \sqcup x < 5\rangle$
$\langle \mathbf{stop}, z < 7 \sqcup x < 7\rangle$
$\langle \mathbf{stop}, z < 5 \sqcup x < 5 \sqcup y = 1\rangle$

Figure 3.1: The Labeled Transition System of a Running Example.

$$
\begin{aligned}
\mathcal{P}^0 &= \{\langle R' + S, \mathit{true}\rangle, \langle S, \mathit{true}\rangle, \langle R + S, \mathit{true}\rangle\},\\
&\quad \{\langle P + Q', z < 5\rangle, \langle P + Q, z < 5\rangle, \langle P, z < 5\rangle\}, \{\langle P, z < 7\rangle\},\\
&\quad \{\langle T', z < 5 \sqcup x < 5\rangle, \langle T, z < 5 \sqcup x < 5\rangle, \langle \mathbf{stop}, z < 5 \sqcup x < 5\rangle\},\\
&\quad \{\langle T, z < 7 \sqcup x < 7\rangle, \langle \mathbf{stop}, z < 7 \sqcup x < 7\rangle\},\\
&\quad \{\langle T, z < 5 \sqcup x < 7\rangle, \langle \mathbf{stop}, z < 5 \sqcup x < 7\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 5 \sqcup y = 1\rangle\}\\
\mathcal{P}^1 &= \{\langle R' + S, \mathit{true}\rangle, \langle S, \mathit{true}\rangle, \langle R + S, \mathit{true}\rangle\},\\
&\quad \{\langle P + Q', z < 5\rangle, \langle P + Q, z < 5\rangle, \langle P, z < 5\rangle\}, \{\langle P, z < 7\rangle\},\\
&\quad \{\langle T', z < 5 \sqcup x < 5\rangle\}, \{\langle T, z < 5 \sqcup x < 5\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 5\rangle\},\\
&\quad \{\langle T, z < 7 \sqcup x < 7\rangle\}, \{\langle \mathbf{stop}, z < 7 \sqcup x < 7\rangle\}, \{\langle T, z < 5 \sqcup x < 7\rangle\},\\
&\quad \{\langle \mathbf{stop}, z < 5 \sqcup x < 7\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 5 \sqcup y = 1\rangle\}\\
\mathcal{P}^2 &= \{\langle R' + S, \mathit{true}\rangle, \langle S, \mathit{true}\rangle, \langle R + S, \mathit{true}\rangle\}, \{\langle P + Q', z < 5\rangle\},\\
&\quad \{\langle P + Q, z < 5\rangle, \langle P, z < 5\rangle\}, \{\langle P, z < 7\rangle\}, \{\langle T', z < 5 \sqcup x < 5\rangle\},\\
&\quad \{\langle T, z < 5 \sqcup x < 5\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 5\rangle\}, \{\langle T, z < 7 \sqcup x < 7\rangle\},\\
&\quad \{\langle \mathbf{stop}, z < 7 \sqcup x < 7\rangle\}, \{\langle T, z < 5 \sqcup x < 7\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 7\rangle\},\\
&\quad \{\langle \mathbf{stop}, z < 5 \sqcup x < 5 \sqcup y = 1\rangle\}\\
\mathcal{P}^3 &= \{\langle R' + S, \mathit{true}\rangle\}, \{\langle S, \mathit{true}\rangle, \langle R + S, \mathit{true}\rangle\}, \{\langle P + Q', z < 5\rangle\},\\
&\quad \{\langle P + Q, z < 5\rangle, \langle P, z < 5\rangle\}, \{\langle P, z < 7\rangle\}, \{\langle T', z < 5 \sqcup x < 5\rangle\},\\
&\quad \{\langle T, z < 5 \sqcup x < 5\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 5\rangle\}, \{\langle T, z < 7 \sqcup x < 7\rangle\},\\
&\quad \{\langle \mathbf{stop}, z < 7 \sqcup x < 7\rangle\}, \{\langle T, z < 5 \sqcup x < 7\rangle\}, \{\langle \mathbf{stop}, z < 5 \sqcup x < 7\rangle\},\\
&\quad \{\langle \mathbf{stop}, z < 5 \sqcup x < 5 \sqcup y = 1\rangle\}\\
\mathcal{P}^4 &= \mathcal{P}^3
\end{aligned}
$$

Figure 3.2: Partitions Computed by the CCP-Partition-Refinement Algorithm.

One can verify in the above example that (a) $\vdash_D$ (b), and notice that both transitions arrive at the same process P': The difference lies in the label and the store. Now imagine the situation where the initial configuration is able to perform another transition with β (as in t'), and let us also assume that such transition arrives at a configuration which is equivalent to the result of t'. Therefore, it is natural to think that, since t derives t', such new transition should also be derivated by t. Let us explain with an example. Consider the two following transitions:

(e) $\langle R+S, true\rangle \xrightarrow{z<7} \langle P, z<7\rangle$ (f) $\langle R+S, true\rangle \xrightarrow{z<5} \langle P+Q, z<5\rangle$

Note that transition (f) cannot be derived by other transitions, since (e) $\nvdash_D$ (f). Indeed, P is syntactically different from $P{+}Q$, even if they have the same behaviour when inserted in the store $z<5$, i.e., $\langle P, z<5\rangle \dot{\sim}_{sb} \langle P+Q, z<5\rangle$ (since $\dot{\sim}_{sb}$ is upward closed). Transition (f) is also "redundant", since its behaviour "does not add anything" to the behavior of (e). The following definition encompasses this situation:

Definition 3.2.2 (Derivation w.r.t $\mathcal{R}$, $\vdash_{\mathcal{R}}$) *We say that the transition $t = \gamma \xrightarrow{\alpha} \gamma_1$ derives $t' = \gamma \xrightarrow{\beta} \gamma_2$ w.r.t. to $\mathcal{R}$ (written $t \vdash_{\mathcal{R}} t'$) if there exists γ_2' s.t. $t \vdash_D \gamma \xrightarrow{\beta} \gamma_2'$ and $\gamma_2' \mathcal{R} \gamma_2$.*

Then, when $\mathcal{R}$ represents some sort of equivalence, this notion will capture the situation above mentioned. Notice that $\vdash_D$ is $\vdash_{\mathcal{R}}$ with $\mathcal{R}$ being the identity relation (id). Now we introduce the concept of *domination*, which consists in strengthening the notion of derivation by requiring labels to be different.

Definition 3.2.3 (Domination $\succ_D$) *We say that the transition $t = \langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ dominates $t' = \langle P, c\rangle \xrightarrow{\beta} \langle P', c''\rangle$ (written $t \succ_D t'$) if $t \vdash_D t'$ and $\alpha \neq \beta$. In other terms: (i) $\beta = \alpha \sqcup e$ (ii) $c'' = c' \sqcup e$ (iii) $\alpha \neq \beta$*

Similarly, as we did for derivation, we can define domination depending on a relation. Again, $\succ_D$ is just $\succ_{\mathcal{R}}$ when $\mathcal{R}$ is the identity relation (id).

Definition 3.2.4 (Redundancy and Domination w.r.t $\mathcal{R}$, $\succ_{\mathcal{R}}$) *We say that the transition $t = \langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ dominates $t' = \langle P, c\rangle \xrightarrow{\beta} \langle Q, d\rangle$ w.r.t. to $\mathcal{R}$ (written $t \succ_{\mathcal{R}} t'$) if there exists c'' s.t. $t \succ_D \langle P, c\rangle \xrightarrow{\beta} \langle P', c''\rangle$ and $\langle P', c''\rangle \mathcal{R} \langle Q, d\rangle$. Also,* *a transition is said to be redundant when it is dominated by another, otherwise it is said to be irredundant.*

Definition 3.2.5 *We say that a relation $\mathcal{R} \subseteq Conf \times Conf$ is closed under the addition of constraints if whenever $\langle P, c\rangle \mathcal{R} \langle Q, d\rangle$ then for all $e \in Con_0$ we have that $\langle P, c \sqcup e\rangle \mathcal{R} \langle Q, d \sqcup e\rangle$*

In other words, if a pair of configurations belong to the relation, then any pair obtained by adding more information in both stores should also be in the relation. We can now prove that under certain conditions $\vdash_{\mathcal{R}}$ and $\succ_{\mathcal{R}}$ are a partial order and a strict partial order respectively.

Proposition 3.2.1 *If $\mathcal{R}$ is reflexive and closed under the addition of constraints then $\vdash_{\mathcal{R}}$ is a partial order.*

Proof *We will proceed by showing that $\vdash_{\mathcal{R}}$ is reflexive, transitive and antisymmetric. Let $t_1 = \langle P, c\rangle \xrightarrow{\alpha_1} \langle P_1, c_1\rangle$, $t_2 = \langle P, c\rangle \xrightarrow{\alpha_2} \langle P_2, c_2\rangle$ and $t_3 = \langle P, c\rangle \xrightarrow{\alpha_3} \langle P_3, c_3\rangle$ be transitions s.t. $t_1 \vdash_{\mathcal{R}} t_2$ and $t_2 \vdash_{\mathcal{R}} t_3$.*

- *(Reflexivity) We need to prove that $t_1 \vdash_{\mathcal{R}} t_1$. First notice that $t_1 \vdash_D t_1$ just by taking $e = true$ and since $\mathcal{R}$ is reflexive then $t_1 \vdash_{\mathcal{R}} t_1$.*

- *(Transitive) We need to prove that $t_1 \vdash_{\mathcal{R}} t_3$. Since $t_1 \vdash_{\mathcal{R}} t_2$ we know there exists $t_2' = \langle P, c\rangle \xrightarrow{\alpha_2} \langle P_1, d_1\rangle$ s.t. $t_1 \vdash_D t_2'$ and $\langle P_1, d_1\rangle \mathcal{R} \langle P_2, c_2\rangle$ which by definition means that there exists an e_1 s.t. $\alpha_1 \sqcup e_1 = \alpha_2$ and $c_1 \sqcup e_1 = d_1$. Now from $t_2 \vdash_D t_3$ we can deduce that there exists $t_3' = \langle P, c\rangle \xrightarrow{\alpha_2} \langle P_2, d_2\rangle$ s.t. $t_2 \vdash_D t_3'$ and $\langle P_2, d_2\rangle \mathcal{R} \langle P_3, c_3\rangle$ which in this case means that there exists an e_2 s.t. $\alpha_2 \sqcup e_2 = \alpha_3$ and $c_2 \sqcup e_2 = d_2$. To conclude, take $e' = e_1 \sqcup e_2$ and $t' = \langle P, c\rangle \xrightarrow{\alpha_1 \sqcup e'} \langle P_1, c_1 \sqcup e'\rangle$, we can check that $t_1 \vdash_D t'$ by taking e equal to e', we can also verify that since $\langle P_1, d_1\rangle = \langle P_1, c_1 \sqcup e_1\rangle \mathcal{R} \langle P_2, c_2\rangle$ and $\mathcal{R}$ is closed under the addition of constraints (Definition 3.2.5) then $\langle P_1, c_1 \sqcup e_1 \sqcup e_2\rangle = \langle P_1, c_1 \sqcup e'\rangle \mathcal{R} \langle P_3, c_3\rangle$ and hence $t_1 \vdash_{\mathcal{R}} t_3$.*

- *(Antisymmetry) We need to prove that if $t_1 \vdash_{\mathcal{R}} t_2$ and $t_2 \vdash_{\mathcal{R}} t_1$ then $\alpha_1 = \alpha_2$ and $\langle P_1, c_1\rangle\ \mathcal{R} \langle P_2, c_2\rangle$[3]. Since $t_1 \vdash_{\mathcal{R}} t_2$ we know there exists $t_2' =$*

[3] Notice that we do not require that $\langle P_1, c_1\rangle$ is syntactically equivalent to $\langle P_2, c_2\rangle$ but that they are related in $\mathcal{R}$, this is due to the fact that $\mathcal{R}$ is said to contain configurations that are equivalent.

$\langle P, c\rangle \xrightarrow{\alpha_2} \langle P_1, d_1\rangle$ s.t. $t_1 \vdash_D t_2'$ and $\langle P_1, d_1\rangle\mathcal{R}\langle P_2, c_2\rangle$ where there exists an e s.t. $\alpha_1 \sqcup e = \alpha_2$ and $c_1 \sqcup e = d_1$, therefore $\alpha_1 \sqsubseteq \alpha_2$. Following the same reasoning on $t_2 \vdash_{\mathcal{R}} t_1$ we can get that $\alpha_2 \sqsubseteq \alpha_1$ hence $\alpha_1 = \alpha_2$. This also means that $\alpha_1 \sqcup e = \alpha_1$ and so $e \sqsubseteq \alpha_1$, now since $\alpha_1 \sqsubseteq c_1$ then $e \sqsubseteq c_1$ therefore $c_1 = d_1$ and finally $\langle P_1, d_1\rangle = \langle P_1, c_1\rangle\mathcal{R}\langle P_2, c_2\rangle$.

Since id[4] is reflexive and is closed under the addition of constraints, the following proposition follows directly.

Proposition 3.2.2 *$\vdash_D$ is a partial order.*

Proof *Follows from the fact that $\vdash_D$ is $\vdash_{\mathcal{R}}$ when $\mathcal{R} = id$, and since id is reflexive and is closed under the addition of constraints (Definition 3.2.5) then by Proposition 3.2.2 we can conclude that $\vdash_D$ is a partial order.*

Now regarding domination, we can prove a similar property, but in this case we can see that requiring the two labels being different does not allow it to be reflexive, therefore $\succ_{\mathcal{R}}$ is a strict (irreflexive) partial order.

Proposition 3.2.3 *If $\mathcal{R}$ is closed under the addition of constraints then $\succ_{\mathcal{R}}$ is a strict (irreflexive) partial order.*

Proof *In this case we will show that $\succ_{\mathcal{R}}$ is irreflexive and transitive. Let $t = \langle P, c\rangle \xrightarrow{\alpha_1} \langle P', c'\rangle$: We proceed to prove each characteristic separately.*

- *(Irreflexivity) Let us assume by means of contradiction that $t \succ_{\mathcal{R}} t$ then this would mean that there exists $t' = \langle P, c\rangle \xrightarrow{\alpha_1} \langle Q, d\rangle$ s.t. $t \succ_D t'$ where $\langle P', c'\rangle\mathcal{R}\langle Q, d\rangle$. But for any such t' the label is α_1 and therefore $t \not\succ_D t'$ since its labels are not different, a contradiction. Hence, $t \not\succ_{\mathcal{R}} t$.*
- *(Transitivity) The proof follows the line of Proposition 3.2.1, where in this case $\alpha_1 \neq \alpha_2 \neq \alpha_3$, notice that this fact does not affect the argument above.*

Again, since id is closed under the addition of constraints, then the following proposition is a consequence of the one above.

[4] The identity relation.

Proposition 3.2.4 *$\succ_D$ is a strict partial order.*

Proof *Follows from the fact that $\succ_D$ is $\succ_{\mathcal{R}}$ when $\mathcal{R} = id$, and since id is closed under the addition of constraints (Definition 3.2.5) then by Proposition 3.2.3 we can conclude that $\vdash_D$ is a partial order.*

As we pointed out for the entailment relation, another important property is well-foundedness. It is obtained via Proposition 2.1.1 in Section 2.1.1 ($\sqsubseteq$ is well-founded).

Proposition 3.2.5 *$\vdash_D$ and $\succ_D$ are well-founded.*

Proposition 3.2.6 *$\vdash_{\mathcal{R}}$ and $\succ_{\mathcal{R}}$ are well-founded.*

3.2.6 Symbolic and Irredundant Bisimilarity

We are now able to introduce an equivalence called *irredundant bisimilarity* which allows to compute bisimilarity and redundancy *at the same time*. This notion can be used to efficiently compute $\dot{\sim}_{sb}$. As its name suggests, the idea is to consider only the transitions that are irredundant (up to $\dot{\sim}_{sb}$ itself). It is proven that both notions coincide, and it is achieved via an intermediate notion called *symbolic bisimilarity* that we will introduce below.

Intuitively, two configurations γ_1 and γ_2 are symbolic bisimilar iff (i) they have the same barbs and (ii) whenever there is a transition from γ_1 to γ_1' using α, then we require that γ_2 must reply with a similar transition $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ (where γ_1' and γ_2' are now equivalent) or some other transition that derives it. In other words, the move from the defender does not need to use exactly the same label, but a transition that is "stronger" (in terms of derivation $\vdash_D$) could also do the job. Formally we have the definition below.

Definition 3.2.6 (Symbolic Bisimilarity) *A symbolic bisimulation is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$ with $\gamma_1 = \langle P, c\rangle$ and $\gamma_2 = \langle Q, d\rangle$:*

(i) if $\gamma_1 \downarrow_e$ then $\gamma_2 \downarrow_e$,

(ii) if $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ *then there exists a transition* $t = \langle Q, d\rangle \xrightarrow{\beta} \langle Q', d''\rangle$ *s.t.* $t \vdash_D \langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ *and* $\langle P', c'\rangle \mathcal{R} \langle Q', d'\rangle$

We say that γ_1 *and* γ_2 *are symbolic bisimilar (*$\gamma_1 \dot{\sim}_{sym} \gamma_2$*) if there exists a symbolic bisimulation* $\mathcal{R}$ *s.t.* $(\gamma_1, \gamma_2) \in \mathcal{R}$*.*

Example 3.2.4 *To illustrate the notion of symbolic bisimilarity (*$\dot{\sim}_{sym}$*), we take* $\langle P+Q, true\rangle$ *and* $\langle P, true\rangle$ *from Example 3.2.1. We provide a symbolic bisimulation* $\mathcal{R} = \{(\langle P+Q, true\rangle, \langle P, true\rangle)\} \cup id$ *(relation* $\mathcal{R}$ *is shown in Figure 3.3) to prove* $\langle P+Q, true\rangle \dot{\sim}_{sym} \langle P, true\rangle$*. We take the pair* $(\langle P+Q, true\rangle, \langle P, true\rangle)$*. The first condition in Definition 3.2.6, concerning the barbs is trivial. For the second one we take* $\langle P+Q, true\rangle \xrightarrow{x<5} \langle T, x<5\rangle$*, and one can find transitions* $t = \langle P, true\rangle \xrightarrow{x<7} \langle T, x<7\rangle$ *and* $t' = \langle P, true\rangle \xrightarrow{x<5} \langle T, x<5\rangle$ *s.t.* $t \vdash_D t'$ *and* $\langle T, x<5\rangle \mathcal{R} \langle T, x<5\rangle$*. The remaining transitions are trivially verified.*

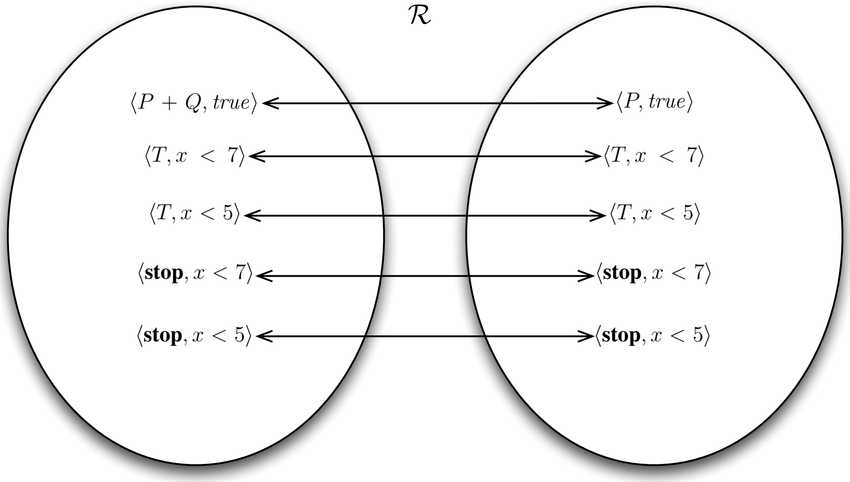

$\mathcal{R}$
$\langle P+Q, true\rangle$
$\langle P, true\rangle$
$\langle T, x < 7\rangle$
$\langle T, x < 7\rangle$
$\langle T, x < 5\rangle$
$\langle T, x < 5\rangle$
$\langle \mathbf{stop}, x < 7\rangle$
$\langle \mathbf{stop}, x < 7\rangle$
$\langle \mathbf{stop}, x < 5\rangle$
$\langle \mathbf{stop}, x < 5\rangle$

Figure 3.3: Symbolic Bisimulation for Example 3.2.4

The following lemmata will lead us to conclude that symbolic bisimilarity (Definition 3.2.6) is equivalent to saturated barbed bisimilarity (Definition 2.2.2 in Section 2.2.1).

Lemma 3.2.3 *If* $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$ *then* $\langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle$*.*

Proof *We will prove that* $\mathcal{R} = \{(\langle P, c \sqcup a\rangle, \langle Q, d \sqcup a\rangle) \mid \langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle\}$ *is a saturated barbed bisimulation.*

(i) Since $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$ *then from condition (i) and by monotonicity we have that for all* e *s.t.* $\langle P, c \sqcup a\rangle \downarrow_e$ *then* $\langle Q, d \sqcup a\rangle \downarrow_e$*.*

(ii) We need to prove that if $\langle P, c \sqcup a\rangle \longrightarrow \langle P_1, c_1\rangle$ *then there exists* $\langle Q_1, d_1\rangle$ *s.t.* $\langle Q, d \sqcup a\rangle \longrightarrow \langle Q_1, d_1\rangle$ *and* $\langle P_1, c_1\rangle \mathcal{R} \langle Q_1, d_1\rangle$*. Now let us assume that* $\langle P, c \sqcup a\rangle \longrightarrow \langle P', c'\rangle$ *(take* $P_1 = P'$ *and* $c_1 = c'$*) then by completeness (Lemma 3.2.2) we know that there exist* α *and* b *s.t.* $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c''\rangle$ *where* $\alpha \sqcup b = a$ *and* $c'' \sqcup b = c'$*, thus if such transition exists then by* $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$ *we get that there exists a transition* $t = \langle Q, d\rangle \xrightarrow{\beta} \langle Q', d'\rangle$ *s.t.* $t \vdash_D \langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d''\rangle$ *and* $\langle P', c''\rangle \dot{\sim}_{sym} \langle Q', d''\rangle$*. Expanding the definition of* $\vdash_D$ *there is a* b' *s.t.* $\beta \sqcup b' = \alpha$ *and* $d' \sqcup b' = d''$*. We are now able to apply soundness (Lemma 3.2.1) on* t *hence* $\langle Q, d \sqcup \beta\rangle \longrightarrow \langle Q', d'\rangle$*, and by monotonicity we are able to add* $b \sqcup b'$ *to the store to obtain* $\langle Q, d \sqcup \beta \sqcup b \sqcup b'\rangle \longrightarrow \langle Q', d' \sqcup b \sqcup b'\rangle$ *which is equivalent to say (by using the equations above)* $\langle Q, d \sqcup a\rangle \longrightarrow \langle Q', d'' \sqcup b\rangle$*. To conclude, take* $Q_1 = Q'$ *and* $d_1 = d'' \sqcup b$*, therefore* $\langle Q, d \sqcup a\rangle \longrightarrow \langle Q_1, d_1\rangle$ *and since* $\langle P', c''\rangle \dot{\sim}_{sym} \langle Q', d''\rangle$ *then* $\langle P_1, c_1\rangle = \langle P', c'\rangle = \langle P', c'' \sqcup b\rangle \mathcal{R} \langle Q', d'' \sqcup b\rangle = \langle Q_1, d_1\rangle$*.*

(iii) By definition of $\mathcal{R}$*, it is already closed under the addition of constraints.*

Lemma 3.2.4 *If* $\langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle$ *then* $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$*,*

Proof *We will prove that* $\mathcal{R} = \{(\langle P, c\rangle, \langle Q, d\rangle) \mid \langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle\}$ *is a symbolic bisimulation.*

(i) Since $\langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle$ *then from condition (i) we have that if* $\langle P, c\rangle \downarrow_e$ *then* $\langle Q, d\rangle \downarrow_e$ *.*

(ii) Let us start by assuming that $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$*: Then we need to prove that there exists a transition* $t = \langle Q, d\rangle \xrightarrow{\beta} \langle Q', d''\rangle$ *s.t.* $t \vdash_D \langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ *and* $\langle P', c'\rangle \mathcal{R} \langle Q', d'\rangle$*. By soundness we know* $\langle P, c \sqcup \alpha\rangle \longrightarrow \langle P', c'\rangle$*, now since* $\langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle$ *by condition (ii) we obtain* $\langle Q, d \sqcup \alpha\rangle \longrightarrow \langle Q', d'\rangle$

where $\langle P', c'\rangle \dot{\sim}_{sb} \langle Q', d'\rangle$. From this transition and completeness we can deduce that there exist β and b s.t. $\langle Q, d\rangle \xrightarrow{\beta} \langle Q', d''\rangle$ (let us call this transition t) where $\beta \sqcup b = \alpha$ and $d'' \sqcup b = d'$. Thus by definition of $\vdash_D$ we can conclude that $t \vdash_D \langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ and since $\langle P', c'\rangle \dot{\sim}_{sb} \langle Q', d'\rangle$ then $\langle P', c'\rangle \mathcal{R} \langle Q', d'\rangle$

This notion, that does not quantify over all possible contexts, coincides with $\dot{\sim}_{sb}$.

Theorem 3.2.1 *$\langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle$ iff $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$.*

Proof *Using Lemma 3.2.3 and Lemma 3.2.4*

Since $\dot{\sim}_{sb}$ is closed under the addition of constraints, then $\dot{\sim}_{sym}$ should also have this property.

Lemma 3.2.5 *If $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$ then $\forall a \in Con_0$, $\langle P, c \sqcup a\rangle \dot{\sim}_{sym} \langle Q, d \sqcup a\rangle$.*

Proof *It follows directly from Theorem 3.2.1.*

We now present the *irredundant bisimilarity*. As we mentioned before, the idea is to focus on the transitions that are irredundant. It then follows the usual bisimulation game where the defender is required to reply with the same label and ends up in an equivalent configuration to the attacker's move. Such irredundancy can be computed along with the equivalence classes of $\dot{\sim}_{sb}$.

Definition 3.2.7 (Irredundant Bisimilarity) *An irredundant bisimulation is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$:*

(i) if $\gamma_1 \downarrow_e$ then $\gamma_2 \downarrow_e$,

(ii) if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ and it is irredundant in $\mathcal{R}$ then there exists γ_2' s.t. $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$.

We say that γ_1 and γ_2 are irredundant bisimilar ($\gamma_1 \dot{\sim}_I \gamma_2$) if there exists an irredundant bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Example 3.2.5 We can verify that the relation $\mathcal{R}$ in Example 3.2.4 is an irredundant bisimulation to show that $\langle P+Q, \mathit{true}\rangle \dot{\sim}_I \langle P, \mathit{true}\rangle$. We take the pair $(\langle P+Q, \mathit{true}\rangle, \langle P, \mathit{true}\rangle)$. The first item in Definition 3.2.7 is obvious. Then take $\langle P+Q, \mathit{true}\rangle \xrightarrow{x<7} \langle T, x<7\rangle$, which is irredundant according to Definition 3.2.4, then there exists a $\langle T, x<7\rangle$ s.t. $\langle P, \mathit{true}\rangle \xrightarrow{x<7} \langle T, x<7\rangle$ and $(\langle T, x<7\rangle, \langle T, x<7\rangle) \in \mathcal{R}$. The other pairs are trivially proven. Notice that $\langle P+Q, \mathit{true}\rangle \xrightarrow{x<7} \langle T, x<7\rangle \succ_{\mathcal{R}} \langle P+Q, \mathit{true}\rangle \xrightarrow{x<5} \langle T, x<5\rangle$ hence $\langle P+Q, \mathit{true}\rangle \xrightarrow{x<5} \langle T, x<5\rangle$ is redundant, thus it does not need to be matched by $\langle P, \mathit{true}\rangle$.

This previous notion of irredundant bisimilarity turns out to be closed under the addition of constraints, something we prove with the following lemma.

Lemma 3.2.6 *If $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ then for all $a \in Con_0$, $\langle P, c\sqcup a\rangle \dot{\sim}_I \langle Q, d\sqcup a\rangle$.*

Proof *We need to prove that $\mathcal{R} = \{(\langle P, c\sqcup e\rangle, \langle Q, d\sqcup e\rangle) \mid \langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle\}$ is an irredundant bisimulation.*

(i) Since $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ then from condition (i) and by monotonicity we have that for all e s.t. $\langle P, c\sqcup a\rangle \downarrow_e$ then $\langle Q, d\sqcup a\rangle \downarrow_e$.

(ii) Let us start by assuming that

$$\langle P, c\sqcup a\rangle \xrightarrow{\alpha} \langle P', c'\rangle \textit{ which is irredundant in } \mathcal{R} \tag{3.1}$$

then we need to prove that there exists $\langle Q', d'\rangle$ s.t. $\langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ and $\langle P', c'\rangle \mathcal{R} \langle Q', d'\rangle$. Now by soundness on the transition from $\langle P, c\sqcup a\rangle$ we know that $\langle P, c\sqcup a\sqcup\alpha\rangle \longrightarrow \langle P', c'\rangle$ and from completeness we get that there exist b and β s.t.

$$t_1 = \langle P, c\rangle \xrightarrow{\beta} \langle P', c''\rangle \textit{ where } \beta\sqcup b = a\sqcup\alpha \textit{ and } c''\sqcup b = c' \tag{3.2}$$

Since $\succ_{\mathcal{R}}$ is well founded then there exist an irredundant (not dominated by the rest) transition $t_2 = \langle P, c\rangle \xrightarrow{\lambda} \langle P_1, c_1\rangle$ that dominates t_1 in $\mathcal{R}$ ($t_2 \succ_{\mathcal{R}} t_1$), therefore

$$t_2 \succ_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c_2\rangle \textit{ and } \langle P_1, c_2\rangle \mathcal{R} \langle P', c''\rangle \tag{3.3}$$

by definition of $\succ_D$ *there exists* b' *s.t.*

$$\lambda \sqcup b' = \beta, c_1 \sqcup b' = c_2 \textit{ and } \lambda \neq \beta \tag{3.4}$$

Now since $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ *and the irredundant* t_2*, then there exists* $\langle Q_1, d_1\rangle$ *s.t.*

$$\langle Q, d\rangle \xrightarrow{\lambda} \langle Q_1, d_1\rangle \textit{ and } \langle P_1, c_1\rangle \dot{\sim}_I \langle Q_1, d_1\rangle \tag{3.5}$$

using soundness $\langle Q, d \sqcup \lambda\rangle \longrightarrow \langle Q_1, d_1\rangle$ *and monotonicity we can obtain* $\langle Q, d \sqcup \underbrace{\overbrace{\lambda \sqcup b'}^{\beta} \sqcup b}_{a \sqcup \alpha}\rangle \longrightarrow \langle Q_1, d_1 \sqcup b' \sqcup b\rangle$ *and from the latter condition in* (3.5) *we can also deduce that* $\langle Q_1, d_1 \sqcup b' \sqcup b\rangle\ \mathcal{R}\langle P_1, \underbrace{c_1 \sqcup b'}_{c_2} \sqcup b\rangle$ *therefore by the second condition in* (3.3) *we know that* $\langle P_1, c_2 \sqcup b\rangle \mathcal{R} \langle P', \underbrace{c'' \sqcup b}_{c'}\rangle$*. Let* $d_1' = d_1 \sqcup b' \sqcup b$*, we can summarize this part by saying that*

$$\langle Q, d \sqcup a \sqcup \alpha\rangle \longrightarrow \langle Q_1, d_1'\rangle \textit{ and } \langle P', c'\rangle \mathcal{R} \langle Q_1, d_1'\rangle \tag{3.6}$$

Now let us reason on this transition, by completeness there exist α_1 *and* b_1 *s.t.*

$$t_3 = \langle Q, d \sqcup a\rangle \xrightarrow{\alpha_1} \langle Q_1, d_2\rangle \textit{ where } \alpha_1 \sqcup b_1 = \alpha \textit{ and } d_2 \sqcup b_1 = d_1' \tag{3.7}$$

By means of contradiction let us assume that $\alpha_1 \neq \alpha$*, then by soundness (on* t_3*)* $\langle Q, d \sqcup a \sqcup \alpha_1\rangle \longrightarrow \langle Q_1, d_2\rangle$ *and, by completeness on this transition, we know there exist* α_2 *and* b_2 *s.t.*

$$t_4 = \langle Q, d\rangle \xrightarrow{\alpha_2} \langle Q_1, d_2'\rangle \textit{ where } \alpha_2 \sqcup b_2 = a \sqcup \alpha_1 \textit{ and } d_2' \sqcup b_2 = d_2 \tag{3.8}$$

By the well-foundedness of $\succ_{\mathcal{R}}$*, we know there exists an irredundant transition* $t_5 = \langle Q, d\rangle \xrightarrow{\alpha'} \langle Q_3, d_3\rangle$ *s.t.* $t_5 \succ_{\mathcal{R}} t_4$*, namely,*

$$t_5 \succ_D \langle Q, d\rangle \xrightarrow{\alpha_2} \langle Q_3, d_3'\rangle \textit{ and } \langle Q_3, d_3'\rangle \mathcal{R} \langle Q_1, d_2'\rangle \tag{3.9}$$

hence, there exists b_3 *s.t.* $\alpha' \sqcup b_3 = \alpha_2$*,* $d_3 \sqcup b_3 = d_3'$ *and* $\alpha' \neq \alpha_2$*. Now since*

$\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ then from the irredundant t_5 we can deduce that

$$\langle P, c\rangle \xrightarrow{\alpha'} \langle P_3, c_3\rangle \text{ and } \langle P_3, c_3\rangle \mathcal{R} \langle Q_3, d_3\rangle \tag{3.10}$$

by soundness we get $\langle P, c \sqcup \alpha'\rangle \longrightarrow \langle P_3, c_3\rangle$ and by monotonicity

$$\langle P, c \sqcup \overbrace{\alpha' \sqcup b_2 \sqcup b_3}^{a \sqcup \alpha_1}\rangle \longrightarrow \langle P_3, \overbrace{c_3 \sqcup b_2 \sqcup b_3}^{c_4}\rangle, \tag{3.11}$$

from the latter condition in (3.10) and by definition of $\mathcal{R}$ we have $\langle P_3, c_3 \sqcup b_2 \sqcup b_3\rangle \mathcal{R} \langle Q_3, d_3 \sqcup b_2 \sqcup b_3\rangle$, and since $d_3 \sqcup b_3 = d'_3$ and from the latter condition in (3.9) then $\langle Q_3, d'_3 \sqcup b_2\rangle \mathcal{R} \langle Q_1, d'_2 \sqcup b_2\rangle$ and using $d'_2 \sqcup b_2 = d_2$ we can conclude that $\langle P_3, c_4\rangle \mathcal{R} \langle Q_1, d_2\rangle$. Going back to the transition in (3.11), we can rewrite it as $\langle P, c \sqcup a \sqcup \alpha_1\rangle \longrightarrow \langle P_3, c_4\rangle$, then by completeness we know there exist b_4 and α'_1 s.t.

$$t_6 = \langle P, c \sqcup a\rangle \xrightarrow{\alpha'_1} \langle P_3, c'_4\rangle \text{ where } \alpha'_1 \sqcup b_4 = \alpha_1 \text{ and } c'_4 \sqcup b_4 = c_4 \tag{3.12}$$

Notice that if such transition exists then $t_6 \succ_{\mathcal{R}}$ (3.1), as we prove as follows

$$t_6 \succ_D \langle P, c \sqcup a\rangle \xrightarrow{\alpha'_1 \sqcup b_4 \sqcup b_1} \langle P_3, \overbrace{c'_4 \sqcup b_4}^{c_4} \sqcup b_1\rangle \tag{3.13}$$

and now it is left to prove that $\langle P_3, c_4 \sqcup b_1\rangle \mathcal{R} \langle P', c'\rangle$, given that $\langle P_3, c_4\rangle \mathcal{R}$ $\langle Q_1, d_2\rangle$ then by definition of $\mathcal{R}$ we have $\langle P_3, c_4 \sqcup b_1\rangle \mathcal{R} \langle Q_1, d_2 \sqcup b_1\rangle$. Since $\langle Q_1, d_2 \sqcup b_1\rangle = \langle Q_1, d'_1\rangle$ and we have already proven that $\langle P', c'\rangle \mathcal{R} \langle Q_1, d'_1\rangle$ (latter condition in (3.6)), finally we can conclude that $\langle P_3, c_4 \sqcup b_1\rangle \mathcal{R} \langle P', c'\rangle$ and therefore (3.1) is redundant, a contradiction. To conclude, α_1 must be equal to α, otherwise we would get an absurd as shown previously, thus we can conclude our main result by using (3.6), (3.7) and assuming $Q' = Q_1$, $d' = d'_1$,

$$\langle Q, d \sqcup a\rangle \xrightarrow{\alpha} \langle Q_1, d'_1\rangle \text{ and } \langle P', c'\rangle \mathcal{R} \langle Q_1, d'_1\rangle \tag{3.14}$$

Indeed $\langle Q, d \sqcup a\rangle$ can defend from the attacker's move (the token game) by using the same label and still remain in the relation.

As we had already done with Lemma 3.2.3 and Lemma 3.2.4 to prove that $\langle P, c\rangle\dot{\sim}_{sb}\langle Q, d\rangle$ iff $\langle P, c\rangle\dot{\sim}_{sym}\langle Q, d\rangle$, the following lemmata will help us find the correspondence between symbolic bisimilarity (Definition 3.2.6) and irredundant bisimilarity (Definition 3.2.7).

Lemma 3.2.7 *If $\langle P, c\rangle\dot{\sim}_{sym}\langle Q, d\rangle$ then $\langle P, c\rangle\dot{\sim}_{I}\langle Q, d\rangle$*

Proof *We will prove that $\mathcal{R} = \{(\langle P, c\rangle, \langle Q, d\rangle) \mid \langle P, c\rangle\dot{\sim}_{sym}\langle Q, d\rangle\}$ is an irredundant bisimulation.*

(i) Since $\langle P, c\rangle\dot{\sim}_{sym}\langle Q, d\rangle$ it is direct result from condition (i).

(ii) Assume that $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ (1) which is irredundant in $\mathcal{R}$ then we need to prove that there exists $\langle Q', d'\rangle$ s.t. $\langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ and $\langle P', c'\rangle\mathcal{R}\langle Q', d'\rangle$. Since $\langle P, c\rangle\dot{\sim}_{sym}\langle Q, d\rangle$ and from (1), we have that there exists $t = \langle Q, d\rangle \xrightarrow{\beta} \langle Q', d''\rangle$ s.t. $t \vdash_D \langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ and $\langle P', c'\rangle\dot{\sim}_{sym}\langle Q', d'\rangle$, thus by definition of $\vdash_D$, there is b s.t. $\beta \sqcup b = \alpha$ and $d'' \sqcup b = d'$. Now let us assume by means of contradiction that $\beta \neq \alpha$, then we can use t to reason about what $\langle P, c\rangle$ can do, again from $\langle P, c\rangle\dot{\sim}_{sym}\langle Q, d\rangle$ and t we can say that there exists a transition $t' = \langle P, c\rangle \xrightarrow{\lambda} \langle P_1, c_1'\rangle$ s.t. $t' \vdash_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c_1\rangle$ and $\langle P_1, c_1\rangle\dot{\sim}_{sym}\langle Q', d''\rangle$ (2). By definition of $\vdash_D$, the last derivation means that there is a b' s.t. $\lambda \sqcup b' = \beta$ and $c_1' \sqcup b' = c_1$. Now we can use Lemma 3.2.5 on (2) to get $\langle P_1, c_1 \sqcup b\rangle\mathcal{R}\langle Q', d'' \sqcup b\rangle$ therefore, given that $d'' \sqcup b = d'$, then $\langle P_1, c_1 \sqcup b\rangle\mathcal{R}\langle Q', d'\rangle$, which by definition of $\mathcal{R}$ means $\langle P_1, c_1 \sqcup b\rangle\dot{\sim}_{sym}\langle Q', d'\rangle$ (3). We can also conclude that since $\langle P', c'\rangle\dot{\sim}_{sym}\langle Q', d'\rangle$ then by transitivity $\langle P_1, c_1 \sqcup b\rangle\dot{\sim}_{sym}\langle P', c'\rangle$ and hence $\langle P_1, c_1 \sqcup b\rangle\mathcal{R}\langle P', c'\rangle$ (4). On the other hand, notice that now t' is able to dominate our originally irredundant transition, namely $t' \succ_{\mathcal{R}}$ (1), as follows

$$t' \succ_D \langle P, c\rangle \xrightarrow{\overbrace{\overbrace{\lambda \sqcup b'}^{\beta} \sqcup b}^{\alpha}} \langle P', \overbrace{c_1' \sqcup b'}^{c_1} \sqcup b\rangle \text{ and } \langle P_1, c_1 \sqcup b\rangle\mathcal{R}\langle P', c'\rangle \text{ from (4)}$$

therefore if $\alpha \neq \beta$ then we would get an absurd since (1) would be redundant in $\mathcal{R}$, thus we can finally say that $\alpha = \beta$ which allow us to conclude that $\langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d'\rangle$ and $\langle P', c'\rangle\dot{\sim}_{sym}\langle Q', d'\rangle$ then $\langle P', c'\rangle\mathcal{R}\langle Q', d'\rangle$.

Lemma 3.2.8 *If $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ then $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$*

Proof *We will prove that $\mathcal{R} = \{(\langle P, c\rangle, \langle Q, d\rangle) \mid \langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle\}$ is a symbolic bisimulation.*

(i) Since $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ it is direct result from condition (i)

(ii) Take $t = \langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ then since $\succ_{\mathcal{R}}$ is well founded then there exists an irredundant transition $t' = \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c_1\rangle$ s.t. $t' \succ_{\mathcal{R}} t$,

$$t' \succ_D \langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c_1'\rangle \text{ where } \langle P_1, c_1'\rangle \mathcal{R} \langle P', c'\rangle \tag{3.15}$$

By definition of $\succ_D$, there exists a b s.t. $\beta \sqcup b = \alpha$, $c_1 \sqcup b = c_1'$ and $\alpha \neq \beta$. Now since $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ and t' then we know there is a transition $t'' = \langle Q, d\rangle \xrightarrow{\beta} \langle Q', d'\rangle$ and $\langle P_1, c_1\rangle \dot{\sim}_{sym} \langle Q', d'\rangle$. Thus, from Lemma 3.2.6 $\langle P_1, c_1 \sqcup b\rangle \mathcal{R} \langle Q', d' \sqcup b\rangle$, equivalently $\langle P_1, c_1'\rangle \mathcal{R} \langle Q', d' \sqcup b\rangle$ and using the latter condition in (3.15) *then $\langle Q', d' \sqcup b\rangle \mathcal{R} \langle P', c'\rangle$. We can finally conclude that $t'' \succ_D \langle Q, d\rangle \xrightarrow{\alpha} \langle Q', d' \sqcup b\rangle$ and $\langle Q', d' \sqcup b\rangle \mathcal{R} \langle P', c'\rangle$, therefore the condition for being a symbolic bisimulation is proven.*

And most importantly, it coincides with the symbolic one.

Theorem 3.2.2 *$\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$ iff $\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$*

Proof *Using Lemma 3.2.7 and Lemma 3.2.8.*

Finally, we can clearly see that the above-defined equivalences (Definition 3.2.6 and Definition 3.2.7) coincide with $\dot{\sim}_{sb}$.

Theorem 3.2.3 *$\langle P, c\rangle \dot{\sim}_I \langle Q, d\rangle$ iff $\langle P, c\rangle \dot{\sim}_{sym} \langle Q, d\rangle$ iff $\langle P, c\rangle \dot{\sim}_{sb} \langle Q, d\rangle$*

Proof *Using transitivity, Theorem 3.2.1 and Theorem 3.2.2.*

3.3 Partition Refinement for CCP

Recall that we mentioned in Section 2.2 that checking $\dot{\sim}_{sb}$ seems hard because of the quantification over all possible constraints. However, by using the relations showed in Theorem 3.2.3 we shall introduce an algorithm for checking $\dot{\sim}_{sb}$ by employing the notion of irredundant bisimulation. In Figure 3.4 we show the design of this algorithm.

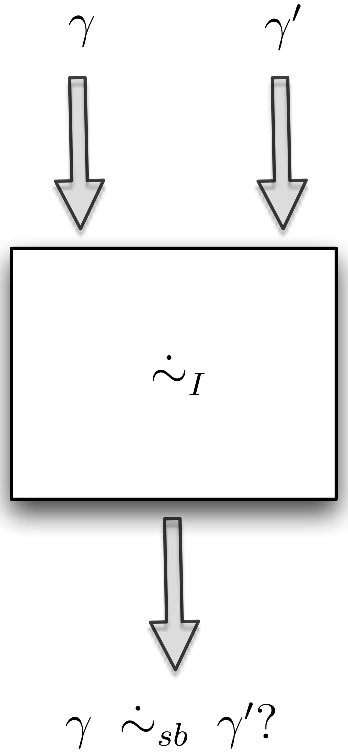


Figure 3.4: CCP Partition Refinement

The first novelty w.r.t. the standard partition refinement (Algorithm 3.1.1) consists in using *barbs*. Since configurations satisfying different barbs are surely different, we can safely start with a partition that equates all and only those states satisfying the same barbs. Note that two configurations satisfy the same barbs iff they have the same store. Thus, we take as initial partition $\mathcal{P}^0 = \{IS^{\star}_{d_1}\} \dots \{IS^{\star}_{d_n}\}$, where $IS^{\star}_{d_i}$ is the subset of the configurations of $IS^{\star}$ with store d_i.

Another difference is that instead of using the function $\mathbf{F}$ of Algorithm 3.1.1, we refine the partitions by employing the function $\mathbf{IR}$ defined as follows:

Definition 3.3.1 (IR) *For all partitions $\mathcal{P}$, $\gamma_1 \, \mathbf{IR}(\mathcal{P}) \, \gamma_2$ if*

- *if* $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ *is irredundant in* $\mathcal{P}$*, then there exists* γ_2' *s.t.* $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ *and* $\gamma_1' \mathcal{P} \gamma_2'$.

It is now important to observe that in the computation of $\mathbf{IR}(\mathcal{P}^n)$, there might be involved also states that are not reachable from the initial states IS. For instance, consider the LTSs of $\langle S, true\rangle$ and $\langle R + S, true\rangle$ in Figure 3.1. The state $\langle P, z < 5\rangle$ is not reachable but is needed to check if $\langle R + S, true\rangle \xrightarrow{z<5} \langle P + Q, z < 5\rangle$ is redundant.

For this reason, we have also to change the initialization step of our algorithm, by including in the set $IS^\star$ all the states that are needed to check redundancy. This is done by using the following closure rules.

$$(\textsc{is})\frac{\gamma \in IS}{\gamma \in IS^\star} \qquad\qquad (\textsc{rs})\frac{\gamma_1 \in IS^\star \quad \gamma_1 \xrightarrow{\alpha} \gamma_2}{\gamma_2 \in IS^\star}$$

$$(\textsc{rd})\frac{\gamma \in IS^\star \quad \gamma \xrightarrow{\alpha_1} \gamma_1 \quad \gamma \xrightarrow{\alpha_2} \gamma_2 \quad \gamma \xrightarrow{\alpha_1} \gamma_1 \succ_D \gamma \xrightarrow{\alpha_2} \gamma_3}{\gamma_3 \in IS^\star}$$

The rule (RD) adds all the states that are needed to check redundancy. Indeed, if γ can perform both $\xrightarrow{\alpha_1} \gamma_1$ and $\xrightarrow{\alpha_2} \gamma_2$ s.t. $\gamma \xrightarrow{\alpha_1} \gamma_1 \succ_D \gamma \xrightarrow{\alpha_2} \gamma_3$, then $\gamma \xrightarrow{\alpha_2} \gamma_2$ would be redundant whenever $\gamma_2 \dot{\sim}_{sb} \gamma_3$.

In Example 3.3.1 we can understand how 'possibly' redundant transitions are found, in order to check redundancy. Likewise, we show how rule (RD) adds new states to the set of reachable states $IS^\star$.

Example 3.3.1 *Let us assume that we have an automaton generated by a ccp program where* $\langle P_0, true\rangle, \langle P_1, x < 10\rangle, \langle P_2, x < 5\rangle \in IS^\star$ *and with transitions* $\langle P_0, true\rangle \xrightarrow{x<10} \langle P_1, x < 10\rangle$ *and* $\langle P_0, true\rangle \xrightarrow{x<5} \langle P_2, x < 5\rangle$*. Therefore,* $\langle P_0, true\rangle \xrightarrow{x<10} \langle P_2, x < 5\rangle$ *is 'possibly' redundant,* $\langle P_0, true\rangle \xrightarrow{x<10} \langle P_1, x < 10\rangle \succ_D \langle P_0, true\rangle \xrightarrow{x<5} \langle P_1, x < 5\rangle$ *and we get* $\langle P_1, x < 5\rangle \in IR^\star$.

Figure 3.2 shows the partitions computed by the algorithm with initial states $\langle R' + S, true\rangle$, $\langle S, true\rangle$ and $\langle R + S, true\rangle$. Note that, as expected, in the final

```
Algorithm 3.3.1 CCP-Partition-Refinement($IS$)
Initialization
    1. Compute $IS^\star$ with the rules (IS), (RS) and (RD),
    2. $\mathcal{P}^0 := \{IS^\star_{d_1}\} \dots \{IS^\star_{d_n}\}$,
Iteration $\mathcal{P}^{n+1} := \mathbf{IR}(\mathcal{P}^n)$
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.
```

partition $\langle R{+}S, true\rangle$ and $\langle S, true\rangle$ belong to the same block, while $\langle R'{+}S, true\rangle$ belong to a different one (meaning that the former two are saturated bisimilar, while $\langle R' + S, true\rangle$ is not). In the initial partition all states with the same store are equated. In $\mathcal{P}^1$, the blocks are split by considering the outgoing transitions: all the final states are distinguished (since they cannot perform any transitions) and $\langle T', z < 5 \sqcup x < 5\rangle$ is distinguished from $\langle T, z < 5 \sqcup x < 5\rangle$. All the other blocks are not divided, since all the transitions with label $x < 5$ are redundant in $\mathcal{P}^0$ (since $\langle P, z < 5\rangle \mathcal{P}^0 \langle P + Q', z < 5\rangle$, $\langle P, z < 5\rangle \mathcal{P}^0 \langle P + Q, z < 5\rangle$ and $\langle T', z < 5 \sqcup x < 5\rangle \mathcal{P}^0 \langle T, z < 5 \sqcup x < 5\rangle$). Then, in $\mathcal{P}^2$, $\langle P + Q', z < 5\rangle$ is distinguished from $\langle P, z < 5\rangle$ since the transition $\langle P + Q', z < 5\rangle \xrightarrow{x<5}$ is not redundant anymore in $\mathcal{P}^1$ (since $\langle T', z < 5 \sqcup x < 5\rangle$ and $\langle T, z < 5 \sqcup x < 5\rangle$ belong to different blocks in $\mathcal{P}^1$). Then in $\mathcal{P}^3$, $\langle R' + S, true\rangle$ is distinguished from $\langle S, true\rangle$ since the transition $\langle R' + S, true\rangle \xrightarrow{x<5}$ is not redundant in $\mathcal{P}^2$ (since $\langle P + Q', z < 5\rangle \not\mathcal{P}^2 \langle P, z < 5\rangle$). Finally, the algorithm computes $\mathcal{P}^4$ that is equal to $\mathcal{P}^3$ and return it. It is interesting to observe that the transition $\langle R + S, true\rangle \xrightarrow{x<5}$ is redundant in all the partitions computed by the algorithm (and thus in $\dot{\sim}_{sb}$), while the transition $\langle R' + S, true\rangle \xrightarrow{x<5}$ is considered redundant in $\mathcal{P}^0$ and $\mathcal{P}^1$ and not redundant in $\mathcal{P}^2$ and $\mathcal{P}^3$.

3.3.1 Termination

Note that any iteration splits blocks and never fuse them. For this reason if $IS^\star$ is finite, the algorithm terminates in at most $|IS^\star|$ iterations. The proof of the next proposition assumes that $\vdash_D$ is decidable. However, as we shall prove in the next section, the decidability of $\vdash_D$ follows from our assumption about the decidability of the ordering relation $\sqsubseteq$ of the underlying constraint system and Theorem 3.3.2

in the next section.

Proposition 3.3.1 *If $IS^\star$ is finite, then the algorithm terminates and the resulting partition coincides with $\dot{\sim}_{sb}$.*

Proof Using Corollary 1 of [BM09] and the decidability of $\succ_D$.

We now prove that if the set $\texttt{Config}(IS)$ of all configurations reachable from IS (through the LTS generated by the rules in Table 2.2) is finite, then $IS^\star$ is finite. This necessary and sufficient condition is standard in all the partition refinement approaches, like e.g. those in [KS83, PS96, MPLS] for CCS and π-calculus.

Although we lose expressiveness, this condition can be easily guaranteed by imposing some syntactic restrictions on ccp terms, like for instance, by excluding either the procedure call or the hiding operator.

Theorem 3.3.1 *If $\texttt{Config}(IS)$ is finite, then $IS^\star$ is finite.*

Proof As a first step, we observe that a configuration $\gamma \in IS^\star$ only if $\gamma = \langle P, d \sqcup e\rangle$ and $\langle P, d\rangle \in \texttt{Config}(IS)$. Then we prove that there are only finitely many such constraints e.

Let $\texttt{Label}(IS)$ be the set of all labels in the LTS of IS generated by rules in Table 2.2. This set is finite (since $\texttt{Config}(IS)$ is finite), and its downward closure $\downarrow\texttt{Label}(IS) = \{a \mid \exists b \in \texttt{Label}(IS) \text{ with } a \sqsubseteq b\}$ is also finite (since $\sqsubseteq$ is well-founded). The set of all e s.t. $\langle P, d \sqcup e\rangle \in IS^*$ (with $\langle P, d\rangle \in \texttt{Config}(IS)$) is a subset of $\downarrow\texttt{Label}(IS)$ and thus it is finite.

Indeed, observe that if $\langle P, d\sqcup e\rangle \xrightarrow{c_1}$, then $\langle P, d\rangle \xrightarrow{c_2}$ with $c_1 \sqsubseteq c_2$. Therefore $\texttt{Label}(IS^\star) \subseteq \downarrow\texttt{Label}(IS)$. Moreover, if $\langle P, d \sqcup e\rangle$ is added to $IS^\star$ by the rule (RD) then, by definition of $\succ_D$ (Definition 3.2.3), $e \sqsubseteq \beta$ for β being a label in $\texttt{Label}(IS^\star)$ (i.e., in $\downarrow\texttt{Label}(IS)$).

3.3.2 Complexity of the Algorithm

Here we give asymptotic bounds for the execution time of Algorithm 3.3.1. We assume that the reader is familiar with the $O(.)$ notation for asymptotic upper bounds in analysis of algorithms–see [CLRS09].

Our implementation of Algorithm 3.3.1 is a variant of the original partition refinement algorithm in [KS83] with two main differences: The computation of $IS^\star$ according to rules (IS), (RS) and (RD) (line 2, Algorithm 3.3.1) and the decision procedure for $\succ_D$ (Definition 3.2.3) needed in the redundancy checks.

We assume $\sqsubset$ to be decidable. Notice that the requirement of having some e that satisfies conditions (i), (ii) and (iii) in Definition 3.2.3 suggests that deciding whether two given transitions belong to $\succ_D$ may be costly. The following theorem, however, provides a simpler characterization of $\succ_D$ allowing us to reduce the decision problem of $\succ_D$ to that of $\sqsubset$.

Theorem 3.3.2 $\langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle \succ_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$ *iff the following conditions hold:* *(a)* $\alpha \sqsubset \beta$ *(b)* $c'' = c' \sqcup \beta$

Proof *(⇒) We assume* $\langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle \succ_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$ *namely (i), (ii) and (iii) in Definition 3.2.3 hold. Take* $e_2 = e \sqcup \alpha$ *therefore we have* $\alpha \sqcup e_2 = \alpha \sqcup e \sqcup \alpha = \alpha \sqcup e$ *that by condition (ii) is equal to* β*. Since* $\alpha \neq \beta$ *(iii), then* $\alpha \sqsubset \beta$*, i.e., condition (a) holds.*

Notice that $\alpha \sqsubseteq c'$ *since labels are added to the stores when performing transitions. By condition (ii), we have that* $c'' = c' \sqcup e = c' \sqcup \alpha \sqcup e$ *that, by condition (i), is equal to* $c' \sqcup \beta$*. Thus, (b) holds.*

(⇐) Conversely, assume that (a) and (b) hold. Since $\alpha \sqsubset \beta$*, there exists an* e_2 *s.t.* $\beta = \alpha \sqcup e_2$ *and* $\alpha \neq \beta$ *(the latter is condition (iii) in Definition 3.2.3). Now to prove (i), (ii) we take* $e = e_2 \sqcup \alpha$*. Since* $\beta = \alpha \sqcup e_2$*, then* $\beta = \alpha \sqcup e_2 = \alpha \sqcup e_2 \sqcup \alpha = \alpha \sqcup e$*, i.e., (i) holds. By (b),* $c'' = c' \sqcup \beta = c' \sqcup \alpha \sqcup e_2 = c' \sqcup e$*, i.e., (ii) also holds.*

Henceforth we shall assume that given a constraint system $\mathbf{C}$, the function $f_{\mathbf{C}}$ represents the time complexity of deciding (whether two given constraints are in) $\sqsubset$. The following is a useful corollary of the above theorem.

Corollary 3.3.1 *Given two transitions* t *and* t'*, deciding whether* $t \succ_D t'$ *takes* $O(f_{\mathbf{C}})$ *time.*

Remark 3.3.1 *We introduced* $\succ_D$ *as in Definition 3.2.3 as natural adaptation of the corresponding notion in [BM09]. The simpler characterization given by the* *above theorem is due to particular properties of ccp transitions, in particular monotonicity of the store, and hence it may not hold in a more general scenario.*

Complexity. The size of the set IS^* is central to the complexity of Algorithm 3.3.1 and depends on topology of the underlying transition graph. For tree-like topologies, a typical representation of many transition graphs, one can show by using a simple combinatorial argument that the size of IS^* is quadratic w.r.t. the size of the set of reachable configurations from IS, i.e., $\texttt{Config}(IS)$. For arbitrary graphs, however, the size of IS^* may be exponential in the number of transitions between the states of $\texttt{Config}(IS)$ as shown by the following construction.

Definition 3.3.2 *Let* $P^0 = \textbf{stop}$ *and* $P^1 = P$. *Given an even number* n, *define* $s_n(n,0) = \textbf{stop}, s_n(n,1) = \textbf{ask}\ (true) \ \rightarrow\ s_n(n,0)$ *and for each* $0 \leq i < n \wedge 0 \leq j \leq 1$ *let* $s_n(i,j) = (\textbf{ask}\ (true) \ \rightarrow\ s_n(i, j \oplus 1)^{j \oplus 1} + (\textbf{ask}\ (b_{i,j}) \ \rightarrow\ \textbf{stop}) + (\textbf{ask}\ (a_i) \ \rightarrow\ s_n(i+1,j))$ *where* $\oplus$ *means addition modulo 2. We also assume that (1) for each* i,j : $a_i \sqsubset b_{i,j}$ *and (2) for each two different* i *and* i' : $a_i \not\sqsubseteq a_{i'}$, *and (3) for each two different* (i,j) *and* (i',j'): $b_{i,j} \not\sqsubseteq b_{i',j'}$.

Let $IS = \{s_n(0,0)\}$. Figure 3.5 shows the transitions for the states in $\texttt{Config}(IS)$. One can verify that the size of IS^* is indeed exponential in the number of transitions between the states of $\texttt{Config}(IS)$.

Since Algorithm 3.3.1 computes IS^* the above construction shows that on some inputs Algorithm 3.3.1 take *at least* exponential time. We conclude by stating an upper-bound on the execution time of Algorithm 3.3.1.

Theorem 3.3.3 *Let* n *be the size of the set of states* $\texttt{Config}(IS)$ *and let* m *be the number of transitions between those states. Then* $n \times 2^{O(m)} \times f_{\mathbf{C}}$ *is an upper bound for the running time of Algorithm 3.3.1.*

Proof Let N be the size of the set of states IS^* and let M be the number of transitions between those states. One can verify that each state s in IS^* corresponds to a state of s' in $\texttt{Config}(IS)$ so that s and s' have the same process and the store of s results from some least upper bound of the stores in the transitions between the states of $\texttt{Config}(IS)$. Hence, N is bounded by $O(n \times 2^m)$. Similarly, we can conclude that M is bounded by $O(m \times 2^m)$. Notice that we need to check for $\succ_D$

in each transition (between the states) in IS^*. With the help of Corollary 3.3.1 we conclude that constructing IS^* takes $O(f_{\mathbf{C}} \times m \times 2^m)$ time.

Let N be the size of the set of states IS^* and let M be the number of transitions between those states. Following the implementation of [KS83] and taking into account the checks for irredundacy we can obtain a $O(NM^3)$ time bound for the overall executions of the iterations of Algorithm 3.3.1. From the above upper-bounds for N and M it follows that $n \times 2^{O(m)} \times f_{\mathbf{C}}$ is indeed an upper bound for the time execution of Algorithm 3.3.1.

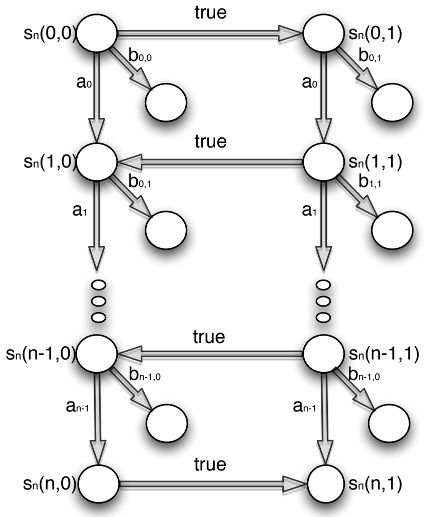


Figure 3.5: Transitions for $s_n(0,0)$ as in Definition 3.3.2.

3.4 Summary of Contributions and Related Work

In this Chapter we introduced an algorithm to verify saturated barbed bisimilarity for ccp, a new notion of strong bisimilarity for concurrent constraint programming. To the best of our knowledge, this is the first algorithm for the automatic verification of a ccp program equivalence. We do this by building upon the results of [BM09], which introduces into the standard partition refinement algorithm a new notion of behavioural equivalence which corresponds to the notion of saturated barbed bisimilarity, namely irredundant bisimilarity, which allows this algorithm to calculate the new notion of strong bisimilarity for ccp which as said before is non over-discriminative. We showed that the new algorithm terminates, since any iteration splits blocks and never fuses them. This means that $IS^{\star}$ is finite, and therefore the algorithm terminates in at most $|IS^{\star}|$ iterations.

As said before, [BM09] studies the notion of saturated bisimilarity from a general perspective and thus proposes an abstract checking procedure. Hence, one can say somehow that our partition refinement algorithm for ccp can be regarded as an instance of the one introduced in the aforementioned paper. As told all along this chapter, even though the Minimization Algorithm for Symbolic Bisimilarity focuses mainly on the symbolic semantics for interactive systems, it has a lot to do with the core of our algorithm. More precisely, some notions such as saturated and symbolic bisimilarity have been central for the development of our algorithm. Likewise, regarding the practical point of view, the concept of redundant transitions and the way Bonchi and Montanari compute redundancy and bisimilarity at the same time, is the key to achieve our goal, that is to verify the well-behaved notion of strong bisimilarity presented in Definition 2.4.2 in Section 2.4.

Chapter 4

Reducing Weak to Strong Bisimilarity in CCP

'Think simple' as my old master used to say - meaning reduce the whole of its parts into the simplest terms, getting back to first principles.
– Frank Lloyd Wright.

In Chapter 2 we presented weak bisimilarity, a central behavioural equivalence in process calculi which is obtained from the strong case [SR90] by taking into account only the actions that are observable in the system. In Chapter 3 we gave an algorithm based on the partition refinement one [KS83] to chek strong bisimilarity for ccp (Definition 2.4.2 in Section 2.4). Typically, the standard partition refinement algorithm can also be used for deciding weak bisimilarity simply by using Milner's reduction from weak to strong bisimilarity; a technique referred to as *saturation* (Chapter 3, The algorithmics of bisimilarity in [San11]). In this chapter we demonstrate that, because of its labeled transitions, the above-mentioned saturation technique does not work for ccp. We give an alternative reduction from weak ccp bisimilarity to the strong one that allows us to use the ccp partition refinement algorithm for deciding this equivalence.

4.1 Background

4.1.1 Reducing Weak to Strong Bisimilarity

We shall briefly present the standard way for computing weak bisimilarity [Mil80] by means of the partition refinement algorithm [KS83] and the saturation method proposed by Milner [Mil80].

A major dichotomy among behavioural equivalences concerns the standard notions of strong and weak equivalences. In strong equivalences, all the transitions performed by a system are deemed observable. In weak equivalences, instead, internal transitions (usually denoted by τ) are unobservable. On the one hand, weak equivalences [Mil80] are more abstract (and thus closer to the intuitive notion of behaviour); on the other hand, strong equivalences [Mil80] are usually much easier to be checked (for instance, in [LPSS11], a standard strong equivalence which is computable for a Turing complete formalism is introduced).

The standard notion of strong bisimilarity [Mil80] is one of the most studied behavioral equivalence and many algorithms (e.g., [VM94, Fer89, FGM^{+}98]) have been developed to check whether two systems are equivalent up to this definition of strong bisimilarity. Among these, the partition refinement algorithm [KS83] is one of the best known.

Weak bisimilarity [Mil80] can be computed by reducing it to strong bisimilarity [Mil80]. Given an LTS $\xrightarrow{a}$, one can build $\overset{a}{\Longrightarrow}$ as follows.

$$\frac{P \xrightarrow{a} Q}{P \overset{a}{\Longrightarrow} Q} \qquad \frac{}{P \overset{\tau}{\Longrightarrow} P} \qquad \frac{P \overset{\tau}{\Longrightarrow} P_1 \overset{a}{\Longrightarrow} Q_1 \overset{\tau}{\Longrightarrow} Q}{P \overset{a}{\Longrightarrow} Q}$$

Since weak bisimilarity [Mil80] on $\xrightarrow{a}$ coincides with strong bisimilarity [Mil80] on $\overset{a}{\Longrightarrow}$, then one can check weak bisimilarity with the algorithms for strong bisimilarity on the new LTS $\overset{a}{\Longrightarrow}$.

In this Chapter, first we show that the standard method for reducing weak to strong bisimilarity (see Chapter 3 of [San11], [Mil80] or [Mil99]) does not work for ccp and then we provide a way out of the impasse. Let us recall Section 2.3

where a ccp transition is:

$$\gamma \xrightarrow{\alpha} \gamma'$$

where γ, γ' are configurations and α is the label that represents a minimal information (from the environment) that needs to be added to the store of configuration γ to evolve from γ into γ'. Therefore, our solution can be readily explained by observing that the labels in the LTS of a ccp agent are constraints (in other words, they are "the minimal constraints" that the store should satisfy in order to make the agent progress). These constraints form a lattice where the least upper bound (denoted by $\sqcup$) intuitively corresponds to conjunction and the bottom element is the constraint $true$ (as expected, transitions labeled by $true$ are internal transitions, corresponding to the τ moves in standard process calculi). Now, rather than closing the transitions just with respect to $true$, we need to close them w.r.t. all the constraints. Formally we build the new LTS with the following rules.

$$\frac{\gamma \xrightarrow{a} \gamma'}{\gamma \overset{a}{\Longrightarrow} \gamma'} \qquad \frac{}{\gamma \overset{true}{\Longrightarrow} \gamma} \qquad \frac{\gamma \overset{a}{\Longrightarrow} \gamma' \overset{b}{\Longrightarrow} \gamma''}{\gamma \overset{a \sqcup b}{\Longrightarrow} \gamma''}$$

Note that, since $\sqcup$ is idempotent, if the original LTS $\xrightarrow{a}$ has finitely many transitions, then also $\overset{a}{\Longrightarrow}$ is finite. This allows us to use the Algorithm 3.3.1 in Section 3.3 to check weak bisimilarity on (the finite fragment) of concurrent constraint programming. We have implemented this procedure in a tool that is available at `http://www.lix.polytechnique.fr/~andresaristi/checkers/`. To the best of our knowledge, this is the first tool for checking weak equivalence of ccp programs.

Standard Reduction from Weak to Strong Bismilarity

As pointed out in the literature (Chapter 3 from [SR12]), in order to compute weak bisimilarity [Mil80] we can use the partition refinement introduced in Section 3.1.1 . The idea is to start from the graph generated via the operational semantics and then *saturate* it using the rules described in Table 4.1 to produce a new labeled relation $\Longrightarrow$. Recall that $\longrightarrow^*$ is the reflexive and transitive closure of the relation

$\longrightarrow$. Now the problem whether two states are weakly bisimilar can be reduced to checking whether they are strongly bisimilar w.r.t. $\Longrightarrow$ using the ccp-partition refinement algorithm in Section 3.3. As we will show later on, this approach does not work in a formalism like concurrent constraint programming. We shall see that the problem involves the ccp transition labels which, being constraints, can be arbitrary combined using the lub operation $\sqcup$ to form a new one. Such a situation does not arise in CCS-like labeled transitions.

$$\text{MR1}\ \frac{\gamma \xrightarrow{\alpha} \gamma'}{\gamma \overset{\alpha}{\Longrightarrow} \gamma'} \qquad \text{MR2}\ \frac{}{\gamma \overset{true}{\Longrightarrow} \gamma} \qquad \text{MR3}\ \frac{\gamma \overset{true}{\Longrightarrow} \gamma_1 \overset{\alpha}{\Longrightarrow} \gamma_2 \overset{true}{\Longrightarrow} \gamma'}{\gamma \overset{\alpha}{\Longrightarrow} \gamma'}$$

Table 4.1: Milner's Saturation Method

Notation 4.1.1 *When the label of a transition is $true$ we will omit it. Namely, henceforth we will use $\gamma \longrightarrow \gamma'$ and $\gamma \Longrightarrow \gamma'$ to denote $\gamma \xrightarrow{true} \gamma'$ and $\gamma \overset{true}{\Longrightarrow} \gamma'$.*

Recall that the labeled semantics in Section 2.3 can be related to the reduction semantics in Section 2.1.3 via the lemmata: Lemma 3.2.1 (soundness) and Lemma 3.2.2 (completeness). (See Section 2.4). The first one deals with the fact that if a process is able to perform an action using some (minimal) information from the environment, then providing such information to the process (in its store) will allow to perform a reduction instead. Now completeness goes from reductions to labeled transitions: if a configuration can perform a reduction then the idea is that one could use a piece of its store as a label, and the process should be able to arrive to the same result by means of a labeled transition. In this case, such label might not be exactly the piece of information we took from the store but something smaller, thus the resulting store could also be smaller. We must mention that Lemma 3.2.1 and Lemma 3.2.2 are essential for proving the correspondence between the alternative notions of saturated strong barbed bisimilarity in ccp (see Definition 2.2.2 in Section 2.2), symbolic bisimilarity (see Definition 3.2.6 in Section 3.2.6) and irredundant bisimilarity (see Definition 3.2.7 in Section 3.2.6). As a matter of fact there is an arising problem we must deal with. Completeness does not hold when we reduce weak bisimilarity for ccp into the strong one by means of using Milner's saturation method, i.e., replacing $\longrightarrow$ with $\Longrightarrow$ in Table 4.1. Henceforth, we will show that the standard reduction from weak to strong does not work.

Incompleteness of Milner's Saturation Method in CCP

As mentioned at the beginning of this section, the standard approach for deciding weak equivalences is to add some transitions to the original processes, so-called *saturation*, and then check for the strong equivalence. In calculi like CCS, such saturation consists in forgetting about the internal actions that make part of a sequence containing one observable action (Table 4.1). However, for ccp this method does not work. The problem is that the relation proposed by Milner is not complete for ccp, hence the relation among the saturated, symbolic and irredundant equivalences is broken. In the next section we will provide a stronger saturation, which is complete, and allows us to use the ccp partition refinement to compute $\dot{\approx}_{sb}$.

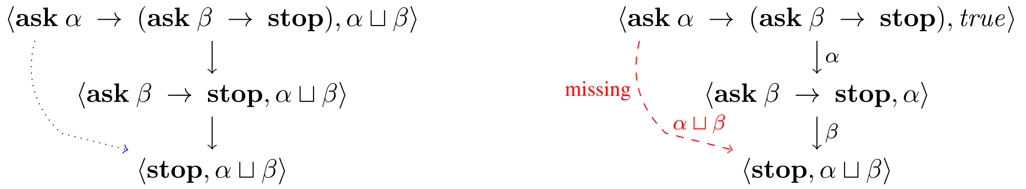
Let us show why Milner's approach does not work. First, we need to introduce formally the concept of *completeness* for a given relation.

Definition 4.1.1 *We say that a relation $\overset{\beta}{\leadsto} \subseteq Conf \times Con_0 \times Conf$ is complete if whenever $(\langle P, c \sqcup a\rangle, true, \langle P', c'\rangle) \in \overset{\beta}{\leadsto}$ then there exist $\alpha, b \in Con_0$ s.t. $\langle P, c\rangle \overset{\alpha}{\leadsto} \langle P', c''\rangle$ where $\alpha \sqcup b = a$ and $c'' \sqcup b = c'$.*

Notice that $\longrightarrow$ (see Table 2.1 in Section 2.1.3 for the definition of this relation) is complete (see Lemma 3.2.2). Now Milner's method defines a new relation $\Longrightarrow$ using the rules in Table 4.1, but it turns out not to be complete.

Proposition 4.1.1 *The relation $\Longrightarrow$ defined in Table 4.1 is not complete.*

Proof *We will show a counter-example where the completeness for $\Longrightarrow$ does not hold. Let $P = \textbf{ask}\ \alpha \rightarrow (\textbf{ask}\ \beta \rightarrow \textbf{stop})$ and $d = \alpha \sqcup \beta$. Now consider the transition $\langle P, d\rangle \Longrightarrow \langle \textbf{stop}, d\rangle$ and let us apply the completeness lemma, we can take $c = true$ and $a = \alpha \sqcup \beta$, therefore by completeness there must exist b and λ s.t. $\langle P, true\rangle \overset{\lambda}{\Longrightarrow} \langle \textbf{stop}, c''\rangle$ where $\lambda \sqcup b = \alpha \sqcup \beta$ and $c'' \sqcup b = d$. However,* *notice that the only transition possible is $\langle P, true\rangle \overset{\alpha}{\Longrightarrow} \langle \mathbf{ask}\ \beta \rightarrow \mathbf{stop}, \alpha\rangle$, hence completeness does not hold since there is no transition from $\langle P, true\rangle$ to $\langle \mathbf{stop}, c''\rangle$ for some c''. Figure 4.1 illustrates the problem.*



$\langle \mathbf{ask}\ \alpha \rightarrow (\mathbf{ask}\ \beta \rightarrow \mathbf{stop}), \alpha \sqcup \beta\rangle$
$\langle \mathbf{ask}\ \beta \rightarrow \mathbf{stop}, \alpha \sqcup \beta\rangle$
$\langle \mathbf{stop}, \alpha \sqcup \beta\rangle$
$\langle \mathbf{ask}\ \alpha \rightarrow (\mathbf{ask}\ \beta \rightarrow \mathbf{stop}), true\rangle$
α
missing
$\langle \mathbf{ask}\ \beta \rightarrow \mathbf{stop}, \alpha\rangle$
$\alpha \sqcup \beta$
β
$\langle \mathbf{stop}, \alpha \sqcup \beta\rangle$

Figure 4.1: Counterexample for Completeness using Milner's Sat. Method. (Cycles from MR2 are omitted). Both graphs are obtained by applying the rules in Table 4.1

We can now use this fact to see why the method does not work for computing $\dot{\approx}_{sb}$ (see Definition 2.2.2 in Section 2.2.1) using the ccp partition refinement algorithm. First, let us redefine some concepts using the new relation $\Longrightarrow$. Because of condition (i) in $\dot{\approx}_{sb}$, we need a new definition of barbs, namely *weak barbs w.r.t. $\Longrightarrow$*.

Definition 4.1.2 (Weak barb w.r.t. $\Longrightarrow$) *We say γ has a weak barb e w.r.t. $\Longrightarrow$ (written $\gamma\lightning_e$) if $\gamma \Longrightarrow^* \gamma' \downarrow_e$.*

Using this notion, we introduce Symbolic and Irredundant bisimilarity w.r.t. $\Longrightarrow$, denoted by $\dot{\sim}_{sym}^{\Longrightarrow}$ and $\dot{\sim}_{I}^{\Longrightarrow}$ respectively. They are defined as in Definition 3.2.6 and Definition 3.2.7 in Section 3.2.6, where in condition (i) weak barbs ($\Downarrow$) are replaced with $\lightning$ and in condition (ii) the relation is now $\Longrightarrow$. More precisely:

Definition 4.1.3 (Symbolic Bisimilarity over $\Longrightarrow$) *A symbolic bisimulation is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$ with $\gamma_1 = \langle P, c\rangle$ and $\gamma_2 = \langle Q, d\rangle$:*

(i) if $\gamma_1\lightning_e$ then $\gamma_2\lightning_e$,

(ii) if $\langle P, c\rangle \overset{\alpha}{\Longrightarrow} \langle P', c'\rangle$ then there exists a transition $t = \langle Q, d\rangle \overset{\beta}{\Longrightarrow} \langle Q', d''\rangle$ s.t. $t \vdash_D \langle Q, d\rangle \overset{\alpha}{\Longrightarrow} \langle Q', d'\rangle$ and $\langle P', c'\rangle \mathcal{R} \langle Q', d'\rangle$

We say that γ_1 and γ_2 are symbolic bisimilar over $\Longrightarrow$ ($\gamma_1 \dot{\sim}^{\Longrightarrow}_{sym} \gamma_2$) if there exists a symbolic bisimulation over a relation $\mathcal{R}$ which is a symbolic bisimulation over $\Longrightarrow$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Definition 4.1.4 (Irredundant Bisimilarity over $\Longrightarrow$) *An irredundant bisimulation is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$:*

(i) if $\gamma_1 \lightning_e$ then $\gamma_2 \lightning_e$,

(ii) if $\gamma_1 \overset{\alpha}{\Longrightarrow} \gamma_1'$ and it is irredundant in $\mathcal{R}$ then there exists γ_2' s.t. $\gamma_2 \overset{\alpha}{\Longrightarrow} \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$.

We say that γ_1 and γ_2 are irredundant bisimilar over $\Longrightarrow$ ($\gamma_1 \dot{\sim}^{\Longrightarrow}_{I} \gamma_2$) if there exists an irredundant bisimulation over a relation $\mathcal{R}$ which is an irredundant bisimulation over $\Longrightarrow$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

One would expect that since $\dot{\sim}_{sb} = \dot{\sim}_{sym} = \dot{\sim}_I$ then the natural consequence will be that $\dot{\approx}_{sb} = \dot{\sim}^{\Longrightarrow}_{sym} = \dot{\sim}^{\Longrightarrow}_{I}$, given that these new notions are supposed to be the weak versions of the former ones when using the saturation method. However, completeness is necessary for proving $\dot{\sim}_{sb} = \dot{\sim}_{sym} = \dot{\sim}_I$, and from Proposition 4.1.1 we know that $\Longrightarrow$ is not complete hence we might expect $\dot{\approx}_{sb} \neq \dot{\sim}^{\Longrightarrow}_{sym} \neq \dot{\sim}^{\Longrightarrow}_{I}$. In fact, the following counter-example shows these inequalities.

Example 4.1.1 *Let P, P' and Q as in Figure 4.3. The figure shows $\langle P, true\rangle$ and $\langle Q, true\rangle$ after we saturate them using Milner's method. First, notice that $\langle P, true\rangle \dot{\approx}_{sb} \langle Q, true\rangle$, since there exists a saturated weak barbed bisimulation $\mathcal{R} = \{(\langle P, true\rangle, \langle Q, true\rangle)\} \cup id$. However, $\langle P, true\rangle \not\dot{\sim}^{\Longrightarrow}_{I} \langle Q, true\rangle$. To prove that, we need to pick an irredundant transition from $\langle P, true\rangle$ or $\langle Q, true\rangle$ (after saturation) s.t. the other cannot match. Thus, take $\langle Q, true\rangle \xrightarrow{\alpha \sqcup \beta} \langle \mathbf{tell}(c), \alpha \sqcup \beta\rangle$ which is irredundant and given that $\langle P, true\rangle$ does not have a transition with $\alpha \sqcup \beta$ then we know that there is no irredundant bisimulation containing $(\langle P, true\rangle, \langle Q, true\rangle)$ therefore $\langle P, true\rangle \not\dot{\sim}^{\Longrightarrow}_{I} \langle Q, true\rangle$. Using the same reasoning we can also show that $\dot{\approx}_{sb} \neq \dot{\sim}^{\Longrightarrow}_{sym}$.*

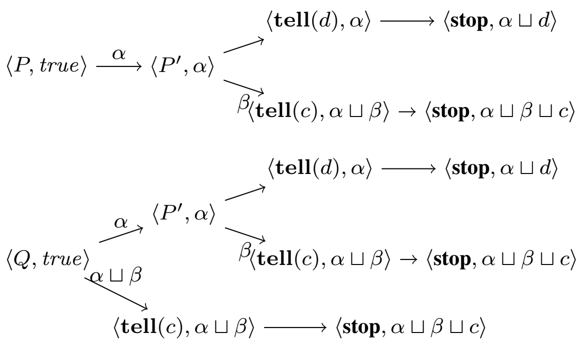


Figure 4.2: Transitions from $\langle P, \mathit{true}\rangle$ and $\langle Q, \mathit{true}\rangle$

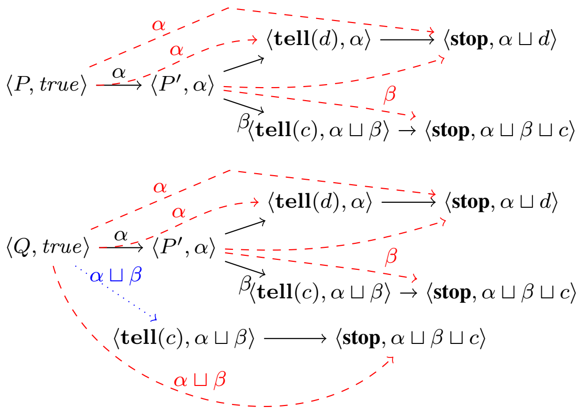


Figure 4.3: Saturated Transitions from $\langle P, \mathit{true}\rangle$ and $\langle Q, \mathit{true}\rangle$.

Let $P = \textbf{ask}\ (\alpha) \rightarrow P'$, $P' = (\textbf{ask}\ (\beta) \rightarrow \textbf{tell}(c)) + (\textbf{ask}\ (\mathit{true}) \rightarrow \textbf{tell}(d))$ and $Q = P + (\textbf{ask}\ (\alpha \sqcup \beta) \rightarrow \textbf{tell}(c))$. The figure represents $\langle P, \mathit{true}\rangle$ and $\langle Q, \mathit{true}\rangle$ after being saturated using Milner's method (cycles from MR2 omited). The red dashed transitions are the new ones added by the rules in Table 4.1. The blue dotted transition is the (irredundant) one that $\langle Q, \mathit{true}\rangle$ can take but $\langle P, \mathit{true}\rangle$ cannot match, therefore showing that $\langle P, \mathit{true}\rangle \overset{\Longrightarrow}{\dot{\not\sim}}_I \langle Q, \mathit{true}\rangle$.

4.2 Defining a New Saturation Method for CCP

In this section we shall provide a method for deciding weak bisimilarity ($\dot{\approx}$) in ccp. As shown in Section 4.1.1, the usual method for deciding weak bisimilarity does not work for ccp. We shall proceed by redefining $\Longrightarrow$ in such a way that it is sound and complete for this language. Then we prove that, w.r.t. $\Longrightarrow$, symbolic and irredundant bisimilarity coincide with $\dot{\approx}_{sb}$, i.e. $\dot{\approx}_{sb} = \dot{\sim}^{\Longrightarrow}_{sym} = \dot{\sim}^{\Longrightarrow}_{I}$. We therefore conclude that the partition refinement algorithm in Section 3.3 can be used to verify $\dot{\approx}_{sb}$ w.r.t. $\Longrightarrow$.

If we analyze the counter-example to completeness (see Figure 4.1), one can see that the problem arises because of the nature of the labels in ccp, namely using this method $\langle \mathbf{ask}\ \alpha \rightarrow (\mathbf{ask}\ \beta \rightarrow \mathbf{stop}), \mathit{true}\rangle$ does not have a transition with $\alpha \sqcup \beta$ to $\langle \mathbf{stop}, \alpha \sqcup \beta \rangle$, hence that fact can be exploited to break the relation among the weak equivalences. Following this reasoning, instead of only forgetting about the silent actions we also take into account that labels in ccp can be added together. Thus we have a new rule that creates a new transition for each two consecutive ones, whose label is the lub of the labels in them. This method can also be thought as the reflexive and transitive closure of the labeled relation ($\xrightarrow{\alpha}$). This relation turns out to be sound and complete and it can be used to decide $\dot{\approx}_{sb}$.

4.2.1 A New Saturation Method

Formally, our new relation $\Longrightarrow$ is defined by the rules in Table 4.2. For simplicity, we are using the same arrow $\Longrightarrow$ to denote this relation. Consequently the definitions of weak barbs, symbolic and irredundant bisimilarity are now interpreted w.r.t. $\Longrightarrow$ ($\lightning$, $\dot{\sim}^{\Longrightarrow}_{sym}$ and $\dot{\sim}^{\Longrightarrow}_{I}$ respectively).

$$\text{R-Tau}\ \frac{}{\gamma \Longrightarrow \gamma} \qquad \text{R-Label}\ \frac{\gamma \xrightarrow{\alpha} \gamma'}{\gamma \overset{\alpha}{\Longrightarrow} \gamma'} \qquad \text{R-Add}\ \frac{\gamma \overset{\alpha}{\Longrightarrow} \gamma' \overset{\beta}{\Longrightarrow} \gamma''}{\gamma \overset{\alpha \sqcup \beta}{\Longrightarrow} \gamma''}$$

Table 4.2: New Labeled Transition System.

We now prove that $\lightning$ coincides with $\Downarrow$, given that a transition under the new relation corresponds to a sequence of reductions.

Lemma 4.2.1 *$\gamma \longrightarrow^{*} \gamma'$ iff $\gamma \Longrightarrow \gamma'$.*

Proof *($\Rightarrow$) We can decompose $\gamma \longrightarrow^{*} \gamma'$ as follows $\gamma \longrightarrow \gamma_1 \longrightarrow \ldots \longrightarrow \gamma_i \longrightarrow \gamma'$, now we proceed by induction on i. The base case is $i = 0$, then $\gamma \longrightarrow \gamma'$ and by rule* R-Label *we have $\gamma \Longrightarrow \gamma'$. For the inductive step, first we have by induction hypothesis that $\gamma \longrightarrow^{i} \gamma_i$ implies $\gamma \Longrightarrow \gamma_i$ (1), on the other hand, by rule* R-Label *on $\gamma_i \longrightarrow \gamma'$ we can deduce $\gamma_i \Longrightarrow \gamma'$ (2). Finally by* R-Add *on (1) and (2) $\gamma \Longrightarrow \gamma'$.*

($\Leftarrow$) We proceed by induction on the depth of the inference of $\gamma \Longrightarrow \gamma'$. First, using R-Tau *, we can directly conclude $\gamma \longrightarrow^{*} \gamma$. Secondly, using* R-Label *, $\gamma \Longrightarrow \gamma'$ implies that $\gamma \longrightarrow \gamma'$. Finally, using* R-Add *, we get $\gamma \Longrightarrow \gamma'' \Longrightarrow \gamma'$ and by induction hypothesis this means that $\gamma \longrightarrow^{*} \gamma'' \longrightarrow^{*} \gamma'$ therefore $\gamma \longrightarrow^{*} \gamma'$.*

Using this lemma, it is straightforward to see that the notions of weak barbs coincide.

Lemma 4.2.2 *$\gamma \Downarrow_e$ iff $\gamma \lightning_e$.*

Proof *First, let us assume that $\gamma \Downarrow_e$ then by definition $\gamma \longrightarrow^{*} \gamma' \downarrow_e$, and from Lemma 4.2.1 we know that $\gamma \Longrightarrow \gamma' \downarrow_e$, hence $\gamma \lightning_e$. On the other hand, if $\gamma \lightning_e$ then by definition $\gamma \Longrightarrow^{*} \gamma' \downarrow_e$, if we decompose these transitions then $\gamma \Longrightarrow \ldots \Longrightarrow \gamma'$, and from Lemma 4.2.1 $\gamma \longrightarrow^{*} \ldots \longrightarrow^{*} \gamma'$, therefore $\gamma \longrightarrow^{*} \gamma' \downarrow_e$, finally $\gamma \Downarrow_e$.*

An important property that the new labeled transition system defined by the new relation $\Longrightarrow$ must fulfill is that it must be finitely branching, if $\longrightarrow$ is also finitely branching. We prove this next.

The following set represents the possible results after performing one step starting from a given configuration γ and using a relation $\longrightarrow$. Such set contains pairs of the form $[\gamma', \alpha]$ in which the first item (γ') is the configuration reached and the second one (α) is the label used for that purpose. Formally we have:

Definition 4.2.1 (Single-step Reachable Pairs) *The set of Single-step reachable pairs is defined as* $\mathsf{Reach}(\gamma, \longrightarrow) = \{[\gamma', \alpha] \mid \gamma \xrightarrow{\alpha} \gamma'\}$.

We can extend this definition to consider more than one step at a time. We will call this new set $\mathsf{Reach}^*(\gamma, \longrightarrow)$ and it is defined below.

Definition 4.2.2 (Reachable Pairs) *The set of reachable pairs is defined as* $\mathsf{Reach}^*(\gamma, \longrightarrow) = \{[\gamma', \alpha] \mid \exists \alpha_1, \dots, \alpha_n.\, \gamma \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} \gamma' \wedge \alpha = \alpha_n\}$.

For convenience, in order to project the first or the second item of such pairs we will define the functions $\mathcal{C}$ and $\mathcal{L}$ which extract the configuration and the label respectively (hence the name).

Definition 4.2.3 *The functions $\mathcal{C}$ and $\mathcal{L}$ are defined as follows, $\mathcal{C}([\gamma, \alpha]) = \{\gamma\}$ and $\mathcal{L}([\gamma, \alpha]) = \{\alpha\}$. They extend to set of pairs as expected, namely given a set of pairs $S = \{p_1, \dots\}$ then $\mathcal{L}(S) = \bigcup_{p_i \in S} \mathcal{L}(p_i)$ and similarly for $\mathcal{C}$.*

The first step to prove that $\Longrightarrow$ is finitely branching is to assume that its strong version is indeed finitely branching.

Definition 4.2.4 *We say that a relation $\leadsto$ is finitely branching if for any γ, $|\mathsf{Reach}(\gamma, \leadsto)| < \infty$.*

Now, under the assumption that $\longrightarrow$ is finitely branching, we can observe that if the amount of configurations that can be reached (in one or more steps) is finite, then the amount of labels should also be finite.

Lemma 4.2.3 *Suppose that $\longrightarrow$ is finitely branching, then for any γ, if $|\mathcal{C}(\mathsf{Reach}^*(\gamma, \longrightarrow))| < \infty$ then $|\mathcal{L}(\mathsf{Reach}^*(\gamma, \longrightarrow))| < \infty$.*

Proof *Fom the hypothesis we know that there are finite γ' that can be reached, and using the finitely branching assumption we can see that from each of those γ' there are only finitely possible α, hence the conclusion.*

Finally, the nature of the labels in ccp is one of the reasons why our new transition system works. The following Lemma illustrates the fact that when generating new labels, with the rule $\mathsf{R\text{-}Add}$ we will not add an infinite amount of those. In the following, $C^{*\sqcup}$ will represent the Kleene closure over $\sqcup$ of the set of constraints C.

Lemma 4.2.4 *Given a set of constraints C, if $|C| < \infty$ then $|C^{*\sqcup}| < \infty$.*

Proof *Follows from the idempotency of $\sqcup$ ($c \sqcup c = c$).*

Using these elements, the finitely branching of $\Longrightarrow$ follows directly under the assumption that $\longrightarrow$ is finitely branching.

Lemma 4.2.5 *Assume that $\longrightarrow$ is finitely branching, then for any γ, if $|\mathcal{C}(\mathsf{Reach}^*(\gamma, \longrightarrow))| < \infty$ then for any γ, $\Longrightarrow$ is finitely braching.*

Proof *(1) One can verify that $\mathcal{C}(\mathsf{Reach}^*(\gamma, \Longrightarrow)) = \mathcal{C}(\mathsf{Reach}^*(\gamma, \longrightarrow))$ hence from the hypothesis, $|\mathcal{C}(\mathsf{Reach}^*(\gamma, \Longrightarrow))| < \infty$.*

(2) From the hypothesis and from Lemma 4.2.3 $|\mathcal{L}(\mathsf{Reach}^*(\gamma, \longrightarrow))| < \infty$.

(3) One can check that $\mathcal{L}(\mathsf{Reach}(\gamma, \Longrightarrow)) \subseteq \mathcal{L}(\mathsf{Reach}^*(\gamma, \longrightarrow))^{*\sqcup}$ therefore from Lemma 4.2.4 and (2) $|\mathcal{L}(\mathsf{Reach}(\gamma, \Longrightarrow))| < \infty$.

Finally, from (1) and (3) we can conclude that for any γ, $\Longrightarrow$ is finitely braching.

4.2.2 A Remark about our Saturation in CCS

We assume that the reader is familiar with CCS [Mil80]. *In this section the transitions, processes and relations are understood in the context of CCS.* It is worth noticing that we could use the saturation method mentioned in the previous section for other formalisms like CCS, but unlike in ccp it would not work as intended. More precisely it will generate a transition system that is not finitely branching. Now, the actions that a process can perform need to be sequences of actions. Moreover, the rules we defined in Table 4.2 must be modified, and the result is in Table 4.3. Essentially, the lub operation ($\sqcup$) is replaced by the concatenation of actions given that now the labels used for are sequences of actions.

$$\textsf{RCCS1}\ \frac{}{P \overset{\tau}{\Longrightarrow} P} \qquad \textsf{RCCS2}\ \frac{P \overset{s}{\longrightarrow} P'}{P \overset{s}{\Longrightarrow} P'} \qquad \textsf{RCCS3}\ \frac{P \overset{s}{\Longrightarrow} P' \overset{s'}{\Longrightarrow} P''}{P \overset{s.s'}{\Longrightarrow} P''}$$

Here $s = a_1 \ldots a_n$, is a sequence of observable actions, hence for the Rule RCCS3 we will take $s.\tau = \tau.s = s$.

Table 4.3: New Labeled Transition System for CCS.

Using these rules we can now define weak bisimilarity in terms of the new relation $\Longrightarrow$ as follows.

Definition 4.2.5 (CCS-Weak Bisimilarity) *A symmetric relation $\mathcal{R}$ is a CCS-weak bisimulation if for every $(P, Q) \in \mathcal{R}$:*

- *If $P \overset{s}{\Longrightarrow} P'$ then there exists Q' s.t. $Q \overset{s}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$.*

We say that P and Q are CCS-weakly bisimilar ($P \approx Q$) iff there is a CSS-weak bisimulation containing (P, Q).

This definition resembles the standard definition of strong bisimilarity, hence we could use the procedure to verify the strong version under $\Longrightarrow$ in order to verify the weak one.

Now the problem is that by applying the rules in Table 4.3, even for a finite LTS, we could end up adding an infinite amount of transitions. I.e., $\Longrightarrow$ in Table 4.3.

The following example illustrates the problem.

Example 4.2.1 *Let us take a very simple recursive CCS process:*

$$P = a.P$$

We can easily see that this process is finitely branching because it just has one transition from the same state P to itself labeled by a. (See Figure 4.4).

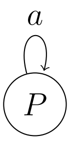


Figure 4.4: CCS Process $P = a.P$ (finitely branching).

But if we apply the rules in Table 4.3, even for Example 4.2.1, we could end up adding an infinite amount of transitions, i.e., infinitely branching (see Figure 4.5). Notice that this is not the case for our ccp transition $\Longrightarrow$ defined in Table 4.2 (see Lemma 4.2.5).

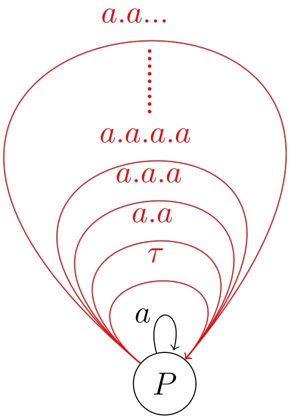


Figure 4.5: Saturated CCS Process $P = a.P$ (infinitely branching).

4.2.3 Soundness and Completeness

As mentioned before, soundness and completeness of the relation are the core properties when proving $\dot{\sim}_{sb} = \dot{\sim}_{sym} = \dot{\sim}_{I}$. We now proceed to show that our method enjoys these properties and they will allow us to prove the correspondence among the equivalences for the weak case.

In Definition 4.1.1 we have introduced the formal definition of completeness, now we shall introduce the notion of soundness in Definition 4.2.6

Definition 4.2.6 *We say that a relation $\overset{\alpha}{\rightsquigarrow} \subseteq Conf \times Con_0 \times Conf$ is sound if whenever $\langle P, c\rangle \overset{\alpha}{\rightsquigarrow} \langle P', c'\rangle$ then $\langle P, c \sqcup \alpha\rangle \overset{true}{\rightsquigarrow} \langle P', c'\rangle$.*

After introducing Definition 4.1.1 and Definition 4.2.6 we prove these properties for the relation $\Longrightarrow$.

Lemma 4.2.6 (Soundness of $\Longrightarrow$) *The relation $\Longrightarrow$ defined in Table 4.2 is sound.*

Proof *We proceed by induction on the depth of the inference of $\langle P, c\rangle \overset{\alpha}{\Longrightarrow} \langle P', c'\rangle$.*

- *Using* R-Tau *we have $\langle P, c\rangle \Longrightarrow \langle P, c\rangle$ and the result follows directly given that $\alpha = true$.*

- *Using* R-Label *we have $\langle P, c\rangle \overset{\alpha}{\Longrightarrow} \langle P', c'\rangle$ then $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$. By Lemma 3.2.1 we get $\langle P, c \sqcup \alpha\rangle \longrightarrow \langle P', c'\rangle$ and finally by rule* R-Label *$\langle P, c \sqcup \alpha\rangle \Longrightarrow \langle P', c'\rangle$.*

- *Using* R-Add *then we have $\langle P, c\rangle \overset{\beta\sqcup\lambda}{\Longrightarrow} \langle P', c'\rangle$ then $\langle P, c\rangle \overset{\beta}{\Longrightarrow} \langle P'', c''\rangle \overset{\lambda}{\Longrightarrow} \langle P', c'\rangle$ where $\beta \sqcup \lambda = \alpha$. By induction hypothesis, $\langle P, c \sqcup \beta\rangle \Longrightarrow \langle P'', c''\rangle$ (1) and $\langle P'', c'' \sqcup \lambda\rangle \Longrightarrow \langle P', c'\rangle$ (2). By monotonicity on (1), $\langle P, c \sqcup \beta \sqcup \lambda\rangle \Longrightarrow \langle P'', c'' \sqcup \lambda\rangle$ and by rule* R-Add *on this transition and (2) then, given that $\beta \sqcup \lambda = \alpha$, we obtain $\langle P, c \sqcup \alpha\rangle \Longrightarrow \langle P', c'\rangle$.*

Lemma 4.2.7 (Completeness of $\Longrightarrow$) *Relation $\Longrightarrow$ defined in Table 4.2 is complete.*

Proof *Assuming that $\langle P, c \sqcup a\rangle \Longrightarrow \langle P', c'\rangle$ then, from Lemma 4.2.1, we can say that $\langle P, c \sqcup a\rangle \longrightarrow^* \langle P', c'\rangle$ which can be written as $\langle P, c \sqcup a\rangle \longrightarrow \langle P_1, c_1\rangle \longrightarrow \ldots \longrightarrow \langle P_i, c_i\rangle \longrightarrow \langle P', c'\rangle$, we will proceed by induction on i.*

- *(Base Case) Assuming $i = 0$ then $\langle P, c \sqcup a\rangle \longrightarrow \langle P', c'\rangle$ and the result follows directly from Lemma 3.2.2 and* R-Label*.*

- *(Induction) Let us assume that $\langle P, c \sqcup a\rangle \longrightarrow^i \langle P_i, c_i\rangle \longrightarrow \langle P', c'\rangle$ then by induction hypothesis there exist β and b' s.t. $\langle P, c\rangle \overset{\beta}{\Longrightarrow} \langle P_i, c_i'\rangle$ (1) where $\beta \sqcup b' = a$ and $c_i' \sqcup b' = c_i$. Now by completeness on the last transition $\langle P_i, c_i' \sqcup b'\rangle \longrightarrow \langle P', c'\rangle$, there exists λ and b'' s.t. $\langle P_i, c_i'\rangle \xrightarrow{\lambda}$*

$\langle P', c'' \rangle$ where $\lambda \sqcup b'' = b'$ and $c'' \sqcup b'' = c'$, thus by rule R-Label *we have $\langle P_i, c_i' \rangle \overset{\lambda}{\Longrightarrow} \langle P', c'' \rangle$ (2). We can now proceed to apply rule* R-Add *on (1) and (2) to obtain the transition $\langle P, c \rangle \overset{\alpha}{\Longrightarrow} \langle P', c'' \rangle$ where $\alpha = \beta \sqcup \lambda$ and finally take $b = b''$, therefore the conditions hold $\alpha \sqcup b = \beta \sqcup \lambda \sqcup b'' = a$ and $c'' \sqcup b = c'' \sqcup b'' = c'$.*

4.3 Correspondence between $\dot{\approx}_{sb}$, $\dot{\sim}_{sym}$, $\dot{\sim}_{I}$

We show our main result of this chapter, a method for deciding $\dot{\approx}_{sb}$. Recall that $\dot{\approx}_{sb}$ is the standard weak bisimilarity for ccp (see Section 2), and it is defined in terms of $\longrightarrow$, therefore it does not depend on $\Longrightarrow$. Roughly, we start from the fact that the ccp partition refinement is able to check whether two configurations are irredundant bisimilar $\dot{\sim}_{I}$. Such configurations evolve according to a relation ($\longrightarrow$), then we provide a new way for them to evolve ($\Longrightarrow$) and we use the same algorithm to compute now $\dot{\sim}_{I}^{\Longrightarrow}$. Here we prove that $\dot{\approx}_{sb} = \dot{\sim}_{sym}^{\Longrightarrow} = \dot{\sim}_{I}^{\Longrightarrow}$ hence we give a reduction from $\dot{\approx}_{sb}$ to $\dot{\sim}_{I}^{\Longrightarrow}$ which has an effective decision procedure.

Given that the relation $\longrightarrow$ is sound and complete (see Lemma 3.2.1 and Lemma 3.2.2 in Section 3.2), then there exists a correspondence between the symbolic and irredundant bisimilarity over $\longrightarrow$ shown in Theorem 3.2.3 in Section 3.2.6. From this statement and since we have proven that new relation $\Longrightarrow$ is sound and complete, a correspondence between the symbolic and irredundant bisimilarity over $\Longrightarrow$ exists.

Corollary 4.3.1 *$\gamma \dot{\sim}_{sym}^{\Longrightarrow} \gamma'$ iff $\gamma \dot{\sim}_{I}^{\Longrightarrow} \gamma'$*

Finally, in the next two lemmata, we prove that $\dot{\approx}_{sb} = \dot{\sim}_{sym}^{\Longrightarrow}$.

Lemma 4.3.1 *If $\gamma \dot{\approx}_{sb} \gamma'$ then $\gamma \dot{\sim}_{sym}^{\Longrightarrow} \gamma'$*

Proof *We need to prove that $\mathcal{R} = \{(\langle P, c \rangle, \langle Q, d \rangle) \mid \langle P, c \rangle \dot{\approx}_{sb} \langle Q, d \rangle\}$ is a symbolic bisimulation over $\Longrightarrow$. The first condition (i) of the bisimulation follows directly from Lemma 4.2.2. As for (ii), let us assume that $\langle P, c \rangle \overset{\alpha}{\Longrightarrow} \langle P', c' \rangle$ then by soundness of $\Longrightarrow$ we have $\langle P, c \sqcup \alpha \rangle \Longrightarrow \langle P', c' \rangle$, now by Lemma 4.2.1 we obtain $\langle P, c \sqcup \alpha \rangle \longrightarrow^{*} \langle P', c' \rangle$. Given that $\langle P, c \rangle \dot{\approx}_{sb} \langle Q, d \rangle$ then from the latter transition we can conclude that $\langle Q, d \sqcup \alpha \rangle \longrightarrow^{*} \langle Q', d' \rangle$ where $\langle P', c' \rangle \dot{\approx}_{sb} \langle Q', d' \rangle$,* *hence we can use Lemma 4.2.1 again to deduce that* $\langle Q, d \sqcup \alpha\rangle \Longrightarrow \langle Q', d'\rangle$*. Finally, by completeness of* $\Longrightarrow$*, there exist* β *and* b *s.t.* $t = \langle Q, d\rangle \overset{\beta}{\Longrightarrow} \langle Q', d''\rangle$ *where* $\beta \sqcup b = \alpha$ *and* $d'' \sqcup b = d'$*, therefore* $t \vdash_D \langle Q, d\rangle \overset{\alpha}{\Longrightarrow} \langle Q', d'\rangle$ *and* $\langle P', c'\rangle \mathcal{R} \langle Q', d'\rangle$*.*

Lemma 4.3.2 *If* $\gamma \dot{\sim}_{sym}^{\Longrightarrow} \gamma'$ *then* $\gamma \dot{\approx}_{sb} \gamma'$

Proof *We need to prove that* $\mathcal{R} = \{(\langle P, c \sqcup a\rangle, \langle Q, d \sqcup a\rangle) \mid \langle P, c\rangle \dot{\sim}_{sym}^{\Longrightarrow} \langle Q, d\rangle\}$ *is a weak saturated bisimulation. First, condition (i) follows from Lemma 4.2.2 and (iii) by definition of* $\mathcal{R}$*. Let us prove condition (ii). Assume* $\langle P, c \sqcup a\rangle \longrightarrow^* \langle P', c'\rangle$ *then by Lemma 4.2.1* $\langle P, c \sqcup a\rangle \Longrightarrow \langle P', c'\rangle$*. Now by completeness of* $\Longrightarrow$ *there exist* α *and* b *s.t.* $\langle P, c\rangle \overset{\alpha}{\Longrightarrow} \langle P', c''\rangle$ *where* $\alpha \sqcup b = a$ *and* $c'' \sqcup b = c'$*. Since* $\langle P, c\rangle \dot{\sim}_{sym}^{\Longrightarrow} \langle Q, d\rangle$ *then we know there exists a transition* $t = \langle Q, d\rangle \overset{\beta}{\Longrightarrow} \langle Q', d'\rangle$ *s.t.* $t \vdash_D \langle Q, d\rangle \overset{\alpha}{\Longrightarrow} \langle Q', d''\rangle$ *and* $(\mathbf{a})\langle P', c''\rangle \dot{\sim}_{sym}^{\Longrightarrow} \langle Q', d''\rangle$*, by definition of* $\vdash_D$ *there exists* b' *s.t.* $\beta \sqcup b' = \alpha$ *and* $d' \sqcup b' = d''$*. Using soundness of* $\Longrightarrow$ *on* t *we get* $\langle Q, d \sqcup \beta\rangle \Longrightarrow \langle Q', d'\rangle$*, thus by Lemma 4.2.1* $\langle Q, d \sqcup \beta\rangle \longrightarrow^* \langle Q', d'\rangle$ *and finally by monotonicity*

$$\langle Q, d \sqcup \overbrace{\underbrace{\beta \sqcup b'}_{\alpha} \sqcup b}^{a}\rangle \longrightarrow^* \langle Q', \overbrace{d' \sqcup b'}^{d''} \sqcup b\rangle \tag{4.1}$$

Notice that the initial transition $\langle P, c \sqcup a\rangle \longrightarrow^* \langle P', c'\rangle$ *can be rewritten as* $\langle P, c \sqcup a\rangle \longrightarrow^* \langle P', c'' \sqcup b\rangle$*, therefore using the transition in* (4.1)*,* $\langle Q, d \sqcup a\rangle \longrightarrow^* \langle Q', d'' \sqcup b\rangle$*, it is left to prove that* $\langle P', c'' \sqcup b\rangle \mathcal{R} \langle Q', d'' \sqcup b\rangle$ *which follows from* $(\mathbf{a})$*.*

Using Lemma 4.3.1 and Lemma 4.3.2 we obtain the following theorem.

Theorem 4.3.1 $\langle P, c\rangle \dot{\sim}_{sym}^{\Longrightarrow} \langle Q, d\rangle$ *iff* $\langle P, c\rangle \dot{\approx}_{sb} \langle Q, d\rangle$

From the above results, we conclude that $\dot{\approx}_{sb} = \dot{\sim}_{I}^{\Longrightarrow}$. Therefore, given that using the ccp partition refinement in combination with $\Longrightarrow$ (and $\Downarrow$) we can decide $\dot{\sim}_{I}^{\Longrightarrow}$, then we can use the same procedure to check whether two configurations are in $\dot{\approx}_{sb}$.

4.4 Algorithm for the Weak Notion of the CCP Partition Refinement Algorithm

In Section 3.3 introduced Definition 3.3.1 and the closure rules to change the initialization step of the algorithm in order to be able to calculate the weak bisimilarity ($\dot{\approx}$) for ccp (we have to change $\longrightarrow$ by $\Longrightarrow$ in both cases).

Therefore instead of using the function $\mathbf{IR}$ of Algorithm 3.3.1, we refine the partitions by employing the function $\mathbf{WIR}$ defined as follows:

Definition 4.4.1 (WIR) *For all partitions $\mathcal{P}$, $\gamma_1 \, \mathbf{WIR}(\mathcal{P}) \, \gamma_2$ if*

- *if $\gamma_1 \overset{\alpha}{\Longrightarrow} \gamma_1'$ is irredundant in $\mathcal{P}$, then there exists γ_2' s.t. $\gamma_2 \overset{\alpha}{\Longrightarrow} \gamma_2'$ and $\gamma_1' \, \mathcal{P} \gamma_2'$.*

In the computation of $\mathbf{WIR}(\mathcal{P}^n)$ also states that are not reachable from the initial states IS might be needed (these are the ones needed to check redundancy: Read the comment after Definition 3.3.1 in Section 3.3).

Thus we need to change the initialization step (prerefinement) of the algorithm by including in the set $IS^\star$, i.e., all the states that are needed to check redundancy. In this particular case, this is done by using the the following closure rules:

$$(\textsc{Wis})\frac{\gamma \in IS}{\gamma \in IS^\star} \qquad\qquad (\textsc{Wrs})\frac{\gamma_1 \in IS^\star \quad \gamma_1 \overset{\alpha}{\Longrightarrow} \gamma_2}{\gamma_2 \in IS^\star}$$

$$(\textsc{Wrd})\frac{\gamma \in IS^\star \quad \gamma \overset{\alpha_1}{\Longrightarrow} \gamma_1 \quad \gamma \overset{\alpha_2}{\Longrightarrow} \gamma_2 \quad \gamma \overset{\alpha_1}{\Longrightarrow} \gamma_1 \succ_D \gamma \overset{\alpha_2}{\Longrightarrow} \gamma_3}{\gamma_3 \in IS^\star}$$

In Figure 4.6 we design the ccp-weak partition refinement algorithm.

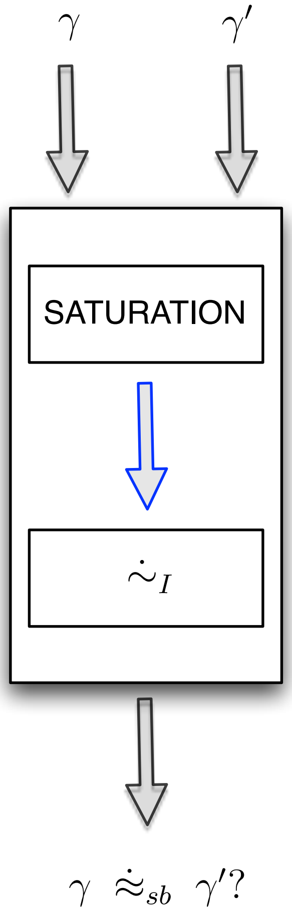


Figure 4.6: CCP-Weak Partition Refinement

Now, in Algorithm 4.4.1 we show the way to calculate the weak notion for ccp.

```
Algorithm 4.4.1 CCP-Partition-Refinement-Weak($IS$)
Initialization
    1. Compute $IS^\star$ with the rules Wis, Wrs and Wrd,
    2. $\mathcal{P}^0 := \{IS^\star_{\Downarrow d_1}\} \dots \{IS^\star_{\Downarrow d_n}\}$,
Iteration $\mathcal{P}^{n+1} :=$ WIR$(\mathcal{P}^n)$
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$
```

4.5 Summary of Contributions and Related Work

In this Chapter we showed that the relation given by Milner's saturation method is not complete for ccp. Thus, we pointed out that weak saturated barbed bisimilarity $\dot{\approx}_{sb}$ introduced in Section 2.2 cannot be computed immediately by using the ccp partition refinement algorithm for (strong) bisimilarity ccp w.r.t. to this last relation. Because of this problem, we then introduced a new relation, which is finitely branching, using a different saturation mechanism, which we showed to be complete for ccp. As a consequence we also showed that the ccp partition refinement can be used to compute $\dot{\approx}_{sb}$ using the new relation we have reached by using this new saturation method. Likewise, we have shown that although this new saturation method elaborated for ccp could be used for any other formalisms such as CCS, it would not work as desired because it would generate an infinite branching. To the best of our knowledge, this is also the first approach to verifying weak bisimilarity for ccp in an automatic way by means of any kind of algorithm. As future work, we plan to analyze different calculi with similar transition systems.

Ccp is not the only formalism where weak bisimilarity cannot be naively reduced to the strong one. Probably the first case in literature can be found in [VM94] that introduces an algorithm for checking weak open bisimilarity of π-calculus. This algorithm is rather different from ours, since it is on-the-fly [Fer89] and thus it checks the equivalence of only two given states (while our algorithm, and more generally all algorithms based on partition refinement, check the equivalence of all the states of a given LTS).

Analogous problems to the one discussed in this chapter arise in Petri nets [Sob10, BMM11], in tile transition systems [GM00] and, more generally, in the theory of reactive systems [Jen06] (the interested reader is referred to [Sob12] for an overview). In all these cases, labels form a monoid where the neutral element is the label of internal transitions. Roughly, when reducing from weak to strong bisimilarity, one needs to close the transitions with respect to the composition of the monoid (and not only with respect to the neutral element). However, in all these cases, labels composition is not idempotent (as it is for ccp) and thus a finite LTS might be transformed into an infinite one. For this reason, this procedure applied to the afore mentioned cases is not effective for automatic verification.

Chapter 5

Conclusions

I think and think for months and years. Ninety-nine times, the conclusion is false. The hundredth time I am right.
– Albert Einstein.

In this PhD thesis we introduced labeled semantics and bisimilarity for ccp. Our equivalence characterizes the observational semantics introduced in [SRP91], which is based on limits of infinite computations, by means of a co-inductive definition. It follows from [SRP91] 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 [LM00]. Syntactical bisimulation (Definition 2.4.1) as an instance of the one in [LM00], while strong and weak bisimulations (Definition 2.4.2 and Definition 2.4.3) are instances of those in [BGM09]. Furthermore, syntactical bisimulation intuitively coincides with the one in [SR90], while saturated barbed bisimulation (Definition 2.2.2) with the one in [MPSS95]. Recall that syntactical bisimilarity is too fine grained, while saturated barbed bisimulation requires the relation to be upward closed (and thus, infinite in dimension). Instead, our weak bisimulation is fully abstract and avoids the upward closure. Summarizing, the framework in [BGM09] 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 [SRP91]. Theorem 2.4.1 in Section 2.4 still holds in the presence of summation. Analogously, we could extend all the definitions to infinite constraints without invalidating these theorems.

Bisimilarity for novel languages featuring the interaction paradigms of both ccp and the π-calculus has been defined in recent work [BM08, JVP10, JBPV10]. 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. We have shown a correspondence with our semantics, although defining it is not trivial as seen in the completeness and soundness proofs in Section 2.3.

In this dissertation we provided an algorithm for verifying (strong) bisimilarity for ccp by building upon the work in [BM09]. That is, we include into the standard partition refinement algorithm a new behavioural equivalence, namely, irredundant bisimilarity. By showing that this notion corresponds to saturated bisimilarity we are able to prove that this modified partition refinement algorithm indeed verifies our well-behaved approach of strong bisimilarity. We showed that the transition relation given by Milner's saturation method is not complete for ccp (in the sense of Definition 4.2.7 and as show in Figure 4.1). As a consequence we also showed that weak saturated barbed bisimilarity $\dot{\approx}_{sb}$ (see Section 2.2) cannot be computed using the ccp partition refinement algorithm for (strong) bisimilarity for ccp w.r.t. to this transition relation. We then presented a new transition relation using another saturation mechanism and showed that it is complete for ccp. We also showed that the ccp partition refinement can be used to compute $\dot{\approx}_{sb}$, an equivalent relation to $\dot{\approx}$ as shown in Chapter 2, Theorem 2.4.2, using the new transition relation. This is an important issue, since the same piece of code works correctly with both transition relations. To the best of our knowledge, this is the first approach to verifying weak bisimilarity for ccp.

As shown in [MPSS95] 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. Likewise, we plan to investigate other calculi where the nature of their transitions systems gives rise to similar situations regarding weak and strong bisimilarity, in particular tcc [SJG], ntcc [PV01], utcc [OV08] and eccp [KPPV12].

Appendix A

Implementation and Experiments for the CCP Partition Refinement

Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.
– Antoine de Saint-Exupery.

In this Appendix we give a brief review about the main insights of the implementation and execution of the algorithm for verifying a well-behaved notion of (strong and weak) bisimilarity for ccp. We shall briefly present the programming language used, give a summary of some of the most relevant auxiliary procedures and functions, and present and analyze some examples of families of automata generated by pairs of ccp processes. We present just a brief sample with important information with the purpose of getting an idea of which details affect the performance of the ccp partition refinement algorithm. The cs we are able to handle under this implementation involves the following simple constraints of the form $var\, oper\, num$, where $var \in variables$, represents a set of variables in the problem; $oper \in \{<, >, \leq, \geq, =\}$, represents a set of operators; and $num \in [0..99]$, represents the domain of the variables.

A.1 Programming Language

As very well said in [JKOK98, Fer89], several data structures are required to represent states, transitions, classes, splitters, etc. Therefore, in order to implement Algorithm 3.3.1 and Algorithm 4.4.1 the most adequate programming language is an object oriented one. Moreover, the use of classes is important to model ccp, since we can abstract each component of the cs into a class, i.e., constraint, store, configuration, etc. Thus, we choose C++, a very well-known and vastly used language with several application domains. Although Java was another nice candidate, C++ was the the chosen one because of several benefits. It is an efficient compiler to native code, exposes low-level system facilities rather than running in a virtual machine, supports pointers, references, and pass-by-value, has an explicit memory management and provides cross-platform access to many different features available in platform-specific libraries.

Also, due to its longevity and popularity, it has several libraries useful enough to integrate its code with other languages. It is important to stress that this language also has a combination of high and low level language features. Another important characteristic of this language, is that it is implemented in a wide variety of hardware and operative systems. Taking into account our future work, its compatibility with PHP (user interface), JSON (parser) [Wil11] and obviously Gecode (library for solving constraint satisfaction problems) [Gec06] is an important issue.

A.2 Abstract Data Types

As mentioned before, we have selected an object oriented language to implement the partition refinement algorithm. This language seems appropriate for modeling all the parts and methods needed. Summing up, we implemented the following abstract data types: store, configuration, transition, automaton, and redundant class. In addition, our tool includes simple constraints, more precisely, in our case, encapsulation is an important issue because it allows hiding information in order to ensure that data structures and operators are used as intended and to make the model more obvious to the developer.

A.2.1 Implementation Layout

In this section we present as figures the general layouts of the main abstract data types. Moreover, we introduce the general overview of the proposed tool (for this layout we include future elements such as the user interface, parser and converter).

A.2.2 Layout of the Automaton

In Figure A.1 we show the design of the automaton used in our algorithm.

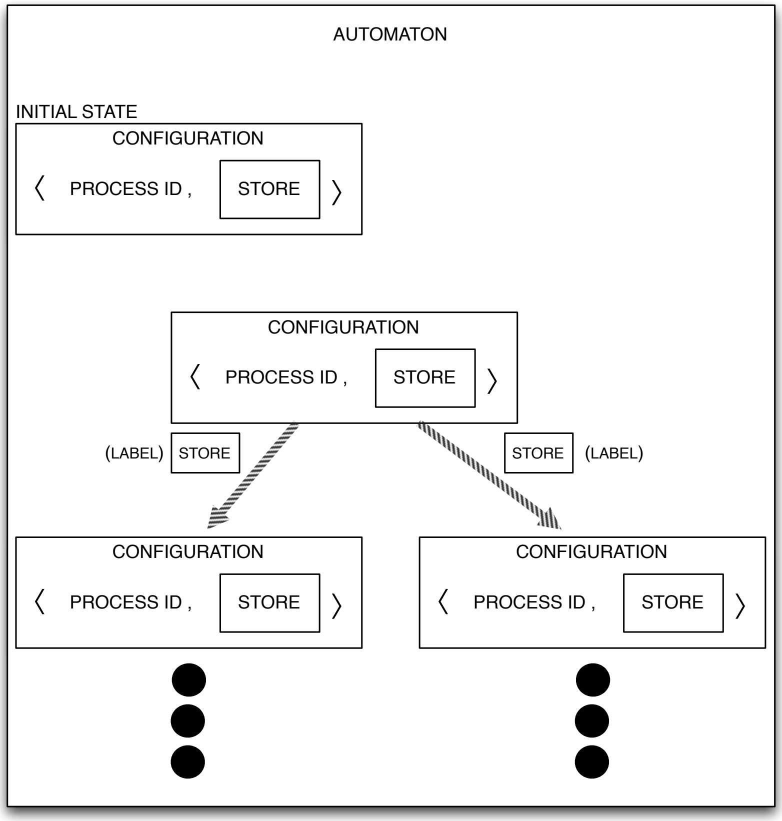


Figure A.1: A Design of an Automaton Class

A.2.3 Layout of the Redundant Class

In Figure A.2 we display a design of the redundant class we use to save the 'possibly' redundant transitions, i.e., the *dominated* ones.

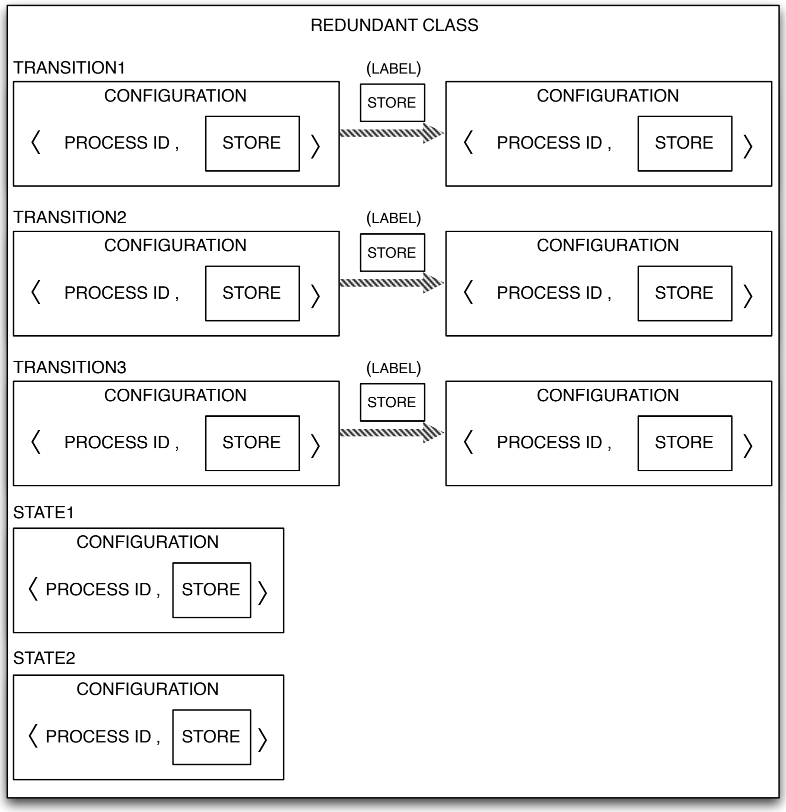


Figure A.2: A Design of a Redundant Class

A.2.4 Layout of a Tool to Verify Bisimilarity in CCP

In Figure A.3 we introduce the design of how the tool would work to verify Strong Bisimilarity in ccp.

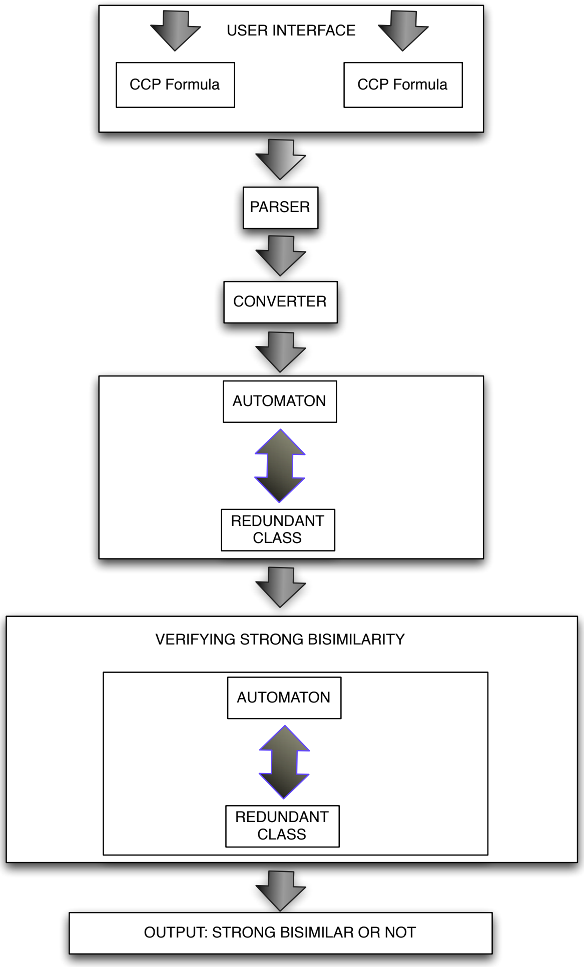


Figure A.3: Layout of a Tool to Verify Strong Bisimilarity in CCP

Now, In the Figure A.4 we present the way in which the tool would verify Weak Bisimilarity in ccp

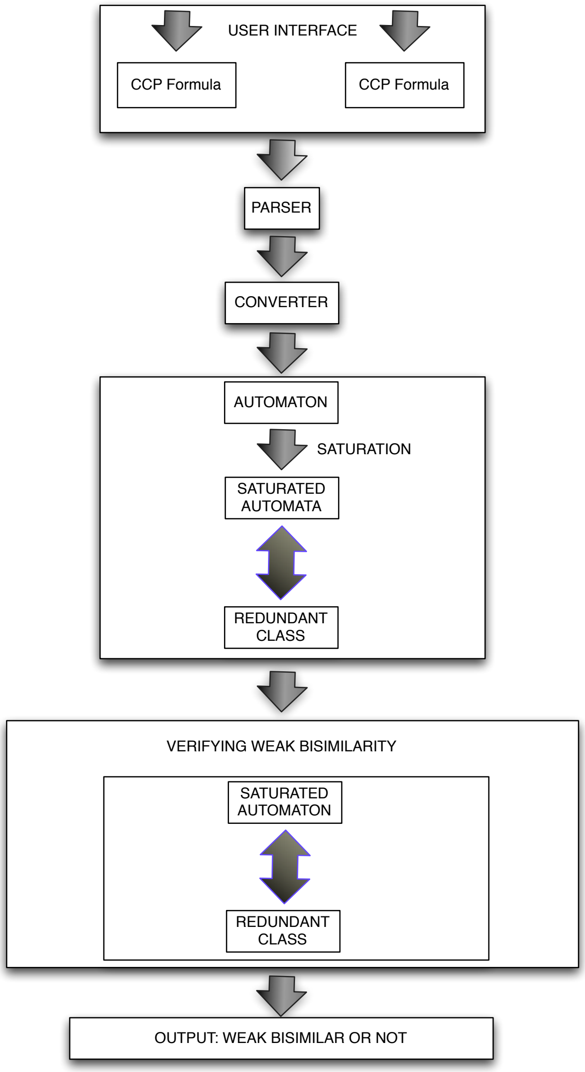


Figure A.4: Layout of a Tool to Verify Weak Bisimilarity in CCP

A.3 Procedures and Functions

A.3.1 Calculating Irredundancy

Calculating the notion of irredundancy is crucial to our algorithm. It is the central modification used on the partition refinement algorithm so it can be used to verify bisimilarity equivalence for ccp. Calculating this notion depends on two definitions. The first one of these that calculates redundant transitions (see Definition 3.2.4 in Section 3.2) and the second one, the irredundant bisimilarity (see Definition 3.2.7 in Section 3.2.6). As a previous step we define a class and a way of storing information related to 'possibly' redundant transitions. Now, as related in Section 3.2, instead of using the function **F** of Algorithm 3.1.1 in Section 3.1.1, we refine the partitions by employing the function **IR** in Definition 3.3.1 in Section 3.3 (this new function is based upon irredundant bisimilarity). However to use this function (see Algorithm A.3.1 to briefly understand how irredundancy can be checked) we might need to get involved also states that are not reachable from the initial states. This is not a difficult task since we have previously saved those artificial sates in the redundant class. We receive as an input the actual partition, the redundant classes, and the transition to be checked. We start by iterating over the redundant classes (see line 1). While doing the cycle we check two properties by means of two functions. First, if the transition belongs to the 'possibly' redundant transitions (we use *findTranRed()*, line 4). Second, if the related transitions we take out from the redundant class belong to the same partition, i.e., if they are in the same equivalent bisimilar equivalence class (line 4). If they do, then they are redundant (line 5), otherwise, we continue looping until the end (lines 1-7), therefore, if the two statements are not fulfilled, the transition is not redundant (line 8).

Algorithm A.3.1 `isredundant(partition,reds,tran)`

```
1: for (each $rd \in reds$) do
2:     $state1 \leftarrow getState1(tran, rd)$;
3:     $state2 \leftarrow getState2(tran, rd)$;
4:     if ($findTranRed(tran, rd)$ and $inSameBlock(partition, state1, state2)$) then
5:         return true;
6:     end if
7: end for
8: return false;
```

A.3.2 Strong Prerefinement

This function is another change we have made to the standard partition refinement algorithm so it can be suited to verify bisimilarity in ccp. This method makes a first partition of a block of all configurations, those reachable from the initial states of the labeled transitions and the other ones created artificially to check redundancy. We call it a strong prerefinement because it divides the only block into several blocks conformed of configurations which satisfy the same strong barb. In this way we guarantee the first item stated in the definition of irredundant bisimilarity, and then, let the new partition function **IR** deal with the token game in Definition 3.2.7 in Section 3.2.6.

Algorithm A.3.2 depicts the *barbcutS()* algorithm. It receives a set of states named states and loops two times over each state so each and every state belonging to this set can be compared with the rest of them. During this cycle if the two states, st, st'' are the same, then the former, st, will go to the the temporal set sameBarbs (see line 10) and so another cycle will start over the states. In that case, if the actual state taken as the loop starts, st', is the same as the one from the first loop, st (line 12), the first state *st* will be sent into the set of repeated states $states''$ (see line 13). Otherwise, the algorithm will loop through the set of repeated states and it will verify if the actual state st'' is already a member of $states'$ (lines 15-19). If it does not and it satisfies the same strong barb as st, then $states''$ add st and st' (since from then on they will be already repeated) and add st' to sameBarbs (lines 21-23). After getting out from the third loop *sameBarbs* will be the set of all states satisfying the same strong barb, so we will add it into the set of sets $result$ (line 28) and we will clear the temporal set *sameBarbs* to be used for another set (line 29), and we continue the same iterations until $result$ has all its components and can be returned.

Algorithm A.3.2 `barbcutS(states)`

```
$value, value1 \leftarrow true;$
for each $st \in states$ do
    for each $st'' \in states$ do
        if $st''.id = st.id$ and $entails(st''.store, state.store)$ and $entails(st.store(), st''.store)$ then
            $value \leftarrow false;$
        end if
    end for
    if $value$ and $st.id \neq \emptyset$ then
        $value \leftarrow true;$
        $sameBarbs \leftarrow sameBarbs \cup \{st\};$
        for each $st' \in states$ do
            if $st.id = st'.id$ and $entails(st.store, st'.store)$ and $entails(st'.store, st.store)$ then
                $states'' \leftarrow states'' \cup \{st\};$
            else
                for each $st''' \in states''$ do
                    if $(st'''.id = st'.id$ and $entails(st'''.store, st'.store)$ and $entails(st'.store, st'''.store)$ then
                        $value1 \leftarrow false;$
                    end if
                end for
                if $value1$ and $entails(st.store, st'.store)$ and $entails(st'.store, st.store)$ then
                    $states'' \leftarrow states'' \cup \{st'\};$
                    $states'' \leftarrow states'' \cup \{st\};$
                    $sameBarbs \leftarrow sameBarbs \cup \{st'\};$
                end if
                $value1 \leftarrow true;$
            end if
        end for
        $result \leftarrow result \cup \{sameBarbs\};$
        $sameBarbs \leftarrow \emptyset$
    else
        $value \leftarrow true;$
    end if
end for
return *result;*
```

A.3.3 Weak Prerefinement

As well as in the previous section here we present the function which prerefines the first block, but instead of refining according to the strong barbs, it does it with relation to the weak barbs. Algorithm A.3.3 works similar to Algorithm A.3.2, however, instead of using the function *entails()*, it uses de *weakEntailsAuts()*.

Algorithm A.3.3 `barbcutW(states, auts)`

```
$value, value1 \leftarrow true$;
for each $st \in states$ do
    for each $st'' \in states$ do
        if $st''.id = st.id$ and $entails(st''.store, st.store)$ and $entails(st.store, st''.store)$ then
            $value \leftarrow false$;
        end if
    end for
    if $value$ and $st.id \neq \emptyset$ then
        $value \leftarrow true$;
        $sameBarbs \leftarrow sameBarbs \cup \{st\}$;
        for each $st' \in states$ do
            if $st.id = st'.id$ and $weakEntailsAuts(st, st'.store, auts)$ and $weakEntailsAuts(st', st.store, auts)$ then
                $states'' \leftarrow states'' \cup \{st\}$;
            else
                for each $st''' \in states$ do
                    if $st'''.id = st'.id$ and $entails(st'''.store, st'.store)$ and $entails(st'.store, st'''.store)$ then
                        $value1 \leftarrow false$;
                    end if
                end for
                if $value1$ and $weakEntailsAuts(st, st'.store, auts)$ and $weakEntailsAuts(st', st.store, auts)$ then
                    $states'' \leftarrow states'' \cup \{st'\} \cup \{st\}$;
                    $sameBarbs \leftarrow sameBarbs \cup \{st'\}$;
                end if
                $value1 \leftarrow true$;
            end if
        end for
        $result \leftarrow result \cup \{sameBarbs\}$;
        $sameBarbs \leftarrow \emptyset$;
    else
        $value \leftarrow true$;
    end if
end for
return $result$;
```

A.3.4 Final Algorithm

In this section we explain the Partition Refinement algorithm for ccp. *bisimilarity_check()* receives as an input, a set of of states *S*, a set of Transitions *T*, and a type which will be used to see if we are using this algorithm to check weak bisimilarity or strong bisimilarity. This is important because if we are checking weak bisimilarity we need to reduce weak into strong bisimilarity by means of the saturation method. Finally it receives as an input a predicate which will either be a function *barbcutS()* which will make a strong prerefinement or a function *barbcutW()* which will make a weak prerefinement.

The algorithm starts by filling the automaton aut with the transitions $trans$ and the set of states *states* via the methods: *changeTransitions()* and *changeStates()* (see lines 1 and 2). By means of the type (line 3) we now know which way to take. Therefore, if we deal with the weak case, we take automaton *aut* and we do the first part of our new saturation method. We do this by adding new transitions to the old automaton. These new transitions are made out of two nodes which are not the same. Neither of them must be related by a previous transition, but a path shall exists between the nodes. The last item of these transitions is the label, made out of the least upper bound of all actions from all the transitions in the path (line 4). Afterwards, to finish the saturation method we add silent cycles to each node in automaton *aut* (line 5).

Now we have a saturated automaton, by which we are able to start the partition refinement in order to verify weak bisimilarity. We already know that the standard partition refinement does not work for this notion. Therefore, we need to calculate the possibly redundant transitions and their correspondent possible evolutions. These will take the form of a new automaton. Functions *allreds()* and *constructAuts()* generate the first redundant classes $reds$ and the first automata $auts$ respectively (lines 6 and 7). Afterwards, *changeAllAuts* (line 8) changes the automata to its final state $auts$ while *getAllReds()* does the same for $reds$ (line 9). Finally, we add the first automaton aut to the set of states *auts* (line 10). If we are dealing with the strong case we do not saturate the automaton and skip that part.

Algorithm A.3.4 `bisimilarity_check(states, trans, type, pred)`

```
$aut.changeStates(states)$;
$aut.changeTransitions(trans)$;
if $type =$ ”$weak$” then
    $addWeakTransitions1(aut)$;
    $addTau(aut)$;
    $reds \leftarrow allReds(aut)$;
    $auts \leftarrow constructAuts(aut, reds)$;
    $changeAllAuts(auts)$;
    $reds \leftarrow getAllReds(auts, reds)$;
    $auts \leftarrow auts \cup \{aut\}$;
    $tmp \leftarrow appendStates\_Auts(auts)$;
else
    $reds \leftarrow allReds(aut)$;
    $auts \leftarrow constructAuts(aut, reds)$;
    $changeAllAuts(auts)$;
    $reds \leftarrow getAllReds(auts, reds)$;
    $auts \leftarrow auts \cup \{aut\}$;
    $tmp \leftarrow appendStates\_Auts(auts1)$;
end if
$w \leftarrow pred(tmp, auts)$;
$p \leftarrow pred(tmp, auts)$;
$acts \leftarrow getAlphabet(aut)$;
while $w \neq \emptyset$ do
    assign an element from $w$ to $block$;
    for each $act \in act$ do
        for each $part \in p$ do
            if $inter(part, pIR(act, auts, reds, p, block, part)).size \neq 0$ and $\neg sbset(part, pIR(act, auts, reds, p, block, part))$ then
                $Iab \leftarrow lab \cup \{part\}$;
            end if
            $pIR \leftarrow pIR(act, auts, reds, p, block, part)$;
            for each $lb \in Iab$ do
                $Iab1 \leftarrow lab1 \cup inter(pIR, lb))$;
            end for
            for each $lb \in lab$ do
                $Iab2 \leftarrow lab2 \cup difference(lb, pIR))$;
            end for
            $p \leftarrow appending2(difference2(p, Iab), appending2(Iab1, Iab2))$;
            $w \leftarrow appending2(difference2(w, Iab), appending2(Iab1, Iab2))$;
            if $\neg w.empty$ then
                $block \leftarrow act$;
            end if
            if $\neg Iab.empty()$ then
                $Iab, lab1, lab2 \leftarrow \emptyset$;
            end if
        end for
    end for
    erase assigned element to $block$ from $w$
end while
return $p$;
```

A.3.5 Final Strong Algorithm

In this Section we show how we manage to take the final algorithm to verify the new notion of bisimilarity for ccp described in Section A.3.4. To this end, we define Algorithm A.3.5 which uses the output of Algorithm A.3.4 to check the strong notion of bisimilarity for ccp. Summing up, this algorithm checks whether the first nodes or states from both automata belong to the same set of states inside the bigger set. If it does, then they are strongly bisimilar. If not, they are not behavioural equivalent.

Algorithm A.3.5 `check_strong(p,s1,s2)`

```
for each $part \in p$ do
    if $isitin(s1, part)$ and $isitin(s2, part)$ then
        print “They are strongly Bisimilar”:
        return ;
    end if
end for
print “They are not strongly Bisimilar”;
```

A.3.6 Saturation

In this section we show how we saturate the automaton to reduce it from weak bisimilarity to strong. The first step is to construct all possible paths which may take several steps into one single step. This is achieved by following Algorithm A.3.6. The algorithm receives an automaton and adds all the possible weak paths it can take (as a single step), except the one where it stays still. It takes out all the states from the automaton and assigns them to a set of states. It goes over the set and inside this cycle it goes over the same set. Then, it checks if there is a path between the state from the first loop and the one from the second loop. Also, it verifies if they are not equal and they are not involved in a transition in the automaton. If this is fulfilled, a new transition is created where the initial state of the transition is the state from the first set; the final state of the transition is the state of the second set and the action/Store is: first the least upper bound of all the Stores collected during the path between the States and second, that result trimmed by the store of the first State, because if the Store has already information asked in the label it cannot be, therefore the Store/label must be modified taking out this information, and this is done by the function *mixminus1()*.

Algorithm A.3.6 `addWeakTransitions1(aut)`

```
$listSt \leftarrow aut.getAllStates();$
for each $st1 \in listSt$ do
    for each $st2 \in listSt$ do
        if $path(st1, st2, aut)$ and $!equalStates(st1, st2)$ and $\neg isTransition(st1, st2, aut)$ then
            $pathmix \leftarrow pathMixStr(st1, st2, aut);$
            $lab \leftarrow mixminus1(st1.store, pathmix);$
            $temp \leftarrow temp \cup \{createTransition(st1, st2, lab)\};$
        end if
    end for
end for
for each $tran \in temp$ do
    $aut.AddTransition2(tran);$
end for
return $aut;$
```

But this is not the final function needed to reduce a weak automaton into a strong automaton. There is the issue of a weak bisimilarity been able to stay still, therefore not moving through a transition to another state. We simply model this as a transition which has the same initial and final state. We follow the function in Algorithm A.3.7. This function adds the tau transitions to the automaton, the ones which go from a State to itself resembling the staying still action. It receives the automaton and takes all the states from it, creates a True Store and iterates over each State. Meanwhile it adds to the automaton a transition with the considered State as the initial and initial State of the created Transition and with the Store *true* as its label.

Algorithm A.3.7 `addTau(aut)`

```
$listSt \leftarrow aut.getAllStates();$
$sil.AddTrue();$
for each $st \in listSt$ do
    $aut.AddTransition(st, st, sil);$
end for
return $aut;$
```

A.3.7 Final Weak Algorithm

Here we follow the same path as in Section A.3.5, Therefore we just present a few lines of code which allows us to give out a final result about the two ccp programs we are comparing, that is, if they are weakly bisimilar or not. So as stated before we use the result given by Algorithm A.3.4 but specifically stating at the beginning of the algorithm by means of a string, given as a parameter, that we are checking the weak case. So therefore the algorithm will know that it needs to apply additional procedures to saturate the automata so it can be ready to apply the weak notion between the two ccp programs. As pointed out before, Algorithm A.3.8 is similar to Algorithm A.3.5.

Algorithm A.3.8 `check_weak(p,s1,s2)`

```
for each $part \in p$ do
    if $isitin(s1, part)$ and $isitin(s2, part)$ then
        print “They are weakly Bisimilar";
        return ;
    end if
end for
print “They are not weakly Bisimilar";
```

A.4 Results and Examples

In this Section we present families of generated automata resembling ccp configurations in order to analyze the behavior in time of the ccp partition refinement algorithm. We select several variables from the automata which we consider to be relevant regarding the algorithm's execution, i.e., those which will directly affect the behavior of the algorithm in relation to execution time. In order to do this we choose the following variables: transitions (Section A.4.1), nodes (Section A.4.1), percentage of transitions with the same label (Section A.4.2), percentage of configurations satisfying the same barb (Section A.4.3) and number of branches (we consider our automata as trees, therefore this name) (Section A.4.5). This analysis shall help us explain the relevance and the insights of how the main algorithm for checking bisimilarity works (in terms of execution time) with respect to the characteristics of the LTS. The examples to be used in this section are simple and satisfy all the properties of any automaton generated by a ccp program. In order to analyze the different and interesting variables that may influence the execution time of the ccp partition refinement algorithm, we shall generate preferably the same family of automata. We consider this as just changing the value of the variable to be studied in each instance of the family, and turning the other properties which are themselves variables into constants. With this procedure we isolate the variable in order to measure the real influence it has on the algorithm.

The machine used to carry out the experiments has a processor of 2.4 GHz Intel Core 2 Duo with ram memory of 4GB 1067 MHz DDR3 .

In order to make things simpler and not overfill the reader with more graphs than needed, in this chapter we will mostly constraint to the analysis of the algorithm for checking strong bisimilarity for ccp, since the result given by the weaker equivalence somehow give us the same feedback (See Chapter 4). Although that does not mean we will not take into account the execution of weak bisimilarity to check some examples. Actually, in Section A.4.5 checking this weaker notion is needed. Therefore, in that case we shall present the saturated automata of the examples. We must remark that although we study several examples per each case, we just give two diagrams of the examples for each of the analyzed properties.

A.4.1 Transitions/Nodes

Here we give several examples where the automata are almost the same. In this case we shall have two properties that variate, the number of transitions and the number of nodes. These two characteristics are the ones that we study along this section. Figure A.5 and A.6 shows a sample of the family of automata that will be used in this first case.

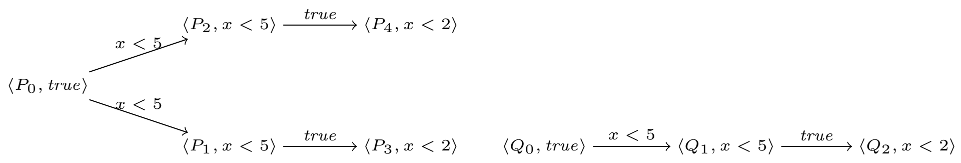

$\langle P_2, x < 5\rangle$
true
$\langle P_4, x < 2\rangle$
$x < 5$
$\langle P_0, true\rangle$
$x < 5$
$\langle P_1, x < 5\rangle$
true
$\langle P_3, x < 2\rangle$
$\langle Q_0, true\rangle$
$x < 5$
$\langle Q_1, x < 5\rangle$
true
$\langle Q_2, x < 2\rangle$

Figure A.5: Example 1 Transitions/Nodes

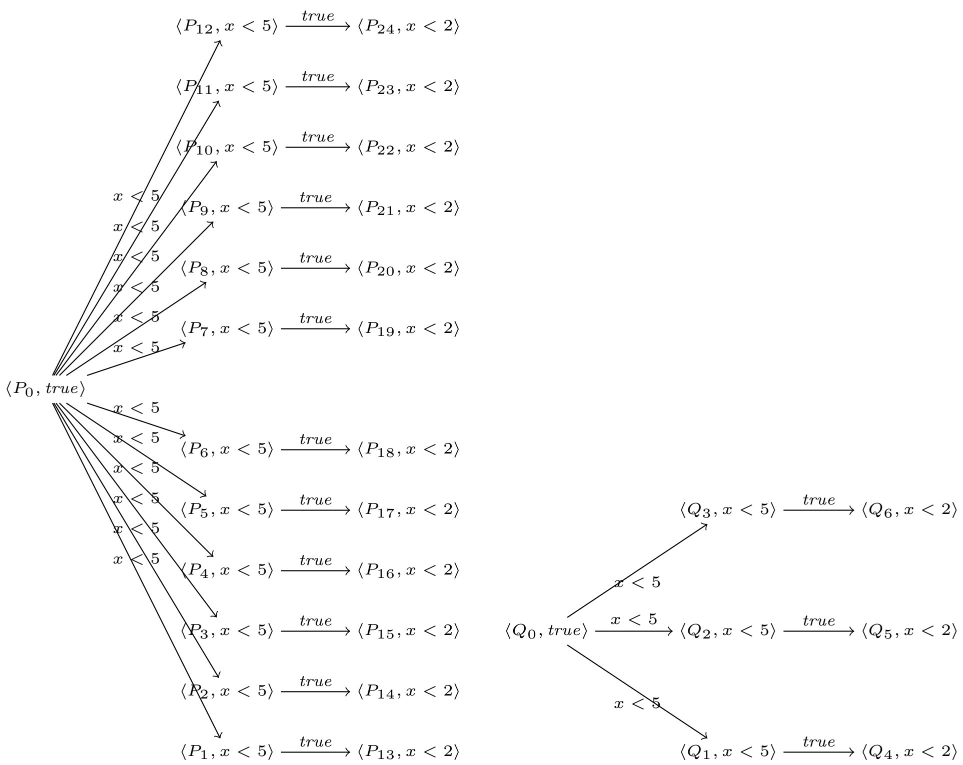

$\langle P_0, true\rangle$
$x < 5$
$\langle P_1, x < 5\rangle$ … $\langle P_{12}, x < 5\rangle$
true
$\langle P_{13}, x < 2\rangle$ … $\langle P_{24}, x < 2\rangle$
$\langle Q_0, true\rangle$
$\langle Q_1, x < 5\rangle$
$\langle Q_2, x < 5\rangle$
$\langle Q_3, x < 5\rangle$
$\langle Q_4, x < 2\rangle$
$\langle Q_5, x < 2\rangle$
$\langle Q_6, x < 2\rangle$

Figure A.6: Example 2 Transitions/Nodes

Now we have shown the several automata which are constrained to have almost the same constants and just two elements which variate.

Graphs and Analysis

From now on, we limit our attention to the execution time of the previous examples. Taking this into account, Table A.1 shows the total number of transitions and nodes vs time. These two tables and Figures A.7 and A.8 help us to understand the importance of these particular variables in relation to the execution time of the ccp partition refinement algorithm.

Processes	**Nodes**	**Transitions**	**Time**
Process 1	8	6	11 ms
Process 2	10	8	18 ms
Process 3	6	4	6 ms
Process 4	12	10	26 ms
Process 5	14	12	42 ms
Process 6	18	16	78 ms
Process 7	22	20	131 ms
Process 8	20	18	109 ms
Process 9	24	22	172 ms
Process 10	34	32	380 ms
Process 11	16	14	58 ms
Process 12	28	26	222 ms
Process 13	30	28	281 ms
Process 14	32	30	320 ms
Process 15	50	48	970 ms
Process 16	682	680	5960420 ms
Process 17	170	168	66947 ms
Process 18	426	424	2570960 ms
Process 19	218	216	522674 ms
Process 20	274	272	896211 ms

Table A.1: Time vs. Transitions/Nodes

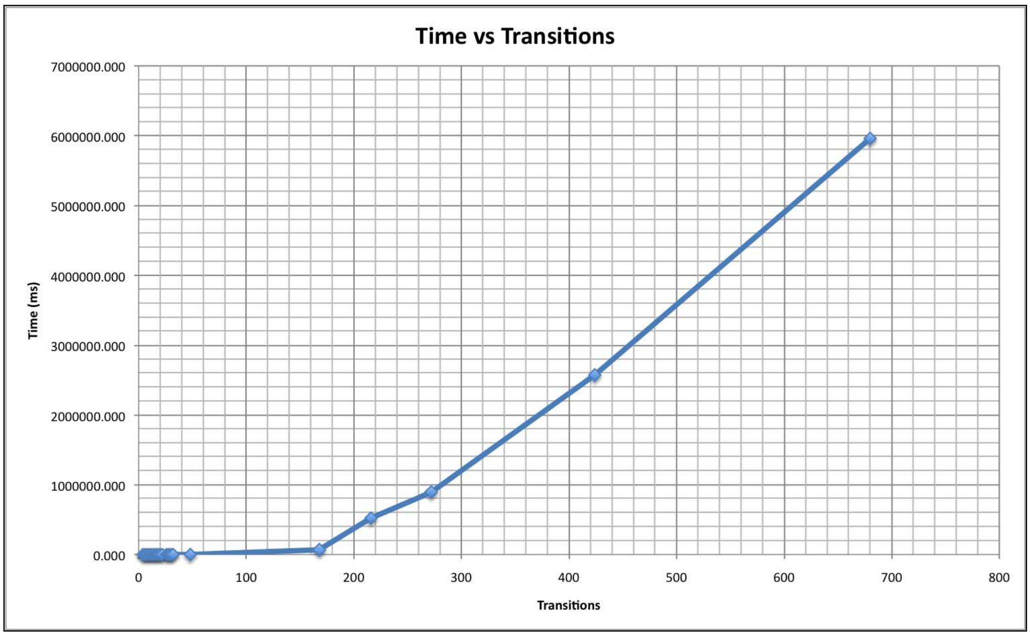


Figure A.7: Time vs. Transitions

In FigureA.7 it can be observed that the execution time increases (linearly) as the number of transitions increase. Actually, this is reasonable since the main goal of the algorithm is to partition set of states in order to refine, and it does that by means of the relations between states or configurations and the transitions. Hence, if there exist more transitions, while the execution of the algorithm, the current splitter (which is the set of states chosen to divide and refine iteratively the set or sets of states) must verify if all the possible existing transitions can be able to separate one or more states from the current set of states in each step. Therefore, if in each step, the splitter needs to check more transitions in order to refine each set of states, the algorithm will take more time to finish its execution.

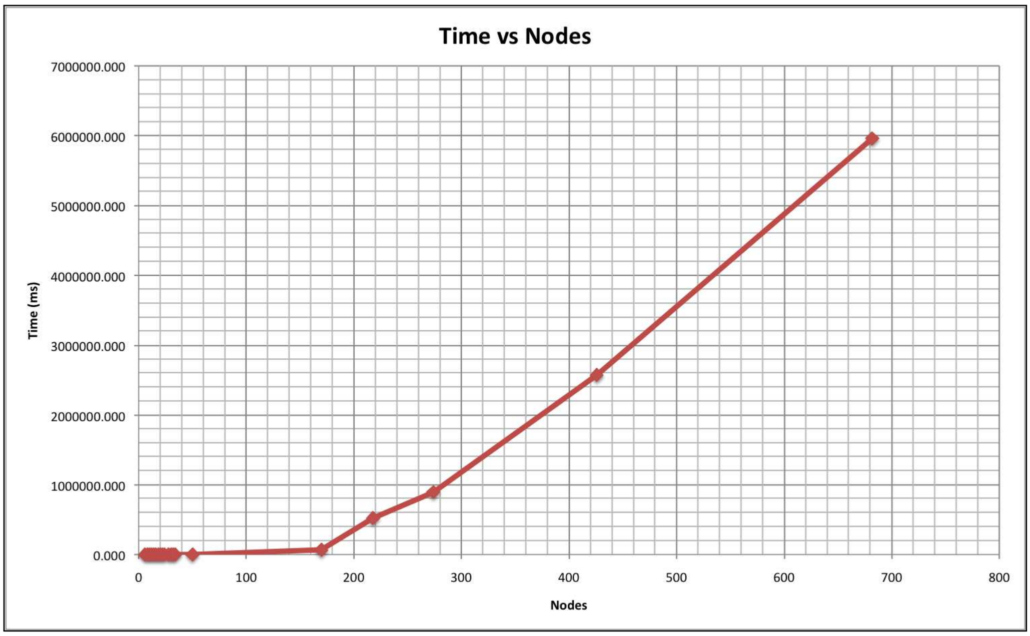


Figure A.8: Time vs. Nodes

Regarding Figure A.8 we can state that as Figure A.7 the graph resembles a linear growth which shows us that if there are more nodes in the automata, the algorithm will take longer to finish. This is feasible since most of the times if they are more nodes there are more transitions. Thus we could use the the same analysis used for the amount of transitions. Moreover, since there are more configurations or nodes in order to refine, during the algorithm run, the splitter must go over each set which will have more components. This, in order to see which of these states can be separated so the partition can be refined. Hence, if the quantity of this nodes is bigger then the execution time will be higher.

A.4.2 Percentage of Same Labels

As already seen in Section A.4.1 we shall present and analyze some examples in order to study one particular property regarding the automata generated by the ccp processes. We take the percentage of transitions with the same labels (to refer to a percentage we must include in the examples just one label, in this particular case, the label $true$ that can be repeated over the transitions of the automata). Meanwhile the other labels shall remain constant. This in order to guarantee that the graphical result will resemble only the impact of the actual variable in the execution of the algorithm).

The following figures show the automata that will be used as our case of study:

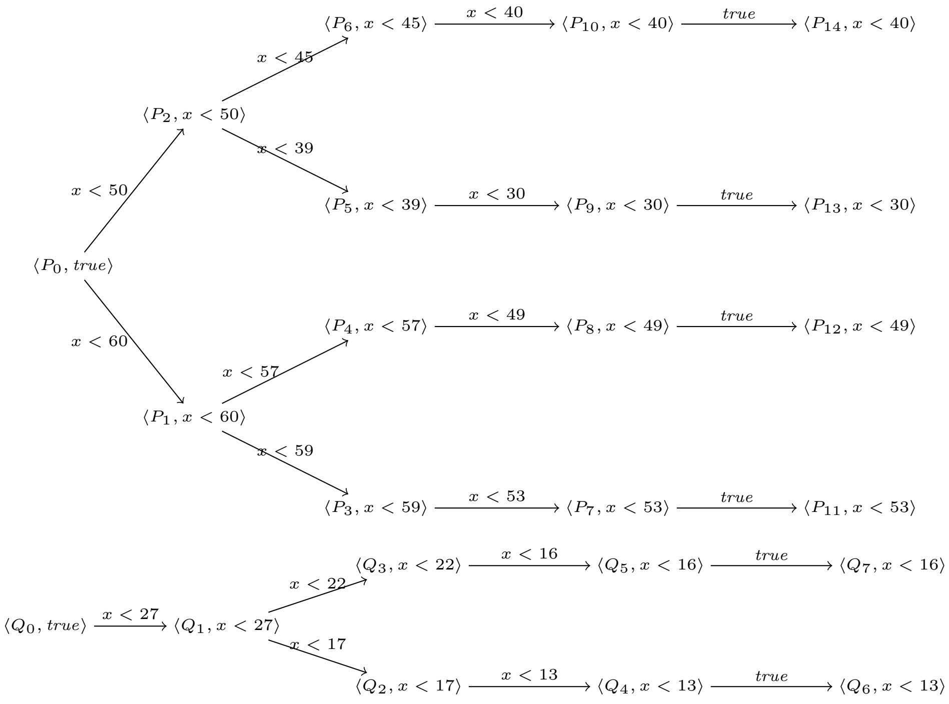

$\langle P_6, x < 45\rangle$
$x < 40$
$\langle P_{10}, x < 40\rangle$
$true$
$\langle P_{14}, x < 40\rangle$
$x < 45$
$\langle P_2, x < 50\rangle$
$x < 39$
$x < 50$
$\langle P_5, x < 39\rangle$
$x < 30$
$\langle P_9, x < 30\rangle$
$true$
$\langle P_{13}, x < 30\rangle$
$\langle P_0, true\rangle$
$\langle P_4, x < 57\rangle$
$x < 49$
$\langle P_8, x < 49\rangle$
$true$
$\langle P_{12}, x < 49\rangle$
$x < 60$
$x < 57$
$\langle P_1, x < 60\rangle$
$x < 59$
$\langle P_3, x < 59\rangle$
$x < 53$
$\langle P_7, x < 53\rangle$
$true$
$\langle P_{11}, x < 53\rangle$
$\langle Q_3, x < 22\rangle$
$x < 16$
$\langle Q_5, x < 16\rangle$
$true$
$\langle Q_7, x < 16\rangle$
$x < 22$
$\langle Q_0, true\rangle$
$x < 27$
$\langle Q_1, x < 27\rangle$
$x < 17$
$\langle Q_2, x < 17\rangle$
$x < 13$
$\langle Q_4, x < 13\rangle$
$true$
$\langle Q_6, x < 13\rangle$

Figure A.9: Example 1 % of Same Labels

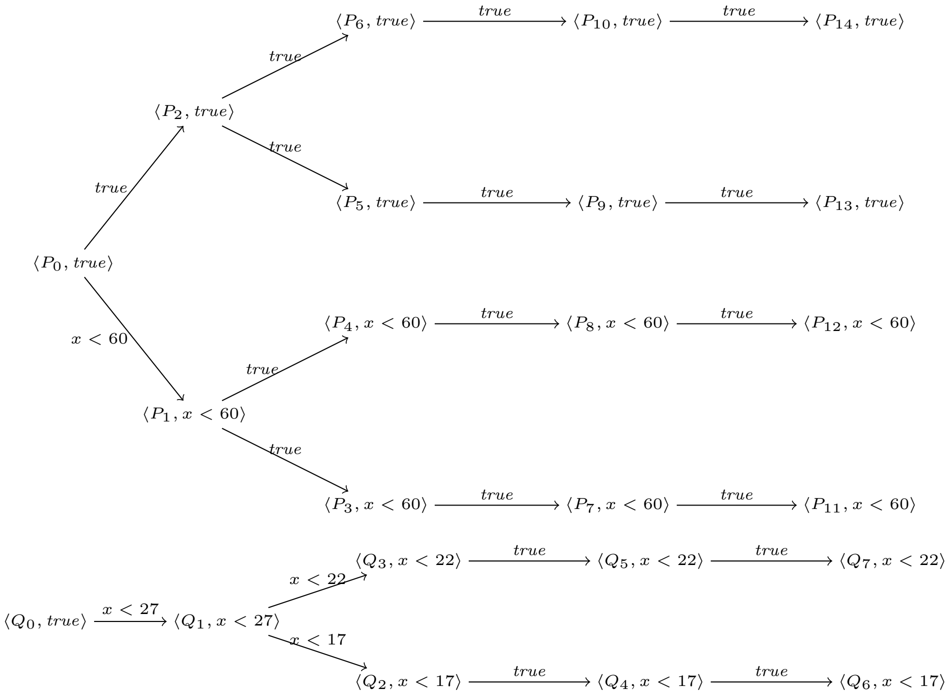


Figure A.10: Example 2 % of Same Labels

Graphs and Analysis

Processes	% of same labels	Time
Process 1	29%	43514 ms
Process 2	33%	37644 ms
Process 3	38%	28849 ms
Process 4	43%	26187 ms
Process 5	48%	24290 ms
Process 6	52%	22527 ms
Process 7	62%	8952 ms
Process 8	71%	3245 ms
Process 9	81%	2423 ms
Process 10	100%	118 ms

Table A.2: Time vs % of Same Labels

Figure A.11 shows the results after introducing the examples and running them using our ccp partition refinement algorithm.

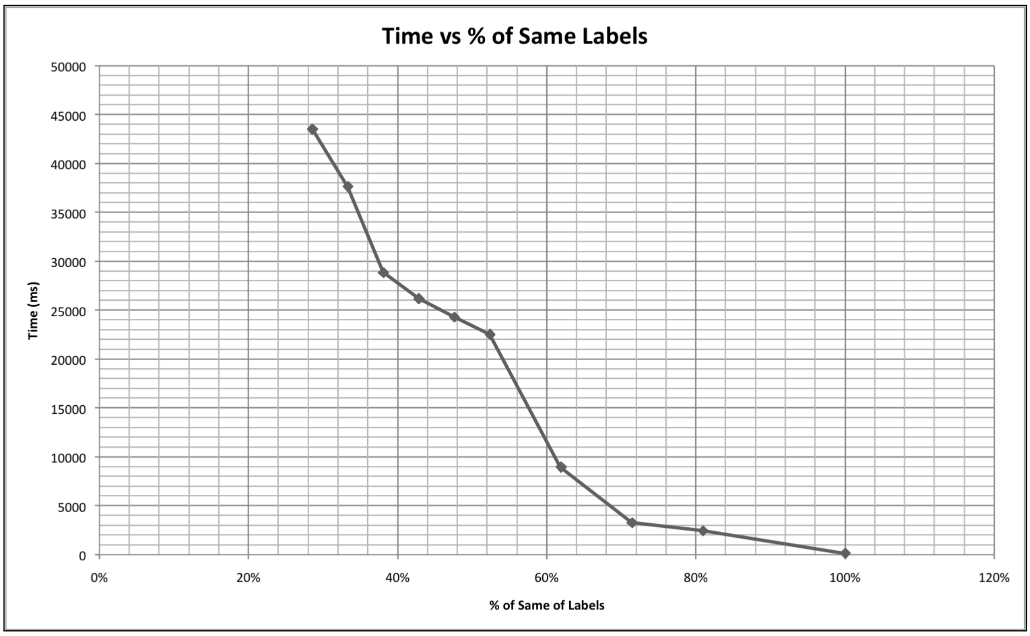


Figure A.11: Time vs Percentage of Same Labels

As expected, the graph shows a decreasing line. This happens since, the highest the percentage of transitions with the same labels the least partitions the algorithm must need in order to refine the blocks. Thus, the time will decrease. Therefore, the execution time is inversely proportional to the percentage of transitions with the same labels.

A.4.3 Percentage of Configurations Satisfying the Same Barb

As in Section A.4.2 we study a property related to the automata generated by the ccp processes. This property is also a percentage, but in this case, the amount of configurations satisfying the same barb. As well as in the aforementioned section, we shall use just only one barb which can be repeated on the automata's configurations, while the other barbs must be different and constant in number of occurrences. In this case the barb we have selected is $x < 55$.

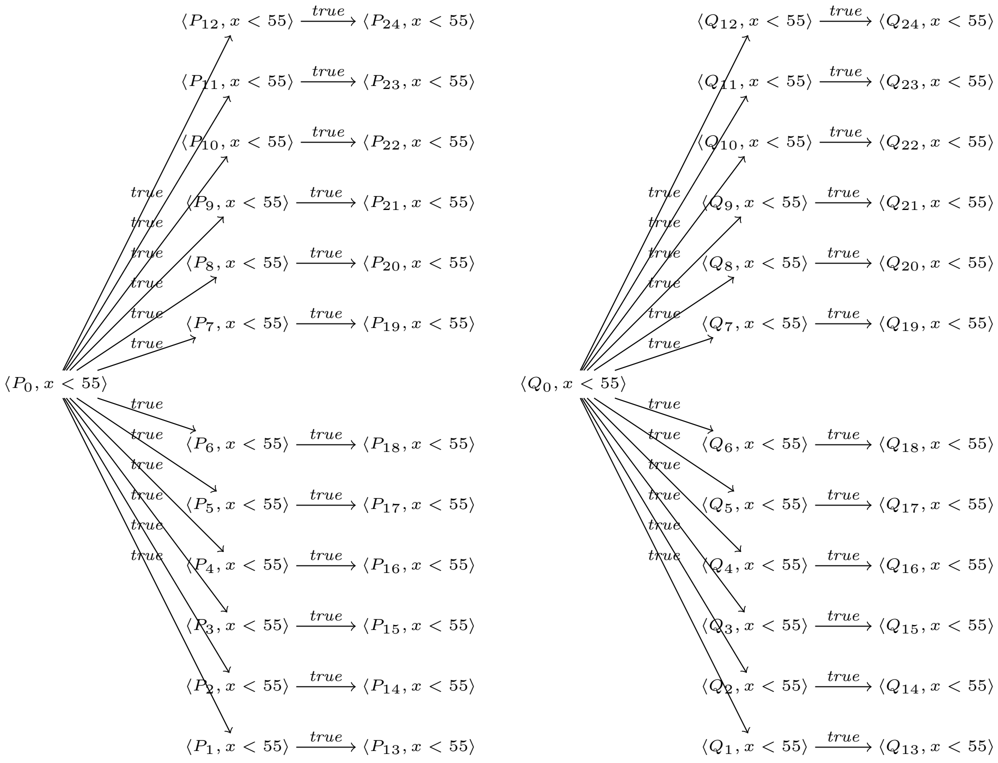

$\langle P_{12}, x < 55\rangle \xrightarrow{true} \langle P_{24}, x < 55\rangle$
$\langle P_{11}, x < 55\rangle \xrightarrow{true} \langle P_{23}, x < 55\rangle$
$\langle P_{10}, x < 55\rangle \xrightarrow{true} \langle P_{22}, x < 55\rangle$
$\langle P_{9}, x < 55\rangle \xrightarrow{true} \langle P_{21}, x < 55\rangle$
$\langle P_{8}, x < 55\rangle \xrightarrow{true} \langle P_{20}, x < 55\rangle$
$\langle P_{7}, x < 55\rangle \xrightarrow{true} \langle P_{19}, x < 55\rangle$
$\langle P_{0}, x < 55\rangle$
$\langle P_{6}, x < 55\rangle \xrightarrow{true} \langle P_{18}, x < 55\rangle$
$\langle P_{5}, x < 55\rangle \xrightarrow{true} \langle P_{17}, x < 55\rangle$
$\langle P_{4}, x < 55\rangle \xrightarrow{true} \langle P_{16}, x < 55\rangle$
$\langle P_{3}, x < 55\rangle \xrightarrow{true} \langle P_{15}, x < 55\rangle$
$\langle P_{2}, x < 55\rangle \xrightarrow{true} \langle P_{14}, x < 55\rangle$
$\langle P_{1}, x < 55\rangle \xrightarrow{true} \langle P_{13}, x < 55\rangle$
true
true
true
true
true
true
true
true
true
true
true
true
$\langle Q_{12}, x < 55\rangle \xrightarrow{true} \langle Q_{24}, x < 55\rangle$
$\langle Q_{11}, x < 55\rangle \xrightarrow{true} \langle Q_{23}, x < 55\rangle$
$\langle Q_{10}, x < 55\rangle \xrightarrow{true} \langle Q_{22}, x < 55\rangle$
$\langle Q_{9}, x < 55\rangle \xrightarrow{true} \langle Q_{21}, x < 55\rangle$
$\langle Q_{8}, x < 55\rangle \xrightarrow{true} \langle Q_{20}, x < 55\rangle$
$\langle Q_{7}, x < 55\rangle \xrightarrow{true} \langle Q_{19}, x < 55\rangle$
$\langle Q_{0}, x < 55\rangle$
$\langle Q_{6}, x < 55\rangle \xrightarrow{true} \langle Q_{18}, x < 55\rangle$
$\langle Q_{5}, x < 55\rangle \xrightarrow{true} \langle Q_{17}, x < 55\rangle$
$\langle Q_{4}, x < 55\rangle \xrightarrow{true} \langle Q_{16}, x < 55\rangle$
$\langle Q_{3}, x < 55\rangle \xrightarrow{true} \langle Q_{15}, x < 55\rangle$
$\langle Q_{2}, x < 55\rangle \xrightarrow{true} \langle Q_{14}, x < 55\rangle$
$\langle Q_{1}, x < 55\rangle \xrightarrow{true} \langle Q_{13}, x < 55\rangle$
true
true
true
true
true
true
true
true
true
true
true
true

Figure A.12: Example 1 % of Configurations Satisfying the Same Barb

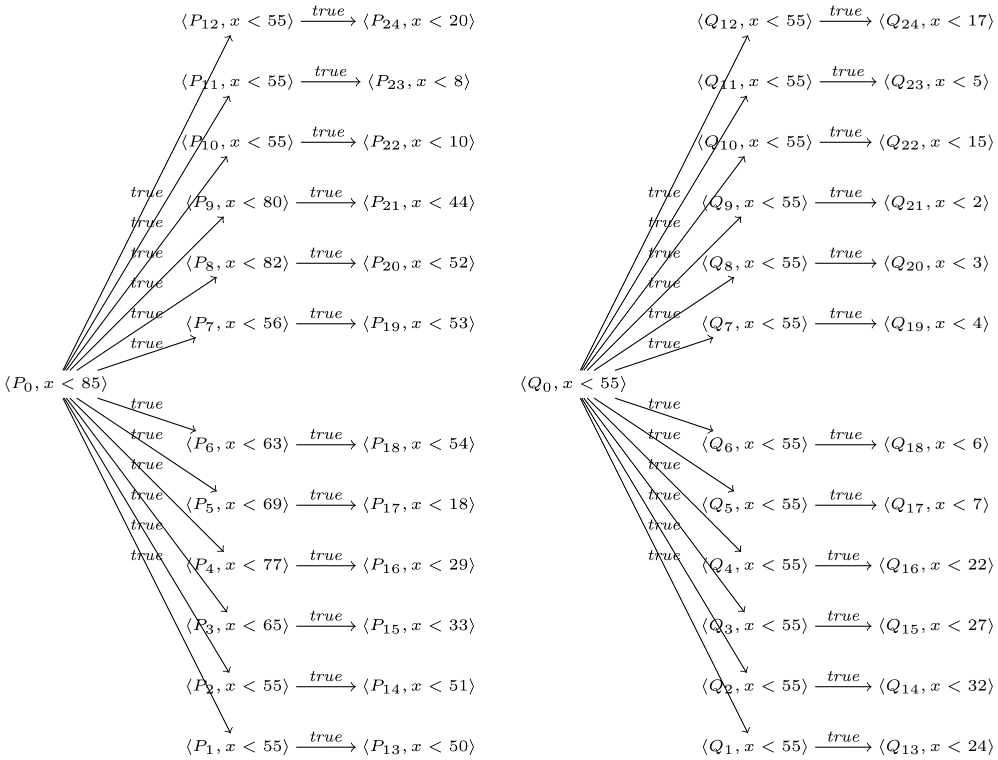

⟨$P_{12}, x < 55$⟩ true ⟨$P_{24}, x < 20$⟩
⟨$P_{11}, x < 55$⟩ true ⟨$P_{23}, x < 8$⟩
⟨$P_{10}, x < 55$⟩ true ⟨$P_{22}, x < 10$⟩
⟨$P_9, x < 80$⟩ true ⟨$P_{21}, x < 44$⟩
⟨$P_8, x < 82$⟩ true ⟨$P_{20}, x < 52$⟩
⟨$P_7, x < 56$⟩ true ⟨$P_{19}, x < 53$⟩
⟨$P_0, x < 85$⟩
⟨$P_6, x < 63$⟩ true ⟨$P_{18}, x < 54$⟩
⟨$P_5, x < 69$⟩ true ⟨$P_{17}, x < 18$⟩
⟨$P_4, x < 77$⟩ true ⟨$P_{16}, x < 29$⟩
⟨$P_3, x < 65$⟩ true ⟨$P_{15}, x < 33$⟩
⟨$P_2, x < 55$⟩ true ⟨$P_{14}, x < 51$⟩
⟨$P_1, x < 55$⟩ true ⟨$P_{13}, x < 50$⟩
⟨$Q_{12}, x < 55$⟩ true ⟨$Q_{24}, x < 17$⟩
⟨$Q_{11}, x < 55$⟩ true ⟨$Q_{23}, x < 5$⟩
⟨$Q_{10}, x < 55$⟩ true ⟨$Q_{22}, x < 15$⟩
⟨$Q_9, x < 55$⟩ true ⟨$Q_{21}, x < 2$⟩
⟨$Q_8, x < 55$⟩ true ⟨$Q_{20}, x < 3$⟩
⟨$Q_7, x < 55$⟩ true ⟨$Q_{19}, x < 4$⟩
⟨$Q_0, x < 55$⟩
⟨$Q_6, x < 55$⟩ true ⟨$Q_{18}, x < 6$⟩
⟨$Q_5, x < 55$⟩ true ⟨$Q_{17}, x < 7$⟩
⟨$Q_4, x < 55$⟩ true ⟨$Q_{16}, x < 22$⟩
⟨$Q_3, x < 55$⟩ true ⟨$Q_{15}, x < 27$⟩
⟨$Q_2, x < 55$⟩ true ⟨$Q_{14}, x < 32$⟩
⟨$Q_1, x < 55$⟩ true ⟨$Q_{13}, x < 24$⟩
true

Figure A.13: Example 2 % of Configurations Satisfying the Same Barb

Graphs and Analysis

Table A.3 presents the data obtained by the automata while studying the influence of the % of configurations satisfying the same barb on the ccp partition refinement algorithm.

Processes	% of configurations satisfying the same barb	Time
Process 1	100%	752 ms
Process 2	96%	1694 ms
Process 3	86%	4084 ms
Process 4	72%	7998 ms
Process 5	64%	10736 ms
Process 6	60%	10743 ms
Process 7	56%	12206 ms
Process 8	52%	12324 ms
Process 9	36%	13184 ms
Process 10	20%	14266 ms

Table A.3: Time vs % of Configurations Satisfying the Same Barb

Now, we switch our attention to Figure A.14. This figure shows the results from Table A.3.

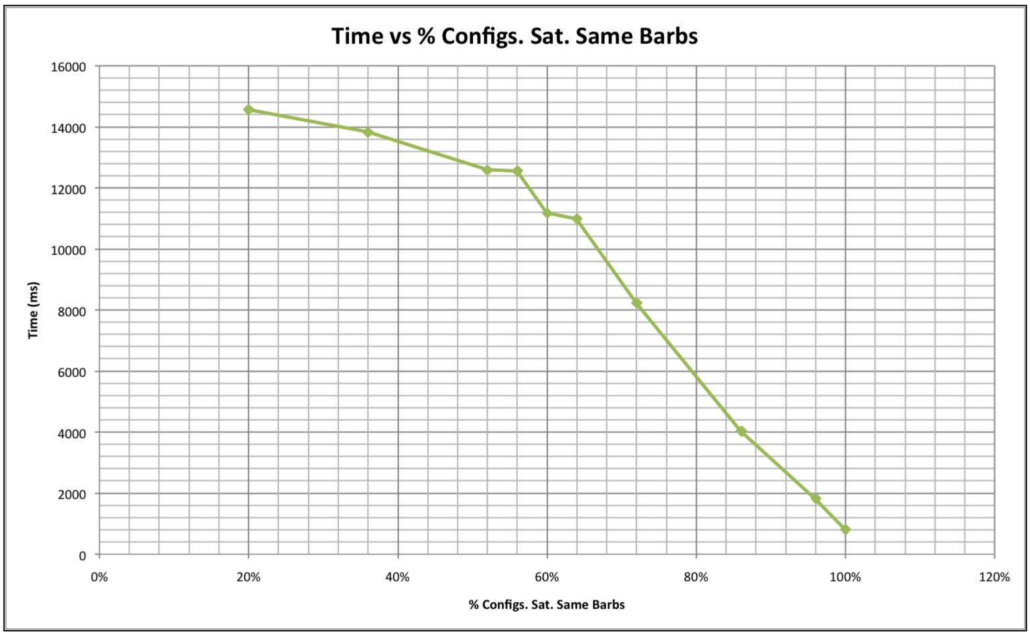


Figure A.14: Time vs Percentage of Configs. Sat. Same Barbs

As in Section A.11 the line shown in the graph is decreasing. This is due to the time the algorithm takes to make the prerefinement, which can be regarded as the first item of the definition of bisimilarity (either strong or weak) (see Definitions 2.4.2 and 2.4.3 in Section 2.4). If the percentage of configurations satisfying the same barb is higher, then the prerefinement will not need to partition many amount of blocks. (See Algorithm A.3.2 for the strong case or Algorithm A.3.3 for the weak notion).

A.4.4 Percentage of Dominated Transitions

In Definition 3.2.3 in Section 3.2.5 we first present the notion of domination. Later on, in that section we show the importance of this dominated transitions or so called possible redundant transitions, in order for the algorithm to calculate the notion of irredundant bisimilarity (see Definition 3.2.7 in Section 3.2.6) which itself depends on the notion of redundant transitions (see Definition 3.2.4 in Section 3.2.5) . This is important for the execution of the algorithm since dealing with redundancy takes a significant part of time in the execution of the algorithm.

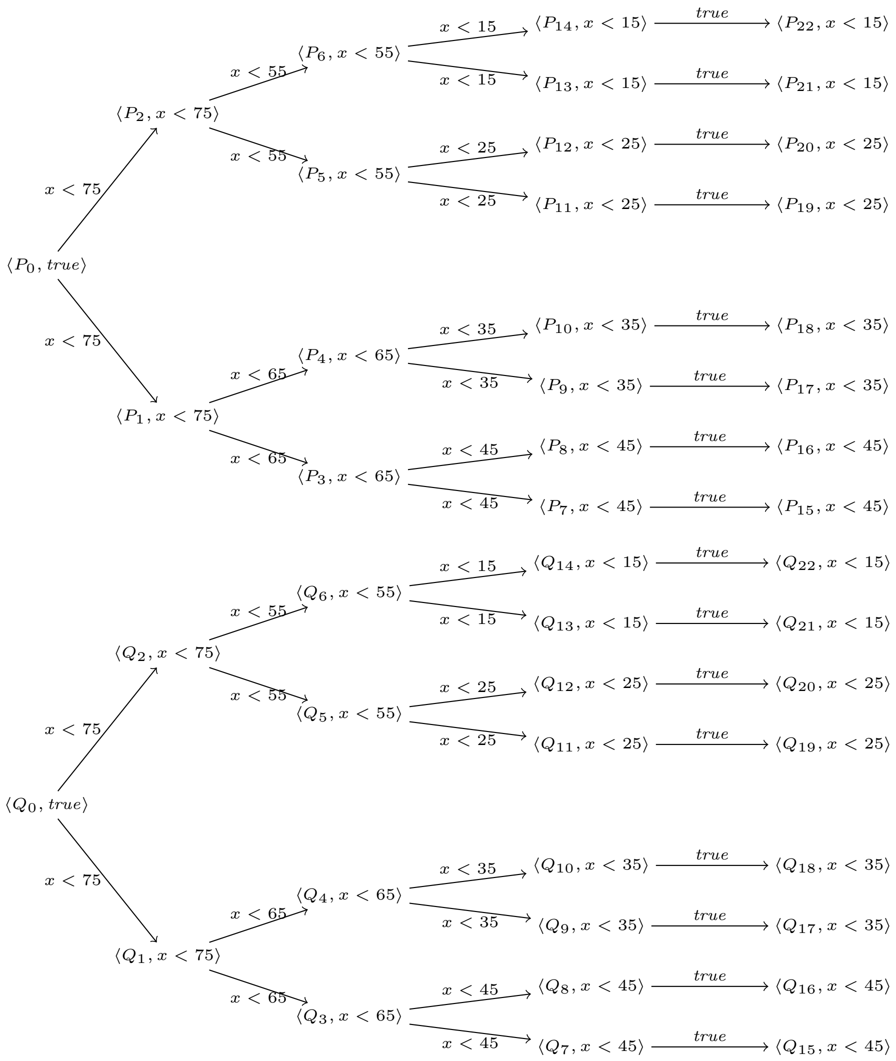


Figure A.15: Example 1 % of Dominated Transitions

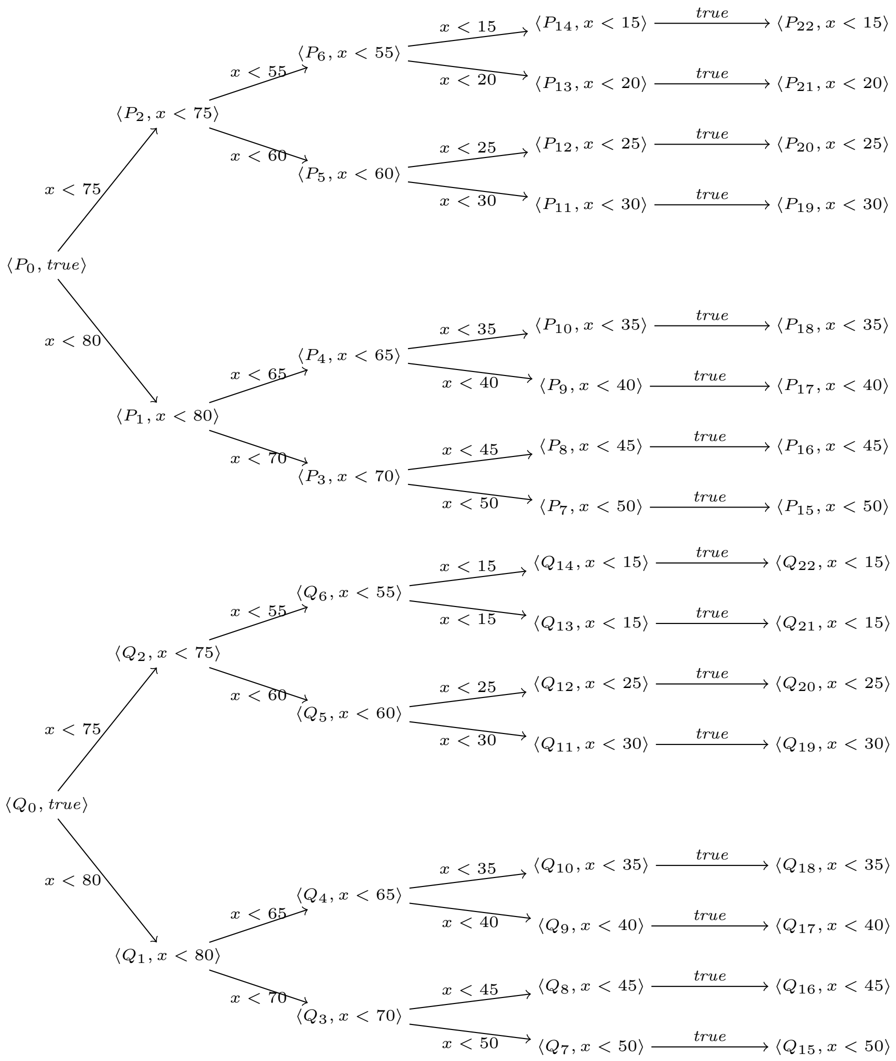

$x < 15$
$\langle P_{14}, x < 15\rangle$
$true$
$\langle P_{22}, x < 15\rangle$
$\langle P_6, x < 55\rangle$
$x < 55$
$x < 20$
$\langle P_{13}, x < 20\rangle$
$true$
$\langle P_{21}, x < 20\rangle$
$\langle P_2, x < 75\rangle$
$x < 60$
$x < 25$
$\langle P_{12}, x < 25\rangle$
$true$
$\langle P_{20}, x < 25\rangle$
$\langle P_5, x < 60\rangle$
$x < 75$
$x < 30$
$\langle P_{11}, x < 30\rangle$
$true$
$\langle P_{19}, x < 30\rangle$
$\langle P_0, true\rangle$
$x < 35$
$\langle P_{10}, x < 35\rangle$
$true$
$\langle P_{18}, x < 35\rangle$
$x < 80$
$\langle P_4, x < 65\rangle$
$x < 65$
$x < 40$
$\langle P_9, x < 40\rangle$
$true$
$\langle P_{17}, x < 40\rangle$
$\langle P_1, x < 80\rangle$
$x < 45$
$\langle P_8, x < 45\rangle$
$true$
$\langle P_{16}, x < 45\rangle$
$x < 70$
$\langle P_3, x < 70\rangle$
$x < 50$
$\langle P_7, x < 50\rangle$
$true$
$\langle P_{15}, x < 50\rangle$
$x < 15$
$\langle Q_{14}, x < 15\rangle$
$true$
$\langle Q_{22}, x < 15\rangle$
$\langle Q_6, x < 55\rangle$
$x < 55$
$x < 15$
$\langle Q_{13}, x < 15\rangle$
$true$
$\langle Q_{21}, x < 15\rangle$
$\langle Q_2, x < 75\rangle$
$x < 25$
$\langle Q_{12}, x < 25\rangle$
$true$
$\langle Q_{20}, x < 25\rangle$
$x < 60$
$\langle Q_5, x < 60\rangle$
$x < 75$
$x < 30$
$\langle Q_{11}, x < 30\rangle$
$true$
$\langle Q_{19}, x < 30\rangle$
$\langle Q_0, true\rangle$
$x < 35$
$\langle Q_{10}, x < 35\rangle$
$true$
$\langle Q_{18}, x < 35\rangle$
$x < 80$
$\langle Q_4, x < 65\rangle$
$x < 65$
$x < 40$
$\langle Q_9, x < 40\rangle$
$true$
$\langle Q_{17}, x < 40\rangle$
$\langle Q_1, x < 80\rangle$
$x < 45$
$\langle Q_8, x < 45\rangle$
$true$
$\langle Q_{16}, x < 45\rangle$
$x < 70$
$\langle Q_3, x < 70\rangle$
$x < 50$
$\langle Q_7, x < 50\rangle$
$true$
$\langle Q_{15}, x < 50\rangle$

Figure A.16: Example 2 % of Dominated Transitions

Graphs and Analysis

We give the data taken by the execution of the previous examples. This informations is used to show the influence of the % of dominated transitions on the execution time of the algorithm. Thus, Table A.4 shows the collected data from the execution of the examples and Figure A.17 depicts the description of the result in the aforementioned table.

Processes	**% of derived Transitions**	**Time**
Process 1	0%	20993 ms
Process 2	5%	36781 ms
Process 3	9%	59620 ms
Process 4	14%	82748 ms
Process 5	18%	130387 ms
Process 6	23%	188745 ms
Process 7	25%	288459 ms
Process 8	27%	296968 ms
Process 9	30%	351186 ms
Process 10	32%	354227 ms

Table A.4: Time vs % of Dominated Transitions

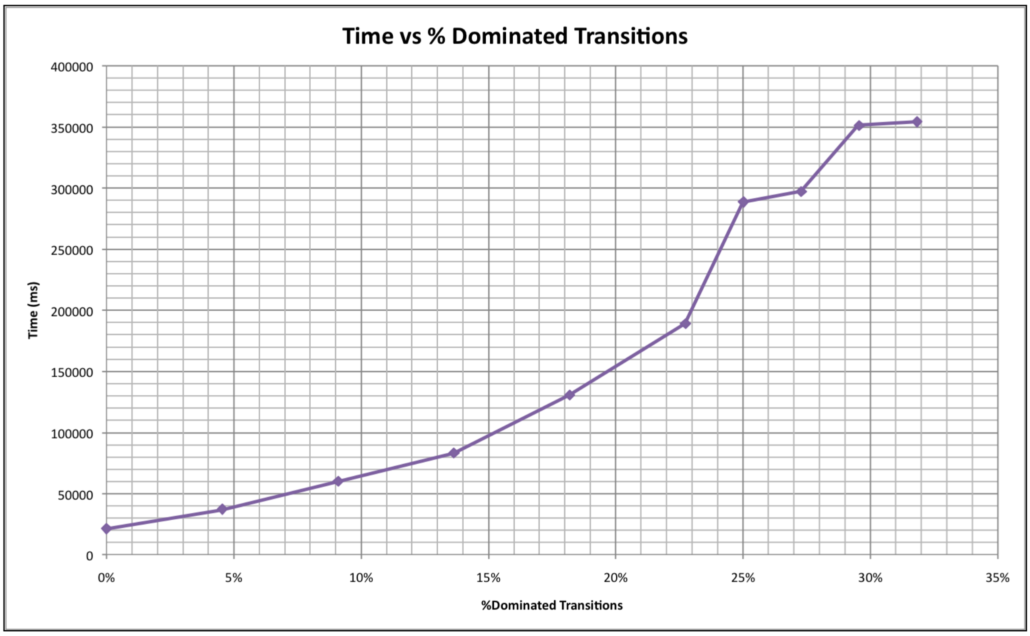


Figure A.17: Time vs Percentage Dominated Transitions

Figure A.17 confirms that checking irredundancy is quite expensive and requires an important amount of time. For this reason, increasing the number of 'possible' redundant transitions might also increase the overall execution time.

A.4.5 Time vs Branches

The following two examples aim at studying the impact of the total number of branches in the execution time of the algorithm. We recall that we refer to branches because the automata we use as an input for our algorithm have always a form of a tree.

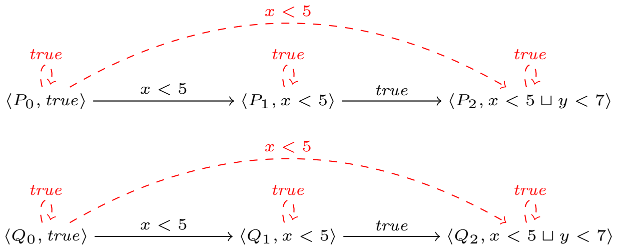

$x < 5$
$true$
$true$
$true$
$\langle P_0, true\rangle$
$x < 5$
$\langle P_1, x < 5\rangle$
$true$
$\langle P_2, x < 5 \sqcup y < 7\rangle$
$x < 5$
$true$
$true$
$true$
$\langle Q_0, true\rangle$
$x < 5$
$\langle Q_1, x < 5\rangle$
$true$
$\langle Q_2, x < 5 \sqcup y < 7\rangle$

Figure A.18: Saturated Example 1 Branches

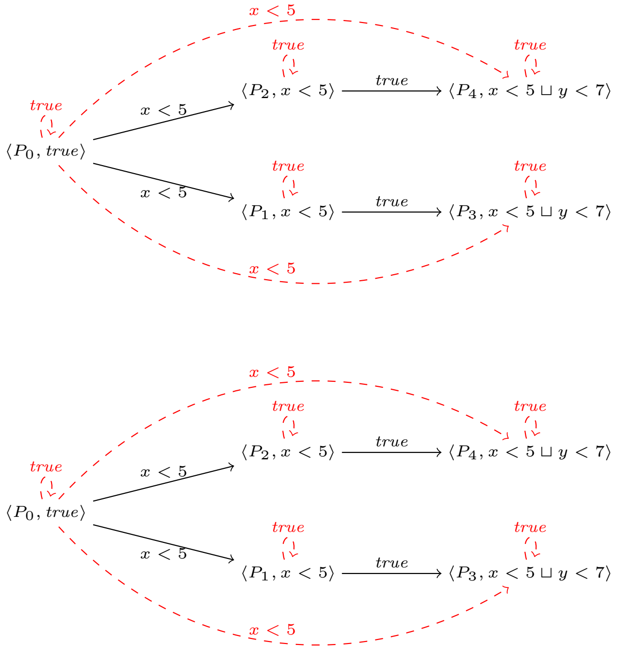

$x < 5$
$true$
$true$
$\langle P_2, x < 5\rangle$
$true$
$\langle P_4, x < 5 \sqcup y < 7\rangle$
$true$
$x < 5$
$\langle P_0, true\rangle$
$x < 5$
$true$
$true$
$\langle P_1, x < 5\rangle$
$true$
$\langle P_3, x < 5 \sqcup y < 7\rangle$
$x < 5$
$x < 5$
$true$
$true$
$\langle P_2, x < 5\rangle$
$true$
$\langle P_4, x < 5 \sqcup y < 7\rangle$
$true$
$x < 5$
$\langle P_0, true\rangle$
$x < 5$
$true$
$true$
$\langle P_1, x < 5\rangle$
$true$
$\langle P_3, x < 5 \sqcup y < 7\rangle$
$x < 5$

Figure A.19: Saturated Example 2 Branches

Graphs and Analysis

Table A.5 shows the results of the execution of the above-mentioned examples. They show the time it takes for the strong and weak algorithm to check their respective notion of bisimilarity. It is important to notice that we compare both execution times (weak and strong) so we can understand how much the amount of branches influence the saturation process, in particular the addition of paths of transitions as a single step transition (see Algorithm A.3.6 and rules R-Label and R-Add in Table 4.2 in Section 4.2.1).

Processes	**Branches**	**Time Strong Case**	**Time Weak Case**
Process 1	2	8 ms	65 ms
Process 2	4	24 ms	295 ms
Process 3	6	69 ms	874 ms
Process 4	8	97 ms	1827 ms
Process 5	10	172 ms	3901 ms

Table A.5: Time vs Branches

Now we intend to interpret the results in a graphical way.

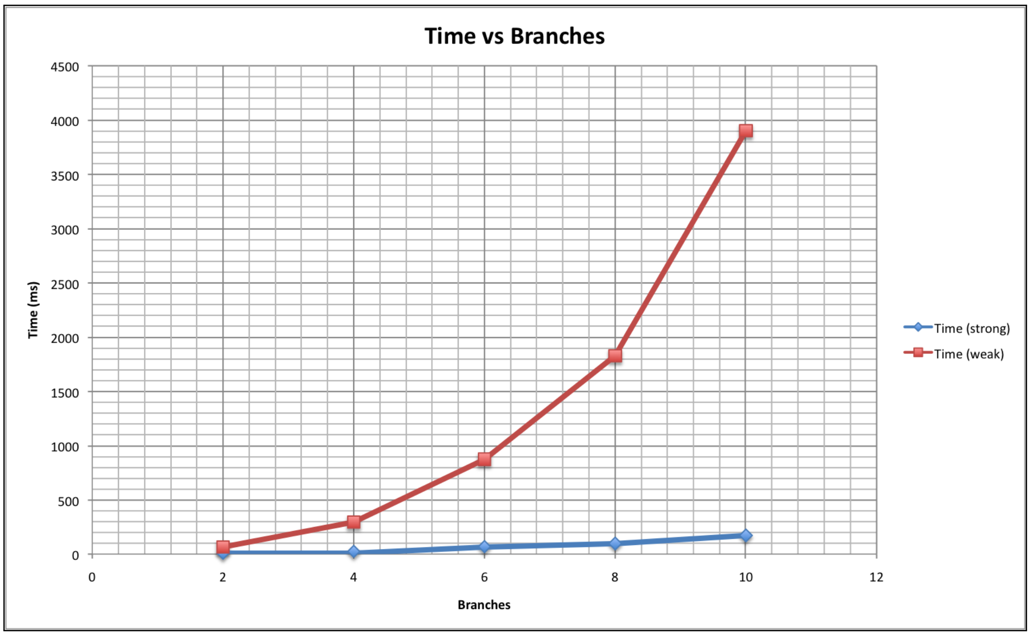


Figure A.20: Time vs Branches

After analyzing Figure A.20, it can be observed that if we have more branches per automata then the execution time of the algorithm with the weak approach increases considerable with respect to the strong one. We can find an interesting and feasible answer by taking into account the notion of saturation, i.e., the method used by us to reduce the weak bisimilarity into the strong notion. Since we have more branches, there is the need of adding new transitions (those which represent a long path of transitions as a single step transition). Recall that we must assume that the branches must have 3 consecutive nodes because if they have less nodes, then the saturation method will not add more transitions at all.

A.5 Summary of Contributions and Related Work

In this Chapter we presented the main aspects of the implementation of the algorithm for verifying (strong and weak) bisimilarity for ccp. We presented the C++ programming language and its object orientation as the main tool for implementing the partition refinement algorithm for ccp. We briefly described the main benefits from this language to be able to implement the partition refinement algorithm for ccp. We gave some examples and analyze by means of graphs how the algorithm behaves according to several properties we have considered central for the execution time of the algorithm. We must stress the fact that even if our worst running case is exponential, the line graphs A.7 and A.8 show us that the execution time of the algorithm has a linear growth with respect to the amount of transitions and nodes. We realized that while using a lesser amount of nodes or transitions the line graph would resemble an exponential line, but while using a much bigger sample of nodes or transitions, in reality, the graph had a linear growth.

We believe that our work represents a very first step to resemble The Concurrency Workbench [CPS93], that is, creating a robust and friendly tool for those who want to automatically check different properties about ccp programs.

As future work, we plan a much more robust implementation of the ccp partition refinement algorithm in order to increase its power, efficiency, performance and scalability. As a short and medium term goal, we intend to include several modifications into the code so the execution time of the algorithm could be decreased. Therefore, one of the most important modifications shall be related with the data structures we must use. Lists are the most used classes to access the data from the constraint system, the labeled transition system and other information needed for the algorithm to work. Therefore we shall take into account their implementation in order to increment the efficiency of the algorithm. Thus, rather than using the lists included by default in C++, we must choose a better implementation or a more efficient class to access the needed data. We shall propose two options, implement a doubly linked list or make use of hash tables. Another alternative for implementing the LTS in a way that the information included in it could be accessed faster, is to represent it as a directed graph implemented as an indexed set of adjacency lists and for each vertex having an inverse adjacency list [JKOK98]. An interesting approach to reduce the complexity of our algorithm is to, instead of modifying the partition refinement algorithm in [KS83], use a better algorithm of the same sort, regarding its complexity, that is, the one in [PT87]. Although we have already said that our worst running case is exponential, we can take ideas of [JKOK98] in order to cope with a growing amount of data. As in the latter cited paper, we can reduce the cost of the algorithm by means of a better implementation, in this case by using a parallel algorithm based on multiway splitting.

As longer term aim we plan to implement a user interface, a parser and a converter so we can create a real tool which could be easily used. We intend to analyze other programming languages that can be more useful than the one we have used, in terms of portability, user friendliness and particularly, execution time. Also we may want to take into account different and more elaborated types of Constraints systems by using libraries such as Gecode [Gec06], choco [cho08], etc.

Bibliography

[ABP$^+$11a] A. Aristizabal, F. Bonchi, C. Palamidessi, L. Pino, and F. Valencia. Deriving labels and bisimilarity for concurrent constraint programming (extended version). Technical report, INRIA-CNRS, 2011. Available at: `http://www.lix.polytechnique.fr/~luis.pino/files/FOSSACS11-extended.pdf`.

[ABP$^+$11b] Andres Aristizabal, Filippo Bonchi, Catuscia Palamidessi, Luis Pino, and Frank D. Valencia. Deriving labels and bisimilarity for concurrent constraint programming. In *FOSSACS*, pages 138–152, 2011.

[ABPV12a] A. Aristizabal, F. Bonchi, L. Pino, and F. Valencia. Partition refinement for bisimilarity in ccp (extended version). Technical report, INRIA-CNRS, 2012. Available at: `http://www.lix.polytechnique.fr/~andresaristi/sac2012.pdf`.

[ABPV12b] Andres Aristizabal, Filippo Bonchi, Luis Pino, and Frank D. Valencia. Partition refinement for bisimilarity in ccp. In *SAC*, pages 88–93, 2012.

[ABPV12c] Andres Aristizabal, Filippo Bonchi, Luis Pino, and Frank D. Valencia. Reducing weak to strong bisimilarity in ccp. In *ICE*, EPTCS, 2012.

[ACS96] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous pi-calculus. In *Proc. of CONCUR*, volume 1119 of *LNCS*, pages 147–162. Springer, 1996.

[AJ94] Samson Abramsky and Achim Jung. Domain theory. In *Handbook of Logic in Computer Science*, pages 1–168. Clarendon Press, 1994.

[Ari10a] Andres Aristizabal. Bisimilarity in concurrent constraint programming. In *ICLP (Technical Communications)*, pages 236–240, 2010.

[Ari10b] Andres Aristizabal. Bisimilarity in concurrent constraint programming. In *2nd Young Researchers Workshop on Concurrency Theory (YR-CONCUR 2010)*, 2010.

[BGM09] Filippo Bonchi, Fabio Gadducci, and Giacoma Valentina Monreale. Reactive systems, barbed semantics, and the mobile ambients. In *FOSSACS*, pages 272–287, 2009.

[BJPV09] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, pages 39–48, 2009.

[BKM06] Filippo Bonchi, Barbara König, and Ugo Montanari. Saturated semantics for reactive systems. In *LICS*, pages 69–80, 2006.

[BM08] Maria Grazia Buscemi and Ugo Montanari. Open bisimulation for the concurrent constraint pi-calculus. In *ESOP*, pages 254–268, 2008.

[BM09] Filippo Bonchi and Ugo Montanari. Minimization algorithm for symbolic bisimilarity. In *ESOP*, pages 267–284, 2009.

[BMM11] Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. A connector algebra for p/t nets interactions. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2011.

[BZ10] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *LICS*, pages 332–341. IEEE Computer Society, 2010.

[cho08] choco Team. choco: an open source java constraint programming library, 2008. `http://choco.emn.fr`.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.

[dBPP95] Frank S. de Boer, Alessandra Di Pierro, and Catuscia Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theor. Comput. Sci.*, 151(1):37–78, 1995.

[EK04] Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the dpo approach to graph rewriting. In *FoSSaCS*, pages 151–166, 2004.

[Fer89] Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Sci. Comput. Program.*, 13(1):219–236, 1989.

[FGM$^+$98] G.L. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. Verifying mobile processes in the hal environment. In *CAV*, pages 511–515, 1998.

[FGMP94] Moreno Falaschi, Maurizio Gabbrielli, Kim Marriott, and Catuscia Palamidessi. Confluence and concurrent constraint programming. In *GULP-PRODE (1)*, pages 140–154, 1994.

[FGMP97] Moreno Falaschi, Maurizio Gabbrielli, Kim Marriott, and Catuscia Palamidessi. Confluence in concurrent constraint programming. *Theor. Comput. Sci.*, 183(2):281–315, 1997.

[Gec06] Gecode Team. Gecode: Generic constraint development environment, 2006. `http://www.gecode.org`.

[GHL08] Pietro Di Gianantonio, Furio Honsell, and Marina Lenisa. Rpo, second-order contexts, and lambda-calculus. In *FoSSaCS*, pages 334–349, 2008.

[GM00] Fabio Gadducci and Ugo Montanari. The tile model. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*, pages 133–166. The MIT Press, 2000.

[HT71] J.D. Monk L. Henkin and A. Tarski. *Cylindric Algebras (Part I)*. North-Holland, 1971.

[HY95] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theor. Comput. Sci.*, 151(2):437–486, 1995.

[JBPV10] Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *LICS*, pages 322–331, 2010.

[Jen06] O. H. Jensen. *Mobile Processes in Bigraphs*. PhD thesis, University of Cambridge, 2006.

[JKOK98] Cheoljoo Jeong, Youngchan Kim, Youngbae Oh, and Heungnam Kim. A faster parallel implementation of the kanellakis-smolka algorithm for bisimilarity checking. In *In Proceedings of the International Computer Symposium*, 1998.

[JR97] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:62–222, 1997.

[JVP10] Magnus Johansson, Björn Victor, and Joachim Parrow. A fully abstract symbolic semantics for psi-calculi. *CoRR*, abs/1002.2867, 2010.

[KPPV12] Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, and Frank D. Valencia. Spatial information distribution in constraint-based process calculi (extended version). Technical report, INRIA, 2012. Available at: `http://www.lix.polytechnique.fr/~fvalenci/papers/eccp-extended.pdf`.

[KS83] Paris C. Kanellakis and Scott A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *PODC*, pages 228–240, 1983.

[LM00] James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *CONCUR*, pages 243–258, 2000.

[LPSS11] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag New York, Inc., 1980.

[Mil99] Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[MPLS] Ugo Montanari, Marco Pistore, Insup Lee, and Scott Smolka. *Checking bisimilarity for finitary π-calculus*, pages 42–56. Springer Berlin / Heidelberg.

[MPSS95] N. P. Mendler, Prakash Panangaden, Philip J. Scott, and R. A. G. Seely. A logical view of concurrent constraint programming. *Nord. J. Comput.*, 2(2):181–220, 1995.

[MS92a] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *ICALP*, pages 685–695, 1992.

[MS92b] Ugo Montanari and Vladimiro Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *FI*, 16(1):171–199, 1992.

[OV08] Carlos Olarte and Frank D. Valencia. Universal concurrent constraint programing: symbolic semantics and applications to security. In *SAC*, pages 145–150, 2008.

[PS96] Marco Pistore and Davide Sangiorgi. A partition refinement algorithm for the *pi*-calculus (extended abstract). In *CAV*, pages 38–49, 1996.

[PSVV06] Catuscia Palamidessi, Vijay A. Saraswat, Frank D. Valencia, and Björn Victor. On the expressiveness of linearity vs persistence in the asychronous pi-calculus. In *LICS*, pages 59–68, 2006.

[PT87] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[PV01] Catuscia Palamidessi and Frank D. Valencia. A temporal concurrent constraint programming calculus. In *CP*, pages 302–316, 2001.

[RS08] Julian Rathke and Pawel Sobocinski. Deconstructing behavioural theories of mobility. In *IFIP TCS*, pages 507–520, 2008.

[RSS07] Julian Rathke, Vladimiro Sassone, and Pawel Sobociński. Semantic barbs and biorthogonality. In *Proceedings of FoSSaCS'07*, volume 4423 of *LNCS*, pages 302–316. Springer, 2007.

[San09] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4), 2009.

[San11] Davide Sangiorgi. *An introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.

[Sew98] Peter Sewell. From rewrite to bisimulation congruences. In *Proc. of CONCUR '98*, volume 1466 of *LNCS*, pages 269–284. Springer, 1998.

[SJG] Vijay Saraswat, Radha Jagadeesan, and Vineet Gupta. Foundations of timed concurrent constraint programming.

[Sob10] Pawel Sobocinski. Representations of petri net interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 554–568. Springer, 2010.

[Sob12] Pawel Sobocinski. Relational presheaves as labelled transition systems. In *In proceedings CMCSâ 2012*, To appear in Lecture Notes in Computer Science, 2012.

[SR90] Vijay A. Saraswat and Martin C. Rinard. Concurrent constraint programming. In *POPL*, pages 232–245, 1990.

[SR12] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

[SRP91] Vijay A. Saraswat, Martin C. Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352, 1991.

[SS05] Vladimiro Sassone and Pawel Sobocinski. Reactive systems over cospans. In *LICS*, pages 311–320, 2005.

[V.A89] V.A.Saraswat. *Concurrent Constraint Programming*. PhD thesis, Carnegie-Mellon University, 1989.

[VM94] Björn Victor and Faron Moller. The mobility workbench - a tool for the pi-calculus. In *CAV*, pages 428–440, 1994.

[Wil11] John W. Wilkinson. Json spirit: A c++ json parser/generator implemented with boost spirit, 15 Sep 2011. Available at: `http://www.codeproject.com/Articles/20027/JSON-Spirit-A-C-JSON-Parser-Generator-Implemented`.

Index