On the Expressivity of Infinite and Local Behaviour in Fragments of the pi-calculus

Jesus Aranda

ÉCOLE POLYTECHNIQUE DE PARIS and **UNIVERSIDAD DEL VALLE COLOMBIA**

Laboratoire d'Informatique (LIX) and
Esc. de Ing. de Sistemas y Computación (EISC)

P H D T H E S I S

to obtain the title of

PhD of Science

of the École Polytechnique and La Universidad del Valle

Specialty: COMPUTER SCIENCE

Defended by

Jesús Alexander ARANDA BUENO

On the Expressivity of Infinite and Local Behaviour in Fragments of the π-calculus

Thesis Advisors: Catuscia PALAMIDESSI (LIX) and
Frank D. VALENCIA (LIX) and
Juan Francisco DÍAZ (EISC)

prepared at LIX, COMETE Team

Abstract

The π-calculus [61] is one the most influential formalisms for modelling and analyzing the behaviour of concurrent systems. This calculus provides a language in which the structure of terms represents the structure of processes together with an operational semantics to represent computational steps. For example, the parallel composition term $P \mid Q$, which is built from the terms P and Q, represents the process that results from the parallel execution of the processes P and Q. Similarly, the *restriction* $(\nu x)P$ represents a process P with local resource x. The *replication* $!P$ can be thought of as abbreviating the parallel composition $P \mid P \mid \ldots$ of an unbounded number of P processes.

As for other language-based formalisms (e.g., logic, formal grammars and the λ-calculus) a fundamental part of the research in process calculi involves the study of the *expressiveness* of fragments or variants of a given process calculus. In this dissertation we shall study the expressiveness of some variants of the π-calculus focusing on the role of the terms used to represent local and infinite behaviour, namely restriction and replication.

The first part of this dissertation is devoted to the expressiveness of the zero-adic variant of the (polyadic) π-calculus, i.e., CCS with replication ($\text{CCS}_!$) [21].

Busi et al [22] show that $\text{CCS}_!$ is Turing powerful [22]. The result is obtained by encoding Random Access Machines (RAMs) in $\text{CCS}_!$. The encoding is said to be *non-faithful* because it may move from a state which can lead to termination into a divergent one which do not correspond to any configuration of the encoded RAM. I.e., the encoding is not termination preserving.

In this dissertation we shall study the existence of faithful encodings into $\text{CCS}_!$ of models of computability strictly less expressive than Turing Machines. Namely, grammars of Types 1 (Context Sensitive Languages), 2 (Context Free Languages) and 3 (Regular Languages) in the Chomsky Hierarchy. We provide faithful encodings of Type 3 grammars. We show that it is impossible to provide a faithful encoding of Type 2 grammars and that termination-preserving $\text{CCS}_!$ processes can generate languages which are not Type 2. We finally conjecture that the languages generated by termination-preserving $\text{CCS}_!$ processes are Type 1 .

We also observe that the encoding of RAMs [22] and several encoding of

Turing-powerful formalisms in π-calculus variants may generate an unbounded number of restrictions during the simulation of a given machine. This unboundedness arises from having restrictions under the scope of replication (or recursion) as in e.g., $!(\nu x)P$ or $\mu X.(\nu x)(P \mid X)$. This suggests that such an interplay between these operators is fundamental for Turing completeness.

We shall also study the expressive power of restriction and its interplay with replication. We do this by considering several syntactic variants of $\text{CCS}_!$ which differ from each other in the use of restriction with respect to replication. We consider three syntactic variations of $\text{CCS}_!$ which do not allow the generation of unbounded number of restrictions: $\text{CCS}_!^{-!\nu}$ is the fragment of $\text{CCS}_!$ not allowing restrictions under the scope of a replication, $\text{CCS}_!^{-\nu}$ is the restriction-free fragment of $\text{CCS}_!$. The third variant is $\text{CCS}_{!+pr}^{-!\nu}$ which extends $\text{CCS}_!^{-!\nu}$ with Phillips' priority guards [76].

We shall show that the use of an unboundedly many restrictions in $\text{CCS}_!$ is necessary for obtaining Turing expressiveness in the sense of Busi et al [22]. We do this by showing that there is no encoding of RAMs into $\text{CCS}_!^{-!\nu}$ which preserves and reflects convergence. We also prove that up to failures equivalence, there is no encoding from $\text{CCS}_!$ into $\text{CCS}_!^{-!\nu}$ nor from $\text{CCS}_!^{-!\nu}$ into $\text{CCS}_!^{-\nu}$. Thus up to failures equivalence, we cannot encode a process with an unbounded number of restrictions into one with a bounded number of restrictions, nor one with a bounded number of restrictions into a restriction-free process.

As lemmata for the above results we prove that convergence is decidable for $\text{CCS}_!^{-!\nu}$ and that language equivalence is decidable for $\text{CCS}_!^{-\nu}$ but undecidable for $\text{CCS}_!^{-!\nu}$. As corollary it follows that convergence is decidable for restriction-free CCS. Finally, we show the expressive power of priorities by providing a faithful encoding of RAMs in $\text{CCS}_{!+pr}^{-!\nu}$, thus bearing witness to the expressive power of Phillips' priority guards [76].

The second part of this dissertation is devoted to expressiveness of the asynchronous monadic π-calculus, Aπ [15, 47]. In [70] the authors studied the expressiveness of persistence in Aπ [15, 47] wrt weak barbed congruence. The study is incomplete because it ignores divergence.

We shall present an expressiveness study of persistence in Aπ wrt De Nicola and Hennessy's testing scenario which is sensitive to divergence. Following [70],

we consider Aπ and three sub-languages of it, each capturing one source of persistence: the persistent-input Aπ-calculus (PIAπ), the persistent-output Aπ-calculus (POAπ) and the persistent Aπ-calculus (PAπ). In [70] the authors showed encodings from Aπ into the semi-persistent calculi (i.e., POAπ and PIAπ) correct wrt weak barbed congruence. We show that, under some general conditions related to compositionality of the encoding and preservation of the infinite behaviour, there cannot be an encoding from Aπ into a (semi)-persistent calculus preserving the must testing semantics. We also prove that convergence and divergence are decidable for POAπ (and PAπ). As a consequence there is no encoding preserving and reflecting divergence or convergence from Aπ into POAπ (and PAπ). This study fills a gap on the expressiveness study of persistence in Aπ in [70].

Contents

Chapter 1

Introduction

The *(polyadic) π-calculus* [61] is one of the most influential formalisms for modeling and analyzing the behavior of concurrent systems; i.e. systems consisting of multiple computing agents, called *processes*, that interact with one another. Indeed, the π-calculus has attained a wide range of applications in different areas of computer science and engineering, among others: Biology [32, 74, 75], business processes [80, 86, 89], object-oriented programming [50, 77, 78], security [1, 2, 36], session types [37, 40, 90], and service oriented computing [54, 57, 89].

One could argue that the development of the π-calculus is reminiscent of those from other standard linguistic formalisms from computability such as logic, formal languages, and most notably the λ-calculus. In fact, the treatment of processes in the π-calculus is akin to that of computable functions in the λ-calculus: The π-calculus provides a language in which the structure of terms represents the structure of processes and an operational semantics representing system evolution. Another similarity with the λ-calculus is that the design of the π-calculus seems to pay special attention to economy: There are few operators, each one denoting a fundamental and primitive role, that can be combined to describe complex concurrent behaviors.

For example, the *parallel composition* term $P \mid Q$, which is built from the terms P and Q, represents the process that results from the parallel execution of the processes P and Q. If P and Q have the form of an *input* (receiver) $x(\vec{y}).P'$ and *output* (sender) $\bar{x}\vec{z}.Q'$, resp., and they have the same arity (i.e., $|\vec{y}| = |\vec{x}|$) then

$P \mid Q$ can be rewritten as $P'[\vec{z}/\vec{y}] \mid Q'$ where $[\vec{z}/\vec{y}]$ denotes the substitution of (the *names* in) the vector $\vec{z}$ for (the placeholders in) the vector $\vec{y}$. This rewritten term can be thought of as resulting from the synchronization on a channel x and the trasmittion along x of the names $\vec{z}$ sent from Q to P. One of the central terms to this dissertation is the *restriction* $(\nu x)P$ which represents a process P with a local *resource* (name) x. The other is the *replication* $!P$ which can be viewed as $P \mid P \mid \ldots$, i.e. an unbounded number of copies of P in parallel. These terms can be combined to represent interesting behaviours, e.g., an unbounded number of local resources $!(\nu x)P$.

Expressiveness

A fundamental part of the research in computability, and linguistic formalisms, in particular, has involved the *expressiveness* of syntactic variants. This includes questions such as whether a given fragment of a logic is as hard for validity as the full language, or whether a given grammar can generate a certain set. Well-known results include the fact that it is not possible to provide a computable transformation, or *encoding*, of arbitrary formulae into monadic ones that preserves validity and that the set $a^n b^n$ cannot be generated by any regular grammar. Standard taxonomies include the Chomsky Hierarchy of formal languages and the logic classification of prenex normal forms [14].

Unfortunately, the subject of expressiveness in the π-calculus, and process calculi at large, is not a well-established discipline, or even a stable craft. Several guiding principles and cogent classification criteria have been put forth in several works such as [69, 43, 64, 92, 27, 45, 53, 16, 72, 91]. Hitherto, however, we do not have a general agreement as to what are the properties that a taxonomy of process calculi must consider in the way we have for the linguistic formalisms of computability where the notion of language (generation) can be taken as the canonical measure for expressiveness. This is perhaps due to the great diversity of observations and properties often used to reason about concurrent behaviour (e.g., divergence, convergence, failures, traces, barbs, must testing, bisimilarity, etc). It may be the case that rather than being absolute, a taxonomy of concurrent calculi ought to be parametric on the observations we wish to make of processes.

After all concurrency is a field with a myriad of aspects for which we may require different terms of discussion and analysis.

Nevertheless, expressiveness studies may offer crucial insights about the limitations, redundancies and capabilities of a family of process calculi. Most works on the expressiveness of the π-calculus consider questions such as whether a given variant can express certain behaviours, whether a given variant is as expressive as another one w.r.t. certain equivalence, or whether a given fragment is as hard for certain property as the full language.

For example, there are certain context-free sets that cannot be represented in a restriction-free variant of the zero-adic π-calculus [28]. (By n-adic we refer to the maximal arity of the vectors of names that can be transmitted upon synchronization.) We also know that every polyadic π term can be encoded into a monadic π term preserving bisimilarity (a standard equivalence in concurrency theory) [83] and that under certain reasonable requirements one cannot encode every monadic π-calculus term into a zero-adic one [69]. These expressiveness questions are of great interest as a variant may simplify the presentation of the calculus, be tailored to specific applications, or be used to single out important aspects of the calculus.

This dissertation is devoted to the study of the expressiveness of several variants of the π-calculus. The main variants under consideration are the zero-adic π-calculus, also known as $\text{CCS}_!$ [21, 22], and the asynchronous monadic π-calculus Aπ [15, 47]. We shall mainly focus on behaviours arising from the *restriction* and *replication* operators as well as from their interplay. The study will be conducted by imposing natural constraints on these operators and their interaction, and then showing their impact on the expressiveness of the constrained variant.

Our study is inspired in part by work and elements from linguistic formalisms such as logic and formal grammars. This is evidenced by the kind of results that we shall discuss in detail later on in this introduction. Namely, we shall give a classification of zero-adic π-calculi following the *Chomsky Hierarchy* of formal grammars as well as a classification of zero-adic π-calculus processes that resembles the classification of prenex first-order formulae w.r.t. constraints on quantifiers [14]. We shall also give a classification of semi-linear (semi-persistent) π-calculus which is inspired by the notion of resource in Girard's linear logic [39].

Our work builds on the seminal paper by Busi, Gabbrielli and Zavattaro on

CCS$_!$ [22] as well as the work by Palamidessi, Saraswat, Victor and Valencia on the asynchronous π-calculus [70]. In fact, this dissertation has been structured in two main parts reflecting the influence of these works. These parts are motivated and discussed next.

Part I: The Expressiveness of CCS$_!$

In [22] Busi et al demostrated that CCS$_!$ is Turing powerful by encoding Random Access Machines (RAMs). The key property of the encoding is that it preserves and reflects *convergence* (i.e., the existence of halting computations): Given a RAM M, the encoding of M in CCS$_!$ converges if and only if M converges.

We wish to outline two observations we made about the encoding given in [22] that are central to this part: (1) The mechanism used to force *"unfaithful"* computational paths of the encoding to be infinite and (2) the mechanism used to generate an unbounded number of restricted names. Let us ellaborate on these two points:

Chapter 3: Computational Expressiveness of CCS$_!$ The first observation is that the CCS$_!$ encoding of RAMs uses a *divergence mechanism* to force the unfaithful computational paths to be infinite. By unfaithful path we mean, informally, paths that do not correspond to those of the encoded machine. The CCS$_!$ encoding of a given RAM can, during evolution, move from a state that may terminate, i.e. a *(weakly) terminating state*, into one that cannot terminate, i.e., a *(strongly) non-terminating state*. Consequently, the encoding does not preserve (weak) termination during evolution. This allows us to ignore the unfaithful behaviour as follows: Whenever the encoding takes a computational path that makes a wrong guess about a test for zero (i.e., it does not correspond to the test for zero of the encoded RAM) then that path is forced to be infinite. This infinite path is thus regarded as a *non-halting* computation and therefore ignored. All finite computations of the encoding, however, correspond to those halting computations of the encoded RAM. Hence, the encoding preserves and reflects convergence.

One may wonder if we can dispense with the above mechanism and still be able to provide an encoding of RAMs that preserves and reflects convergence.

Busi et al has answered negatively this question in [21]. Another legitimate question is thus:

Q1: *What less expressive computational models can be encoded into CCS$_!$ without using this divergence mechanism, i.e. with weak-termination preserving processes ?*

We shall partially answer this question in Chapter 3. We study the family of weak-termination preserving processes by considering models of computability strictly less expressive that RAM's. In particular we shall study the expressiveness of CCS! w.r.t. the existence of termination-preserving encodings of grammars of Types 1 (Context Sensitive grammars), 2 (Context Free grammars) and 3 (Regular grammars) in the Chomsky Hierarchy whose expressiveness corresponds to (non-deterministic) Linear-bounded, Pushdown and Finite-State Automata, respectively. We shall show that they can encode any Type 3 grammars but not Type 2 grammars. We also conjecture that the set of all finite sequences performed by a weak-termination preserving process corresponds to Type 1 grammars.

Chapter 4: The Expressiveness of Restriction and Replication. The second observation is that the CCS$_!$ encoding of RAMs [22] uses restriction operators *under the scope* of replication as for example in $!(\nu x)P$. Indeed, to the best of our knowledge, all the encodings of Turing-powerful formalisms in π-calculus variants such as CCS, the monadic π-calculus, the asynchronous π-calculus involve restrictions under the scope of replication or under the scope recursive expressions as for example in $\mu X.(\nu x)(P \mid X)$.

As mentioned earlier in this introduction, a restriction under replication represents (the potential generation of) unboundedly many local processes in parallel: $!(\nu x)P$ represents infinitely many *restricted declarations* of x. This mechanism seems crucial for the encoding of Turing-powerful formalisms mentioned above. We then find it natural to ask:

Q2: *Is the occurrence of restriction under the scope of replication necessary for the Turing-completeness of some π-based calculi ?*

At this point it is perhaps worth mentioning that a somewhat similar expressiveness situation, from which we partially took inspiration, arises in logic. We can think of a formula $\forall y \exists x F(y, x)$ as describing potentially infinitely many existential declarations of x, one for each each possible y. There is a complete study of decidable classes (w.r.t. satisfiability) of formulae involving the occurrence of existential quantifiers *under the scope* of universal quantification. For example, Skolem showed that the class of formulae of the form $\forall y_1 ... \forall y_n \exists z_1 ... \exists z_m F$, where F is quantifier-free formula, is undecidable while from Gödel we know that its subclass $\forall y_1 \forall y_2 \exists z_1 ... \exists z_m F$ is decidable [14].

Perhaps a closer analogy arises in temporal logic [56]: The formula $\Box \exists x F(x)$, whose intended meaning is "always there exists x such that $F(x)$" can be viewed as describing an unbounded number of existential declarations of x over time. This scoping of the "always" modality over existential quantification is central to the proof that monadic temporal logic is undecidable w.r.t. validity and hence cannot be encoded in propositional temporal logic [68].

In Chapter 4 we study the expressiveness of restriction and its interplay with replication. We consider two syntactic variations of $\text{CCS}_!$ that do not allow the use of an unbounded number of restrictions: $\text{CCS}_!^{-!\nu}$ is the fragment of $\text{CCS}_!$ not allowing restrictions under the scope of a replication. $\text{CCS}_!^{-\nu}$ is the restriction-free fragment of $\text{CCS}_!$.

We shall show that having restriction under replication in $\text{CCS}_!$ is necessary for obtaining Turing expressiveness in the sense of Busi et al [22] hence providing an answer to question Q2 above. We do this by showing that there is no encoding of RAMs into $\text{CCS}_!^{-!\nu}$ which preserves and reflects convergence.

Furthermore, we shall also prove that up to failures equivalence, an standard equivalence in concurrency theory, there is no encoding from $\text{CCS}_!$ into $\text{CCS}_!^{-!\nu}$ nor from $\text{CCS}_!^{-!\nu}$ into $\text{CCS}_!^{-\nu}$. In other words, up to failures equivalence, we cannot encode all processes that may generate an unbounded number of restrictions with processes that can only generate a bounded number, nor all processes that may generate bounded number of restrictions with restriction-free processes.

In the light of the above-mentioned results, one may now wonder whether some other natural process construction can replace the use in $\text{CCS}_!$ of unboundedly many restrictions for achieving Turing expressiveness. We shall answer pos-

itively this question by considering a third variant $\text{CCS}^{-!\nu}_{!+pr}$ which extends $\text{CCS}^{-!\nu}_{!}$ with Phillips' priority guards [76]. This bears witness to the expressive power of this guarding construction.

Part II: The Asynchronous π-calculus

In [70] the authors presented an expressiveness study of *linearity* and *persistence* of processes in the asynchronous version of the π-calculus, henceforth Aπ. Linearity (and persistence) is understood in a similar sense of Girard's linear logic; the ability (incapability) of consuming a resource. The *replication* operator is central in [70] and plays a role similar to the "bang" operator from linear logic, also denoted as !.

Chapter 5: Linearity and Persistence The study in [70] is conducted in the *asynchronous* π-calculus (Aπ), which naturally captures the notion of linearity and persistence also present in other calculi.

Let us for example consider π-calculus system

$$\bar{x}z \mid x(y).P \mid x(y).Q$$

This system represents a *linear* message with a datum z, tagged with x, that can be *consumed* by either (linear) receiver $x(y).P$ or $x(y).Q$. Persistent messages (and receivers) can simply be specified using the *replication operator* which, as previously mentioned, creates an unbounded number of copies of a given process. One can then consider the existence of encodings from Aπ into three sub-languages of it, each capturing one source of persistence: the *persistent-input* calculus (PIAπ), defined as Aπ where inputs are replicated; the *persistent-output* calculus (POAπ), defined dually, i.e. outputs rather than inputs are replicated; the *persistent* calculus (PAπ), defined as Aπ but with all inputs and outputs replicated.

The main result in [70] basically states that we need one source of linearity, i.e. either on inputs (PIAπ) or outputs (POAπ) to encode the behavior of arbitrary Aπ processes via weak barbed congruence.

The notion of linearity (persistency) is present is several concurrency frameworks. *Persistence of messages* is present , e.g., in Concurrent Constraint Pro-

gramming (CCP) [84], Winskel's SPL [31] and the Spi Calculus variants in [35, 5]. In all these formalisms messages cannot be consumed. In the π-calculus persistent receivers are used, for instance, to model functions, objects, higher-order communications, or procedure definitions. Furthermore, persistence of *both* messages and receivers arise in the context of CCP with universally-quantified persistent ask operations [67] and in the context of calculi for security, persistent receivers can be used to specify protocols where principals are willing to run an unbounded number of times (and persistent messages to model the fact that every message can be remembered by the spy [12]).

Now, the previously mentioned positive result in [70] may give insights in the context of the expressiveness of the above frameworks. The main drawback of the work [70] is, however, that the notion of correctness for the encodings is based on weak barbed bisimulation (congruence), which is not sensitive to *divergence*. In particular, the encoding provided in [70] from Aπ into PIAπ is weak barbed congruent preserving but not divergence preserving. Although in some situations divergence may be ignored, in general it is an important issue to consider in the correctness of encodings [26, 44, 43, 27, 24, 69].

As a matter of fact the informal claims of extra expressivity of Linear CCP over CCP in [11, 34] are based on discrimination introduced by divergence that is clearly ignored by the standard notion of weak bisimulation. Furthermore, in [30] the author suggests as future work to extend SPL, which uses only persistent messages and replication, with recursive definitions to be able to program and model recursive protocols such as those in [4, 73]. Nevertheless, one can give an encoding of recursion in SPL from an easy adaptation of the composition between the Aπ encoding of recursion [83] (where recursive calls are translated into *linear* Aπ outputs and recursive definitions into persistent inputs) and the encoding of Aπ into POAπ in [70]. The resulting encoding is correct up to weak bisimilarity. The encoding of Aπ into POAπ, however, introduces divergence and hence the composite encoding does not seem to invalidate the justification for extending SPL with recursive definitions.

The above works suggest that the expressiveness study of persistence of [70] is relevant but incomplete since divergence is ignored. We therefore ask ourselves:

Q3: *Can the above-mentioned linearity be captured in (semi) persistent calculi without introducing divergence ?*

In the Chapter 5 we shall study the existence of encodings from Aπ into the persistent sub-languages mentioned above using testing semantics [65] which takes divergence into account. We shall provide a uniform and general negative answer to question Q3 by stating that, under some reasonable conditions, Aπ cannot be encoded into any of the above (semi) persistent calculi while preserving the $must$ testing semantics. The general conditions involve compositionality on the encoding of constructors such as parallel composition, prefix, and replication. The main result contrasts and completes the ones in [70]. It also supports the informal claims of extra expressivity mentioned above.

We shall also state other more specialized impossibility results for must preserving encodings from Aπ into the semi-persistent calculi, focusing on specific properties of each target calculus. This helps clarifying some previous assumptions on the interplay between syntax and semantics in encodings of process calculi. We believe that, since the study is conducted in Aπ with well-established notions of equivalence, our results can be easily adapted to other asynchronous frameworks such as CCP languages and the above-mentioned calculi for security.

Contributions and Organization

In summary this dissertation extends and strengthens the works [22, 70] by singling out key aspects of these works and then filling their gaps in the context of the expressiveness in concurrency theory. Among the other previously mentioned results, we shall show that without the divergence mechanism used in [22] for encoding Minsky machines, CCS$_!$ can capture regular but not context-free behaviours. We shall also show that the scoping of replication (or recursion) over restriction in the CCS$_!$ encoding in [22] (and in other π-based encodings of Turing-powerful models) is necessary to achieve Turing completeness. We shall show that under some natural conditions, the linearity of the asynchronous π-calculus [70] cannot be encoded with semi-persistent asychronous π-calculi.

All in all, we shall show how the restriction operator, replication and their interplay play a fundamental and complementary role in the computational ex-

pressiveness of some π-based calculi from the literature.

Organization

This dissertation is organized as follows: We begin with two introductory chapters followed by the two parts previously discussed in the Introduction. We conclude with some discussion about future work and an index table with the most relevant notions and notations.

The first chapter motivates and discusses our work. In the second chapter we provide some preliminary notions and prove some general properties that will be used in the forthcoming chapters.

Each chapter in Part I and Part II begins with an introduction explaining the motivation, lines of developments and the contributions, and it concludes with a short summary and a discussion about related work.

Chapters 3-4 are included in Part I and were described earlier in the Introduction. The third chapter provides the first classification based on the Chomsky Hierarchy. The fourth chapter provides the classification based on the scoping of replication over restriction.

Chapter 5 is included in Part II and was also previously described in the Introduction. This chapter provides a classification of linearity vs persistence in the asychronous π-calculus.

Contributions

Most of the material of this dissertation has been previously reported in the following works. The unpublished material will be explicitely mentioned in each chapter.

- J. Aranda, F. Valencia and C. Versari. On the Expressive Power of Restriction and Priorities in CCS with Replication. In FoSSaCS 2009: 242-256. Springer-Verlag. 2009.

  The contributions of this paper are included in Chapter 4.

- J. Aranda, C. Di Giusto, M. Nielsen and F. Valencia. CCS with Replication in the Chomsky Hierarchy: The Expressive Power of Divergence. APLAS 2007: 383-398. Springer-Verlag. 2007.

  The contributions of this paper are included in Chapter 3.

- D. Cacciagrano, F. Corradini, J. Aranda, F. Valencia. Persistence and Testing Semantics in the Asynchronous Pi Calculus. in EXPRESS'07, Electr. Notes Theor. Comput. Sci. 194(2): 59-84. Elsevier. 2007.

  The contributions of this paper are included in Chapter 5.

- J. Aranda, C. Di Giusto, C. Palamidessi and F. Valencia. On Recursion, Replication and Scope Mechanisms in Process Calculi. FMCO 2006: 185-206 Springer-Verlag. 2006.

  The contributions of this paper are included in Chapter 3.

Chapter 2

Preliminaries

This chapter introduces some basic notions, notations and frameworks that will be used in the following chapters. In particular, we shall introduce two subcalculi of the *polyadic* π-calculus: Namely the *asynchronous monadic* π-calculus (Aπ) [15, 47] and the *zero-adic* π-calculus (CCS$_!$) [21]. We shall recall notions such as bisimilarity, language and failures equivalence and frameworks such as Petri nets, and random access machines. We shall also prove some general properties about bisimilarity, failures and language equivalence.

2.1 The π-calculus

In what follows we shall introduce the π-calculus and the variants relevant for this thesis.

2.1.1 Syntax

Names are the most primitive entities in the π-calculus. We presuppose a countable set $\mathcal{N}$ of (port, links or channel) *names* , ranged over by $x, y, \ldots$. For each name x, we assume a *co-name* $\overline{x}$ thought of as *complementary*, so we decree that $\overline{\overline{x}} = x$. We use $\vec{x}$ to denote a finite sequence of names $x_1 x_2 \cdots x_n$. The other entity in the π-calculus is a *process.* Processes are built from names as follows.

Definition 2.1.1 *(Syntax) The processes, the summations and the prefixes in* π-

calculus are given respectively by:

$$\begin{aligned} P &:= M \,\big|\, (\nu x)P \,\big|\, P \mid P \,\big|\, \sigma \,\big|\, !P \\ M &:= 0 \,\big|\, \pi.P \,\big|\, M + M' \\ \pi &:= x(y) \,\big|\, \overline{x}y \,\big|\, \tau \end{aligned}$$

First we explain the summations and the prefixes and then the processes. The process (summation) 0 does nothing. $\overline{x}y.P$ and $x(y).P$ represent the output and input process respectively, $\overline{x}y.P$ is a process which can output a datum y on channel x and then it behaves like P, $\overline{x}y$ is called a guard or *(output) prefix* . $x(y).P$ is a process which can perform an input action on channel x and then it behaves like $P\{z/y\}$, the process which has replaced every occurrence of the name y, by the datum z received, $\{z/y\}$ is a substitution of z by y, $x(y)$ is called a guard or *(input) prefix* . The unobservable prefix $\tau.P$ can evolve invisibly to P. τ can be thought of as expressing an internal action of a process, τ is called a guard or *(unobservable) prefix* . the sum (or choice) $P + Q$ represents the process which can performs the capabilities of either P or Q but not both. Once a capability of P (Q) has been performed, Q (P) is disregarded. In $P \mid Q$, the parallel composition of P and Q, P and Q can proceed independently or can synchronise via shared names. In $(\nu x)P$, the name x is declared private to P, i.e. initially, components of P can use x to interact with one another but not with other processes, the scope of x could change as a result of interaction between processes as will be seen later. Finally, the replication $!P$ can be thought of as unboundedly many P's in parallel $P \mid P \mid P \mid \ldots$, replication is the means to express infinite behaviour.

Notice that the operands in a sum must themselves be summations. Hence it says that the π-calculus considers guarded-choice.

In each of $x(y).P$ and $(\nu y)P$, the occurrence of y is *binding* with *scope* P. An occurrence of a name in a process is *bound* if it is under the scope of a biding occurrence of the name. An occurrence of a name is *free* if it is not bound. Given Q we define its *bound names* $bn(Q)$ as the set of names with a bound occurrence in Q, and its *free names* $fn(Q)$ as the set of names with a non-bound occurrence in Q, hence $n(Q) = fn(Q) \cup bn(Q)$ is the set of names of Q.

As consequence of the interchange of names between processes an unintended capture of names by binders could arise, to avoid it, the following definition of α-convertability is useful.

Definition 2.1.2 *(α-convertability) [83]*

1. *If the name w does not occur in the process P, then $P\{w/z\}$ is the process obtained by replacing each occurence of z in P by w.*
2. *A change of bound names in a process P is the replacement of a subterm $x(z).P$ of P by $x(z).Q\{w/z\}$, or the replacement of a subterm $(\nu z)Q$ of a P by $(\nu w)Q\{w/z\}$, where in each case w does not occur in Q.*
3. *Processes P and Q are α-convertible, $P = Q$, if Q can be obtained from P by a finite number of changes of bound names.*

Hence we adopt two well-known conventions:

Convention 2.1.1 *[83] Processes that are α-convertible are identified.*

Convention 2.1.2 *[83] When considering a collection of processes and substitutions, we assume that the bound names of the processes are chosen to be different from their free names and from the names of the substitutions.*

2.1.2 Semantics

The above description is made precise by a labelled transition system. A transition rule $P \xrightarrow{\alpha} Q$ says that P can perform an *action* α and evolve into Q, the set of actions used in the transition system is composed by $\bar{x}y$, xy, $\bar{x}(y)$, τ. $\bar{x}y$, *a free output* , sends the name y on the name x, xy, *an input* , receives the name y on the name x, $\bar{x}(y)$, *a bound output* , sends a fresh name on x and τ is an *internal action* .

Definition 2.1.3 *(Actions) The actions, which are ranged over by α, are given by:*

$$\alpha \quad := \quad 0 \;\Big|\; \bar{x}y \;\Big|\; xy \;\Big|\; \bar{x}(y) \;\Big|\; \tau \tag{2.1}$$

Act refers the set of actions. The set of labels*, ranged over by l and l', is $\mathcal{L}$ which is composed of all non-internal actions.*

Functions $fn(\_)$, $bn(\_)$ and $n(\_)$ are extended to cope with labels as follows:

$$\begin{array}{lllllllllll} bn(xy) & = & \emptyset & bn(\bar{x}y) & = & \emptyset & bn(\bar{x}(y)) & = & \{y\} & bn(\tau) & = & \emptyset \\ fn(xy) & = & \{x,y\} & fn(\bar{x}y) & = & \{x,y\} & fn(\bar{x}(y)) & = & \{x\} & fn(\tau) & = & \emptyset \end{array}$$

The *subject* , $subj(\_)$, and *object* , $obj(\_)$, of these actions is defined as: $subj(xy) = subj(\bar{x}y) = subj(\bar{x}(y)) = x$, $obj(xy) = obj(\bar{x}y) = obj(\bar{x}(y)) = y$, $subj(\tau) = subj(\tau) = \emptyset$.

Definition 2.1.4 *(Semantics) The labelled transition relation $\xrightarrow{\alpha}$ is given by the rules in Table 2.1. Ommited from Table 2.1 are the symmetric forms of Sum-L, Par-L, Com-L and Close-L. Let us define the relation $\stackrel{\alpha}{\Longrightarrow}$, with $s = \alpha_1 \ldots \alpha_n \in Act^*$, as $(\xrightarrow{\tau})^* \xrightarrow{\alpha_1} (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* \xrightarrow{\alpha_n} (\xrightarrow{\tau})^*$. $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$. $\stackrel{\hat{\tau}}{\Longrightarrow}$ is $\Longrightarrow$ and $\stackrel{\hat{\beta}}{\Longrightarrow}$ is $\stackrel{\beta}{\Longrightarrow}$.*

Some comments on the rules: the side-condition in Rule Par-L rule avoids the capture of a name by the extrusion of the scope of another name. The Open rule expresses extrusion of the scope of a name, this action allows the passing of a name beyond its original scope, its side-condition avoids the execution of an action whose subject is a bound-name as it should not interact with other processes out of the scope of the name. Rule Close-L reflects the interaction between processes in which the left-process has transmitted a bound name to the right-process, thus the scope of the restricted name is extended to include the process which receives it.

Remark 2.1.1 *We abbreviate, for any names x, y, the guards $x(y)$ and $\bar{x}y$ by x and $\bar{x}$, respectively, where y, is a dummy name: in theses cases the datum which can be received or sent is irrelevant*

Notation 2.1.1 *Throughout this dissertation, we use $(\nu a_1 \ldots a_n)P$ as a short hand for $(\nu a_1) \ldots (\nu a_n)P$. We often omit the "$0$" in $\alpha.0$.*

Now we define $\equiv$ which shall be useful in the dissertation although it is not included in the semantics:

$$\text{Input} \quad x(y).P \xrightarrow{xz} P\{z/y\} \text{ where } x,\ y \in \mathcal{N}$$

$$\text{Output} \quad \bar{x}y.P \xrightarrow{\bar{x}y} P \qquad\qquad \text{Tau} \quad \tau.P \xrightarrow{\tau} P$$

$$\text{Sum-L} \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\text{Open} \quad \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(y)} P'}\, x \neq y \qquad\qquad \text{Res} \quad \frac{P \xrightarrow{\alpha} P'}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'}\, y \notin n(\alpha)$$

$$\text{Par-L} \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\, bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{Com-L} \quad \frac{P \xrightarrow{\bar{x}y} P',\ Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad\qquad \text{Close-L} \quad \frac{P \xrightarrow{\bar{x}(y)} P',\ Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')}$$

$$\text{Rep-Act} \quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$\text{Rep-Comm} \quad \frac{P \xrightarrow{\bar{x}y} P',\ P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\text{Rep-Close} \quad \frac{P \xrightarrow{\bar{x}(z)} P',\ P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} ((\nu z)(P' \mid P'')) \mid !P}\, z \notin fn(P)$$

Table 2.1: Operational semantics for the π-calculus.

Definition 2.1.5 *Let $\equiv$ the smallest congruence over processes satisfying α-equivalence, the commutative monoid laws for composition with 0 as identity, the replication law $!P \equiv P|!P$, the restriction laws $(\nu x)0 \equiv 0$, $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$ and the extrusion law: $(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$ if $x \in fn(P)$.*

An important property of $\equiv$ is that it respects the transitions of the operational semantics:

Proposition 2.1.1 (First assertion of Harmony Lemma) *[83]*
$P \equiv \xrightarrow{\alpha} P'$ implies $P \xrightarrow{\alpha} \equiv P'$.

Remark 2.1.2 *All the calculi studied in this dissertation are restrictions of the* polyadic π-calculus . *The polyadic π-calculus is a natural and convenient exten-*

sion of π-calculus where it is admitted processes that pass tuples of names, in this case the prefixes are $\bar{x}\vec{y}$, $x(\vec{z}), \tau$ where $\vec{y}$ is a tuple of names. Thus, the π-calculus and its variant asynchronous π-calculus, described in Section 2.2, calculi where an interaction involves the transmission of a single name from one process to another, can be seen as restrictions of the polyadic π-calculus where the size of the tuple is 1 . Similarly, the Calculus of Communicating Systems (CCS), described in Section 2.3, can be considered a restriction of the polyadic π-calculus where the size of the tuple is 0.

The definitions and concepts in this Section are extended naturally to them, subject to the specific features of them.

2.2 The asynchronous π-calculus: Aπ

Communication in π-calculus is considered synchronous. The key property relies on the fact that the output and the input prefix impose a precedence over the terms which are underneath such that once a communication involving the output and the input prefix occurs, the terms which were underneath the prefixes are unguarded at the same time. This behaviour can be seen as a kind of acknowledgement of the execution of the communication over the processes involved in it.

Asynchronous π-calculus (Aπ) is a variant of the π-calculus introduced in [47, 15]. In this variant the communication can be as asynchronous, in the sense that the act of sending a datum and the act of receiving it can be seen as separate, hence not simultaneous. It is achieved by restricting the term underneath the output prefix to be 0 (the null process). In this way the kind of acknowledge provided by the precedence in the output prefix is lost. Moreover, an unguarded occurrence of $\bar{x}y$ can be thought of as a datum y in an implicit communication medium, tagged with x to indicate that it is available to any unguarded term of the form $x(z).P$. Thus, in the evolution of a term, the datum y can be considered to be sent when $\bar{x}y$ becomes unguarded, and to be received when $\bar{x}y$ disappears via an internal action.

We consider a version of Aπ without τ and $choice$ as proposed in [47, 15]. In general, the definitions, conventions and notions in Section 2.1 apply to Aπ, of course taking into account the differences between the π and the Aπ calculi described previously. However let us see the syntax and semantics for Aπ.

Definition 2.2.1 *(Syntax) Processes in Aπ-calculus are given respectively by:*

$$P, Q, \ldots \;:=\; 0 \;\Big|\; x(y).P \;\Big|\; \overline{x}y \;\Big|\; (\nu x)P \;\Big|\; P \mid Q \;\Big|\; !P \tag{2.2}$$

Definition 2.2.2 *(Semantics) The labelled transition relation $\xrightarrow{\alpha}$ is given by the rules in Table 2.2. Ommited from Table 2.2 are the symmetric forms of Par-L, Com-L and Close-L.*

$$\text{Input} \quad x(y).P \xrightarrow{xz} P\{z/y\} \text{ where } x,\ y \in \mathcal{N}$$

$$\text{Output} \quad \bar{x}y \xrightarrow{\bar{x}y} 0$$

$$\text{Open} \quad \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(y)} P'} x \neq y \qquad \text{Res} \quad \frac{P \xrightarrow{\alpha} P'}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'} y \notin n(\alpha)$$

$$\text{Par-L} \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{Com-L} \quad \frac{P \xrightarrow{\bar{x}y} P',\ Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{Close-L} \quad \frac{P \xrightarrow{\bar{x}(y)} P',\ Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')}$$

$$\text{Rep-Act} \quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$\text{Rep-Comm} \quad \frac{P \xrightarrow{\bar{x}y} P',\ P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\text{Rep-Close} \quad \frac{P \xrightarrow{\bar{x}(z)} P',\ P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} ((\nu z)(P' \mid P'')) \mid !P} z \notin fn(P)$$

Table 2.2: Operational semantics for the Aπ-calculus.

2.3 The Calculus of Communicating Systems

Undoubtedly CCS [59], a calculus for synchronous communication, remains as a standard representative of process calculi. In fact, many foundational ideas in the theory of concurrency have sprung from this calculus. In the following we

shall consider two variants of CCS according to its mechanism to model infinite behaviour. Hence, first we show the Finite fragment of CCS and the we introduce the two infinite extensions.

2.3.1 Finite CCS

The finite CCS processes can be obtained as a restriction of the finite processes of the π-calculus, i.e those π processes without occurrence of a term of the form $!P$, by requiring all inputs and outputs to have empty subjects only. Intuitively, this means that in CCS there is no sending/receiving of links but synchronisation on them.

In CCS, the actions are names, co-names and τ and therefore, we shall use $l, l', \ldots$ to range over names and co-names, where $\mathcal{L}$ is the set of names and co-names. The set of *actions* Act, ranged over by α and β, extends $\mathcal{L}$ with the symbol τ.

The syntax of finite CCS processes would be the following:

Definition 2.3.1 *(Syntax) Processes in finite CCS are given respectively by:*

$$P, Q, \ldots \;:=\; 0 \;\Big|\; x.P \;\Big|\; \overline{x}.P \;\Big|\; (\nu x)P \;\Big|\; P \mid Q \;\Big|\; P + Q \tag{2.3}$$

Definition 2.3.2 *(Semantics) The labelled transition relation $\xrightarrow{\alpha}$ is given by the rules in Table 2.3. Ommited from Table 2.3 are the symmetric forms of Par-L, Com-L and Close-L.*

In this document, we consider two variants of CCS which extends the above syntax to express infinite behaviour in a different way. We describe them next.

2.3.2 Replication: CCS$_!$

As said before, replication is the way of expressing infinite behaviour which has been used in the π-calculus and the Aπ-calculus. It has also studied in the context of CCS in [21, 38].

For replication the syntax of finite processes (Table 2.3) is extended as follows:

$$\text{Input} \quad x.P \xrightarrow{x} P$$

$$\text{Output} \quad \bar{x}.P \xrightarrow{\bar{x}} P \qquad \text{Tau} \quad \tau.P \xrightarrow{\tau} P$$

$$\text{Sum-L} \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{Res} \quad \frac{P \xrightarrow{\alpha} P'}{(\nu y)P \xrightarrow{\alpha} (\nu y)P'}\, y \notin n(\alpha)$$

$$\text{Par-L} \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{Com-L} \quad \frac{P \xrightarrow{\bar{x}} P', \;\; Q \xrightarrow{x} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Table 2.3: Operational semantics for the finite CCS.

$$P, Q, \ldots := \ldots \mid !P \tag{2.4}$$

CCS$_!$ is the restriction of the π-calculus seen by requiring all inputs and outputs to have empty subjects only.

The operational rules for CCS$_!$ are those in Table 2.3 plus the following rules:

$$\text{Rep-Act} \quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \text{Rep-Comm} \quad \frac{P \xrightarrow{\bar{x}y} P' \quad P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} P' \mid P'' \mid !P}$$

Table 2.4: Transition Rules for Replication in $CCS!$

2.3.3 Parametric Definitions: CCS and CCS$_\mathtt{P}$

A typical way of specifying infinite behaviour is by using parametric definitions [61]. In this case we extend the syntax of finite processes (Equation 4.1) as follows:

$$P, Q, \ldots := \ldots \mid A(y_1, \ldots, y_n) \tag{2.5}$$

Here $A(y_1, \ldots, y_n)$ is an *identifier* (also *call*, or *invocation*) of arity n. We assume that every such an identifier has a unique, possibly recursive, *definition* $A(x_1, \ldots, x_n) \stackrel{\texttt{def}}{=} P_A$ where the x_i's are pairwise distinct, and the intuition is that $A(y_1, \ldots, y_n)$ behaves as its *body* P_A with each y_i replacing the *formal parameter* x_i. For each $A(x_1, \ldots, x_n) \stackrel{\texttt{def}}{=} P_A$, we require $fn(P_A) \subseteq \{x_1, \ldots, x_n\}$.

Following [38], we should use CCS_{p} to denote the calculus with parametric definitions with the above syntactic restrictions.

Remark 2.3.1 *As shown in [38], however, CCS_{p} is equivalent w.r.t. strong bisimilarity to the standard CCS. We shall then take the liberty of using the terms CCS and CCS_{p} to denote the calculus with parametric definitions as done in [61].*

The rules for CCS_{p} are those in Table 2.3 plus the rule:

$$\text{CALL}\ \frac{P_A[y_1, \ldots, y_n/x_1, \ldots, x_n] \stackrel{\alpha}{\longrightarrow} P'}{A(y_1, \ldots, y_n) \stackrel{\alpha}{\longrightarrow} P'}\ \text{if } A(x_1, \ldots, x_n) \stackrel{\texttt{def}}{=} P_A \qquad (2.6)$$

As usual $P[y_1 \ldots y_n/x_1 \ldots x_n]$ results from replacing every free occurrence of x_i with y_i renaming bound names in P wherever needed to avoid capture.

2.4 Notions and equivalences

The following notions are used in our expressiveness study throughout the thesis.

A central concept is the notion of *encoding* : A map from the terms of a π-calculus variant (e.g., CCS_{p}) into the terms of another (e.g., $\text{CCS}_!$). The existence of encodings that satisfy certain properties is typically used as a measure of expressiveness (see [26, 44, 43, 27, 24, 69]).

We now introduce the process equivalences we will use in the forthcoming chapters.

2.4.1 Bisimilarity

In Section 4.2.1, we shall make use of the reduction bisimilarity and strong bisimilarity equivalences which preserve convergence.

Definition 2.4.1 (Reduction Bisimilarity) *A* reduction simulation *is a binary relation $\mathcal{R}$ satisfying the following: $(P, Q) \in \mathcal{R}$ implies that:*

- *if $P \xrightarrow{\tau} P'$ then $\exists Q' : Q \xrightarrow{\tau} Q' \wedge (P', Q') \in \mathcal{R}$.*

The relation $\mathcal{R}$ is a reduction bisimulation *iff both $\mathcal{R}$ and its converse $\mathcal{R}^{-1}$ are reduction simulations. We say that P and Q are* reduction bisimilar*, written $P \sim_r Q$ iff $(P, Q) \in \mathcal{R}$ for some reduction bisimulation $\mathcal{R}$.*

Definition 2.4.2 (Strong Bisimilarity) *A* strong simulation *is a binary relation $\mathcal{R}$ satisfying the following: $(P, Q) \in \mathcal{R}$ implies that:*

- *if $P \xrightarrow{\alpha} P'$ then $\exists Q' : Q \xrightarrow{\alpha} Q' \wedge (P', Q') \in \mathcal{R}$.*

The relation $\mathcal{R}$ is a strong bisimulation *iff both $\mathcal{R}$ and its converse $\mathcal{R}^{-1}$ are strong simulations. We say that P and Q are* strong bisimilar*, written $P \sim Q$ iff $(P, Q) \in \mathcal{R}$ for some strong bisimulation $\mathcal{R}$.*

Definition 2.4.3 (Weak Bisimilarity) *A* (weak) simulation *is a binary relation $\mathcal{R}$ satisfying the following: $(P, Q) \in \mathcal{R}$ implies that:*

- *if $P \stackrel{s}{\Longrightarrow} P'$ where $s \in \mathcal{L}^*$ then $\exists Q' : Q \stackrel{s}{\Longrightarrow} Q' \wedge (P', Q') \in \mathcal{R}$.*

The relation $\mathcal{R}$ is a bisimulation *iff both $\mathcal{R}$ and its converse $\mathcal{R}^{-1}$ are simulations. We say that P and Q are* (weak) bisimilar*, written $P \approx Q$ iff $(P, Q) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$.*

Aπ bisimilarity

Let us now recall some standard process equivalences for the special case of Aπ. First we recall a basic notion of observation.

Definition 2.4.4 (*Barbs*) Define $P \downarrow_{\bar{x}}$ iff $\exists z_1, \ldots, z_n, y, R : P \equiv (\nu z_1)..(\nu z_n)(\bar{x}y \mid R)$ and $\forall i \in [1..n], x \neq z_i$. Furthermore, $P \Downarrow_{\bar{x}}$ iff $\exists Q : P \Longrightarrow Q \downarrow_{\bar{x}}$.

Intuitively, we say that $\bar{x}$, a barb, can be (strongly) observed at an Aπ process P, written $P \downarrow_{\bar{x}}$, iff P can perform an output on channel x. We also say that $\bar{x}$ can be weakly observed at P, written $P \Downarrow_{\bar{x}}$, iff P can perform an output on channel x after zero or more τ transitions.

We now recall the standard notion of asynchronous barbed bisimilarity.

Definition 2.4.5 (*Asynchronous barbed bisimilarity and barbed congruence*) [83] An *asynchronous weak barbed bisimulation* is a symmetric relation $\mathcal{R}$ satisfying the following: $(P, Q) \in \mathcal{R}$ implies that:

1. $P \xrightarrow{\tau} P'$ then $\exists Q' : Q \Longrightarrow Q' \wedge (P', Q') \in \mathcal{R}$.

2. $P \downarrow_{\bar{x}}$ then $Q \Downarrow_{\bar{x}}$.

We say that P and Q are *asynchronous weak barbed bisimilar*, written $P \dot{\approx}_a \ Q$, iff $(P, Q) \in \mathcal{R}$ for some asynchronous weak barbed bisimulation $\mathcal{R}$. Furthermore, *asynchronous weak barbed congruence* $\approx_a$ is defined as: $P \approx_a \ Q$ iff for every process context $C[\cdot]$, $C[P] \dot{\approx}_a \ C[Q]$.

2.4.2 Language and failures equivalences

In Chapters 3 and 4 we shall use the notion of language and failures in order to measure the expressive power of the calculi. Language notion is particularly suitable for the Chapter 3 where the comparison involves different computability models. The notion of failure is central in Chapter 4 as failures equivalence allows to study and compare different calculi by considering convergence as fundamental. The notion of language shall used again in Chapter 4 as the directed relation beween languages and failures, which will be defined formally in 2.4.2 .

Following [9], we say that a process generates a sequence of non-silent actions s if it can perform the actions of s in a finite maximal sequence of transitions. More precisely:

Definition 2.4.6 (Sequence and language generation) *The process P* generates *a sequence $s \in \mathcal{L}^*$ if and only if there exists Q such that $P \stackrel{s}{\Longrightarrow} Q$ and $Q \stackrel{\alpha}{\not\longrightarrow}$ for any $\alpha \in Act$. Define the language of (or generated by) a process P, $L(P)$, as the*

set of all sequences P generates. We say that P and Q are language equivalent , written $P \sim_L Q$, iff $L(P) = L(Q)$.

The above definition basically states that a sequence is generated when no transition rules can be applied. It is clearly related to the notion of language generation of models of computation we are comparing our processes with in this thesis (see Chapter 3). Namely, formal grammars where a sequence is generated when no rewriting rules can be applied.

We recall the notion of *failure* following [60]. We first need the following notion:

Definition 2.4.7 *We say that P is stable iff $P \stackrel{\tau}{\not\longrightarrow}$.*

Intuitively we say that a pair $\langle e, L\rangle$, with $e \in \mathcal{L}^*$ and $L \subseteq \mathcal{L}$, is failure of P if P can perform e and thereby reach a state in which no further action (including τ) is possible if the environment will only allow actions in L.

Definition 2.4.8 (Failures) *A pair $\langle e, L\rangle$, where $e \in \mathcal{L}^*$ and $L \subseteq \mathcal{L}$, is a failure of P iff there is P' such that: (1) $P \stackrel{e}{\Longrightarrow} P'$, (2) $P' \stackrel{l}{\not\longrightarrow}$ for all $l \in L$, and (3) P' is stable.*

Define $Failures(P)$ as the set of failures of a process P. We say that P and Q are failures equivalent *, written $P \sim_F Q$ iff $Failures(P) = Failures(Q)$.*

We recall the notions of convergence and divergence following [21, 22]. Intuitively, a process *converges* if it can reach a stable process after a sequence of τ moves. A process is deemed *divergent* iff it can perform an infinite sequence of τ moves.

Definition 2.4.9 (Convergence and Divergence) *We say that P is* convergent*, $P \downarrow$, iff there is a stable process Q such that $P(\stackrel{\tau}{\longrightarrow})^*Q$. We say that P is* divergent*, $P \uparrow$, iff $P(\stackrel{\tau}{\longrightarrow})^\omega$, i.e., there exists an infinite sequence $P = P_0 \stackrel{\tau}{\longrightarrow} P_1 \stackrel{\tau}{\longrightarrow} \ldots$. $P \not\uparrow$ means P is not divergent and $P \not\downarrow$, P is not convergent.*

We conclude this section by stating relations between the above notions which we shall use in the rest of the document.

Some Basic Properties of Failures

As said before the suitability of failures in our study mainly relies on its sensitivity to convergence. The following proposition states it formally.

Proposition 2.4.1 *Suppose that $P \sim_F Q$. Then P is convergent iff Q is convergent.*

Proof. Suppose as a means of contradiction that $P \sim_F Q$ and that either (1) P is convergent but Q is not, or (2) P is not convergent but Q is. If we assume (1), we conclude that P has the failure $\langle \epsilon, \emptyset \rangle$ which Q does not, a contradiction. The other case is symmetric. □

To justify the rest of the above claim, take $P = \tau.!a.0$ and $P' = !\tau.0$. Clearly P converges but P' does not, however they are both language equivalent. Now take $Q = \tau.!\tau.0 + \tau.0$ and $Q' = !\tau.0$. Thus Q converges but Q' does not. It can be verified that Q and Q' are equated by these standard equivalences.

The relation between failures and languages is relevant in Chapter 4, as it facilitates the comparison between the calculi and it allows to make use of previous results from the literature.

Now, we show that failures equivalence implies language equivalence.

Proposition 2.4.2 $\sim_F \subseteq \sim_L$.

Proof. As a means of contradiction, let us suppose there are two processes P and Q such that $P \sim_F Q$ but $P \not\sim_L Q$. So either:

- There exists a string s such that $s \in L(P)$ and $s \notin L(Q)$. From $s \in L(P)$ we have $\{ \langle s, L \rangle \mid L \subseteq \mathcal{L} \} \subseteq Failures(P)$. Since $s \notin L(Q)$ we have the following two situations:

  - There is no R such that $Q \stackrel{s}{\Longrightarrow} R$. In this case then Q has no failure $\langle s, L \rangle$ for any $L \subseteq \mathcal{L}$, a contradiction.
  - For every R such that $Q \stackrel{s}{\Longrightarrow} R$ we have $R \stackrel{\alpha}{\longrightarrow}$. Let S be $\{ \alpha \mid \exists R : Q \stackrel{s}{\Longrightarrow} R \text{ and } R \stackrel{\alpha}{\longrightarrow} \}$. Clearly $\langle s, S \rangle$ is not a failure of Q but is failure of P, a contradiction.

- There exists a string s such that $s \in L(Q)$ and $s \notin L(P)$: Analogous to the previous one.

□

The following proposition shall be used to prove the correctness of the encoding studied in Section 4.5 up to $\sim_F$ by using the fact that the encoding preserves strong bisimulation.

Proposition 2.4.3 $\sim \subseteq \sim_F$.

Proof.

We shall that prove that for two any processes P and Q, if $P \sim Q$ then $Failures(P) = Failures(Q)$. Assuming $P \sim Q$, we prove $Failures(P) \subseteq Failures(Q)$ and $Failures(Q) \subseteq Failures(P)$ as follows:

- $Failures(P) \subseteq Failures(Q)$: Let $\langle e, L\rangle$ be a failure of P, by Definition 2.4.8 there is P' such that $P \stackrel{e}{\Longrightarrow} P'$, $P' \stackrel{l}{\not\longrightarrow}$ for all $l \in L$, and $P' \stackrel{\tau}{\not\longrightarrow}$. As $P \sim Q$ there is Q' such that $Q \stackrel{e}{\Longrightarrow} Q'$ where $P' \sim Q'$. As $P' \sim Q'$, $Q' \stackrel{l}{\not\longrightarrow}$ for all $l \in L$, and $Q' \stackrel{\tau}{\not\longrightarrow}$, therefore $\langle e, L\rangle \in Failures(Q)$ by Definition 2.4.8.

- $Failures(Q) \subseteq Failures(P)$: Analogous to the previous one.

□

2.4.3 Testing semantics

In Chapter 5, we shall use *Testing semantics*, a well-known framework sensitive to divergence, to measure the expressiveness of Aπ and its semi-persistent subcalculi. Testing semantics shed light on the expressiveness gap between Aπ and its fragments when divergence is taken into account.

In [65] De Nicola and Hennessy propose a framework for defining pre-orders that is widely acknowledged as a realistic scenario for system testing. It means to define formally when one process is a correct implementation of another considering specially unsafe contexts, in which is particularly important what is the revealed information of the process in any context or test. In this section we summarize the basic definitions behind the testing machinery for the π-calculi.

Definition 2.4.10 (Observers) - The set of names $\mathcal{N}$ is extended as $\mathcal{N}' = \mathcal{N} \cup \{\omega\}$ with $\omega \notin \mathcal{N}$. By convention we let $fn(\omega) = \{\omega\}$ and $bn(\omega) = \emptyset$ (ω is used to report success).

- The set $\mathcal{O}$ (ranged over by $o, o', o'', E, E', \ldots$) of observers (tests) is defined by following the syntax of the corresponding calculus, where the grammar is extended with the production $P := \omega.P$.

- $\xrightarrow{\omega}$ is the least predicate over $\mathcal{O}$ satisfying the inference rules in Table 2.5.

Omega	$\omega.E \xrightarrow{\omega}$	Res	$\dfrac{E \xrightarrow{\omega}}{(\nu y)E \xrightarrow{\omega}}$
Par	$\dfrac{E_1 \xrightarrow{\omega}}{E_1 \mid E_2 \xrightarrow{\omega}}$	Cong	$\dfrac{E' \xrightarrow{\omega} \quad E' \equiv E}{E \xrightarrow{\omega}}$

Table 2.5: Predicate $\xrightarrow{\omega}$.

Definition 2.4.11 (Maximal computations) Given a process P and $o \in \mathcal{O}$, a maximal computation from $P \mid o$ is either an infinite sequence of the form

$$P \mid o = E_0 \xrightarrow{\tau} E_1 \xrightarrow{\tau} E_2 \xrightarrow{\tau} \ldots$$

or a finite sequence of the form

$$P \mid o = E_0 \xrightarrow{\tau} E_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} E_n \not\xrightarrow{\tau} .$$

Definition 2.4.12 (May, must and fair relations[1]) Given a process P and $o \in \mathcal{O}$, define:

- P *may* o if and only if *there is* a maximal computation (as in Def. 2.4.11) such that $E_i \xrightarrow{\omega}$, for some $i \geq 0$;

- P *must* o if and only if *for every* maximal computation (as in Def. 2.4.11) there exists $i \geq 0$ such that $E_i \xrightarrow{\omega}$;

- P *fair* o [17] if and only if *for every* maximal computation (as in Def. 2.4.11) and $\forall i \geq 0$, $\exists\, E_i'$ such that $E_i \Longrightarrow E_i'$ and $E_i' \xrightarrow{\omega}$.

Correctness wrt testing: Concerning semantic correctness, we consider preservation of *sat* testing, where *sat* can be respectively *may*, $must$ and *fair*. Given an encoding $e = [\![\cdot]\!]$ from Aπ into some Aπ variant, we assume that its lifted version e' from the set of observers of π to the ones of $\mathcal{P}$ is an encoding satisfying the following: $e'(o) = e(o)$, in the case o has no occurrences of ω.

Definition 2.4.13 (Soundness, completeness and *sat*-preservation) We say that $[\![\cdot]\!]$ is:

- *sound w.r.t. sat* iff $\forall\, P \in \mathrm{A}\pi,\ \forall\, o \in \mathcal{O}$, $[\![P]\!]$ *sat* $[\![o]\!]$ implies P *sat* o;

- *complete w.r.t. sat* iff $\forall\, P \in \mathrm{A}\pi,\ \forall\, o \in \mathcal{O}$, P *sat* o implies $[\![P]\!]$ *sat* $[\![o]\!]$;

-*sat-preserving* iff $[\![\cdot]\!]$ is *sound* and *complete* w.r.t*sat.* Hence we have the definitions of *may* , $must$ and *fair* -preserving.

2.5 Petri Nets

We shall use some decidability results from one of the most representative models for concurrent behaviour: *Petri Nets* [81]. The Petri net theory is a generalization of the theory of automata to allow for the occurrence of several actions (state-transitions) independently.

A Petri net is a graph in which the nodes represent transitions (i.e. discrete events that may occur, signified by bars), places (i.e. conditions, signified by circles), and directed arcs (that describe which places are pre- and/or postconditions for which transitions, signified by arrows). More precisely,

Definition 2.5.1 (Petri Nets) *A Petri net is a tuple (S, T), where S is a set of* places*, T is a set of* transitions $\mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(S)$ *with $\mathcal{M}_{fin}(S)$ being a finite multiset of S called a* marking.

A transition (c, p) is written in the form $c \Longrightarrow p$. A transition is enabled at a marking m if $c \subseteq m$. The execution of the transition produces the marking m' $= (\ m \setminus c\ ) \oplus p$ (where $\setminus$ and $\oplus$ are the difference and the union operators on multisets). This is written as $m \rhd m'$. If no transition is enable at m we say that m is a dead marking. *A marked Petri net is a tuple (S, T, m_0), where (S, T) is a Petri net and m_0 is the initial marking.*

A central property we shall use in Chapter 4 is the decidability of the convergence problem for Petri Nets.

Definition 2.5.2 *We say that the marked Petri net* (S, T, m_0) converges *iff there exists a dead marking* m' *such that* $m_0(\triangleright)^* m'$.

Theorem 2.5.1 *[33] The convergence problem for Petri Nets is decidable.*

2.6 Random Access Machines RAMs

In Chapters 3 and 4 we will encode Random Access Machines into a certain calculus in order to show that the calculus is Turing-expressive.

A Random Access Machine, RAM [63] $M(v_0, v_1, \ldots, v_n)$ is a Turing-complete computational model which consists of a finite set of registers R_1, R_2, ...holding arbitrary large natural numbers and initialised with the values v_0 and v_1, ..., v_n and a program, i.e. a finite sequence of numbered instructions which modify the registers. There are three types of instructions $j : Inst()$ where j is the number of the instruction:

- $j : Succ(R_i)$: adds 1 to the content of register R_i and goes to instruction $j + 1$;
- $j : DecJump(R_i, l)$: if the content of the register R_i is not zero, then decreases it by 1 and goes to instruction $j+1$, otherwise jumps to instruction l;
- $j : Halt$: stops computation and returns the value in register R_1.

where $1 \leq i \leq 2$, $j, l \leq n$ and n is the maximum number of instructions of the program.

An internal state of the machine is given by a tuple $(p_i, r_1, r_2, \ldots, r_n)$ where the program counter p_i indicates the next instruction and r_1, r_2, ..., r_n are the current contents of the registers. Given a program, its computation proceeds by executing the instructions as indicated by the program counter. The execution stops when an instruction number higher than the length of the program is reached, it is equivalent to reach a $Halt$ instruction.

Part I

Chapter 3

$CCS_!$ in the Chomsky Hierarchy

A remarkable result in [22] shows that in spite of its being strictly less expressive than CCS w.r.t. weak bisimilarity, CCS with replication ($CCS_!$) is Turing powerful. This is done by encoding Random Access Machines (RAM) in $CCS_!$. The encoding is said to be *non-faithful*, in the sense that it may move from a state which can lead to termination into a non-convergent one which do not correspond to any configuration of the encoded RAM. I.e., the encoding is not termination preserving.

In this chapter we study the existence of faithful encodings into $CCS_!$ of models of computability *strictly less* expressive than Turing Machines. Namely, grammars of Types 1 (Context Sensitive Languages), 2 (Context Free Languages) and 3 (Regular Languages) in the Chomsky Hierarchy. We provide faithful encodings of Type 3 grammars. We show that it is impossible to provide a faithful encoding of Type 2 grammars and that termination-preserving $CCS_!$ processes can generate languages which are not Type 2. We finally show that the languages generated by termination-preserving $CCS_!$ processes are Type 1 .

The classification of the termination-preserving $CCS_!$ processes in the Chomsky Hierarchy in this chapter was originally published as [7]. In addition to the work in [7], in this chapter we prove that the set of termination-preserving $CCS_!$ processes is undecidable.

3.1 Introduction

Infinite behaviour is ubiquitous in concurrent systems. Hence, it ought to be represented by process terms. In the context of CCS we can find at least two representations of them: Recursive definitions and Replication.

An interesting result is that in the π-calculus, itself a generalisation of CCS, parametric recursive definitions can be encoded using replication up to weak bisimilarity. This is rather surprising since the syntax of $!P$ and its description are so simple. In fact, in [21] it is stated that in CCS recursive expressions are more expressive than replication. More precisely, it is shown that it is impossible to provide a weak-bisimilarity preserving encoding from CCS with recursion, into the CCS variant in which infinite behaviour is specified only with replication. From now on we shall use CCS to denote CCS with recursion and CCS$_!$ to the CCS variant with replication.

Now, a remarkable expressiveness result in [22] states that, in spite of its being less expressive than CCS in the sense mentioned above, CCS$_!$ is Turing powerful. This is done by encoding (Deterministic) Random Access Machines (RAM) in CCS$_!$. Nevertheless, the encoding is not faithful (or deterministic) in the sense that, unlike the encoding of RAMs in CCS, it may introduce computations which do not correspond to the expected behaviour of the modeled machine. Such computations are forced to be infinite and thus regarded as non-halting computations which are therefore ignored. Only the finite computations correspond to those of the encoded RAM.

A crucial observation from [22] is that to be able to force wrong computation to be infinite, the CCS$_!$ encoding of a given RAM can, during evolution, move from a state which may terminate (i.e. weakly terminating state) into one that cannot terminate (i.e., strongly non-terminating state). In other words, the encoding does not preserve (weak) termination during evolution. It is worth pointing that since RAMs are deterministic machines, their faithful encoding in CCS given in [21] does preserve weak termination during evolution. A legitimate question is therefore: What can be encoded with termination-preserving CCS$_!$ processes?

In this chapter, we shall investigate the expressiveness of CCS$_!$ processes which indeed preserve (weak) termination during evolution by studying the ex-

istence of faithful encodings into CCS$_!$ of models of computability *strictly less* expressive than Turing Machines. This way we disallow the technique used in [20] to unfaithfully encode RAMs.

Notice that a sequence of actions s (over a finite set of actions) performed by a process P specifies a sequence of interactions with P's environment. For example, $s = a^n.\bar{b}^n$ can be used to specify that if P is input n a's by environment then P can output n b's to the environment. We therefore find it natural to study the expressiveness of processes w.r.t. sequences (or patterns) of interactions (languages) they can describe. In particular we shall study the expressiveness of CCS$_!$ w.r.t. the existence of termination-preserving encodings of grammars of Types 1 (Context Sensitive grammars), 2 (Context Free grammars) and 3 (Regular grammars) in the Chomsky Hierarchy whose expressiveness corresponds to (non-deterministic) Linear-bounded, Pushdown and Finite-State Automata, respectively. As elaborated later in the related work, similar characterizations are stated in the Caucal hierarchy of transition systems for other process algebras [19].

It is worth noticing that by using the non termination-preserving encoding of RAM's in [21] we can encode Type 0 grammars (which correspond to Turing Machines) in CCS$_!$.

Remark 3.1.1 *In this chapter we focus our study on the summation-free CCS$_!$ fragment, in fact, the term CCS$_!$ will refer to the summation-free fragment, except for Section 3.3. Although the work [22] considers guarded-summation for CCS$_!$, the results about the encodability of RAMs our work builds on can straightforwardly be adapted to our summation-free CCS$_!$ fragment. (See Section 3.3).*

Remark 3.1.2 *In principle the mere fact that a computation model fails to generate some particular language may not give us a definite answer about its computation power. For a trivial example, consider a model similar to Turing Machines except that the machines always print the symbol a on the first cell of the output tape. The model is essentially Turing powerful but fails to generate b. Nevertheless, our restriction to termination-preserving processes is a natural one, much like restricting non-deterministic models to deterministic ones, meant to rule out unfaithful encodings of the kind used in [22]. As matter of fact, Type 0 grammars*

can be encoded by using the termination-preserving encoding of RAMs in CCS [21].

3.1.1 Contributions.

For simplicity, let us use $\text{CCS}_!^{-\omega}$ to denote the set of $\text{CCS}_!$ processes which preserve weak termination during evolution as described above. We show that $\text{CCS}_!^{-\omega}$ can generate all the regular languages by providing a language preserving encoding of Regular grammars into $\text{CCS}_!^{-\omega}$ (Section 3.5.1) . We also prove that $\text{CCS}_!^{-\omega}$ processes can generate languages which cannot be generated by any Regular grammar by showing a particular process. Our main contribution is to show that it is *impossible* to provide language preserving encodings from Context-Free grammars into $\text{CCS}_!^{-\omega}$, it is done by showing that there is particular context-free languages which can not be generated by any $\text{CCS}_!^{-\omega}$ process, a family of $\text{CCS}_!$ processes, namely $trios - processes$, is defined for technical reasons (Section 3.5.2). Conversely, we also show that $\text{CCS}_!^{-\omega}$ can generate languages which cannot be generated by any Context-free grammar by showing a particular process (Section 3.5.2). We conclude our classification by conjecturing that all languages generated by $\text{CCS}_!^{-\omega}$ processes are context sensitive (Section 3.5.3). These results are summarized in Fig. 3.1. Additionally, we prove the undecidability of the set of $\text{CCS}_!^{-\omega}$ processes.

Outline of this chapter. The rest of this chapter is organized as follows. In Section 3.2 we discuss how the non-termination us used in $\text{CCS}_!$ and introduce formally the notion of termination-preserving process. In Section 3.3 we show the choice operator is not necessary for the Turing expressiveness of $\text{CCS}_!$. In Section 3.4 we prove that the set of $\text{CCS}_!$ terminating-processes is undecidable. In Section 3.5, we present the main results of this chapter, which are summarized in Fig. 3.1. Finally, in Section 3.6 we conclude by summarising this chapter and discussing some related work .

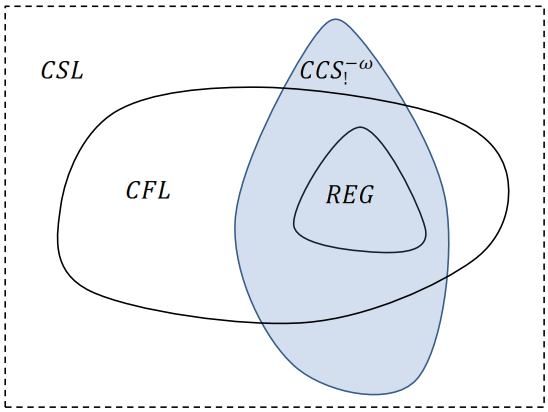


Figure 3.1: Termination-Preserving CCS$_!$ Processes (CCS$_!^{-\omega}$) in the Chomsky Hierarchy: the shaded area represents the set of CCS$_!^{-\omega}$ processes, an area with continuous border and name C represents the set of processes whose languages is C, the area with dotted border represents the conjecture that the CCS$_!^{-\omega}$ processes are context sensitive.

3.2 The Role of Strong Non-Termination

In this section we shall single out the fundamental non-deterministic strategy for the Turing-expressiveness of CCS$_!$.

First, we need to define strongly (weakly) (non-)termination. As we shall see below (strong) non-termination plays a fundamental role in the expressiveness of CCS$_!$. We borrow the following terminology from rewriting systems:

Definition 3.2.1 (Termination) *We say that a process P is* (weakly) terminating *(or that it can terminate) if and only if there exists a sequence s such that P generates s. We say that P is* (strongly) non-terminating*, or that it* cannot terminate *if and only if P cannot generate any sequence.*

Busi et al. in [22] show the Turing-expressiveness of CCS$_!$, by providing a CCS$_!$ encoding $[\![\cdot]\!]$ of RAMs [63]. The encoding is said to be *unfaithful* (or non-deterministic) in the following sense: Given M, during evolution $[\![M]\!]$ may make a transition, by performing a τ action, from a weakly terminating state (process) into a state which do not correspond to any configuration of M. Nevertheless such states are strongly non-terminating processes. Therefore, they may be thought

of as being configurations which cannot lead to a halting configuration. Consequently, the encoding $[\![M]\!]$ does not *preserve (weak) termination* during evolution.

Now rather than giving the full encoding of RAMs in CCS$_!$, let us use a much simpler example which uses the same technique in [22]. Below we encode a typical context sensitive language in CCS$_!$.

Example 3.2.1 *Consider the following processes:*

$$\begin{aligned} P &= (\nu\, k_1, k_2, k_3, u_b, u_c)(\ \overline{k_1} \mid \overline{k_2} \mid Q_a \mid Q_b \mid Q_c) \\ Q_a &= !k_1.a.(\overline{k_1} \mid \overline{k_3} \mid \overline{u_b} \mid \overline{u_c}) \\ Q_b &= k_1.!k_3.k_2.u_b.b.\overline{k_2} \\ Q_c &= k_2.(!u_c.c \mid u_b.DIV) \end{aligned}$$

where $DIV = !\tau$. It can be verified that $L(P) = \{a^n b^n c^n\}$. Intuitively, in the process P above, Q_a performs (a sequence of actions) a^n for an arbitrary number n (and also produces n u_b's). Then Q_b performs b^m for an arbitrary number $m \leq n$ and each time it produces b it consumes a u_b. Finally, Q_c performs c^n and diverges if $m < n$ by checking if there are u_b's that were not consumed.

The Power of Non-Termination. Let us underline the role of strong non-termination in Example 3.2.1. Consider a run

$$P \stackrel{a^n b^m}{\Longrightarrow} \ldots$$

Observe that the name u_b is used in Q_c to test if $m < n$, by checking whether some u_b were left after generating b^m. If $m < n$, the non-terminating process DIV is triggered and the extended run takes the form

$$P \stackrel{a^n b^m c^n}{\Longrightarrow} \stackrel{\tau}{\longrightarrow} \stackrel{\tau}{\longrightarrow} \ldots$$

. Hence the sequence $a^n b^m c^n$ arising from this run (with $m < n$) is therefore not included in $L(P)$.

The tau move. It is crucial to observe that there is a τ transition arising from the moment in which $\overline{k_2}$ chooses to synchronise with Q_c to start performing the c actions. One can verify that if $m < n$ then the process just before that τ transition is weakly terminating while the one just after is strongly non-terminating.

Formally the class of termination-preserving processes is defined as follows.

Definition 3.2.2 (Termination Preservation) *A process P is said to be weakly termination-preserving if and only if whenever $P \stackrel{s}{\Longrightarrow} Q \stackrel{\tau}{\longrightarrow} R$:*

- *if Q is weakly terminating then R is weakly terminating.*

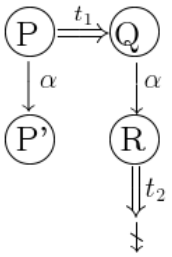
We use $CCS_!^{-\omega}$ to denote the set of $CCS_!$ processes that are termination-preserving. (Notice that $CCS_!^{-\omega}$ does not denote a sub-calculus of $CCS_!$; it is a semantically, not syntactically, defined set of processes)

One may wonder why only τ actions are not allowed in Definition 3.2.2 when moving from a weakly terminating state into a strongly non-terminating one. The next proposition answers to this.

Proposition 3.2.1 *For every $P, P', \alpha \neq \tau$ if $P \stackrel{\alpha}{\longrightarrow} P'$ and P is weakly terminating then P' must be weakly terminating.*

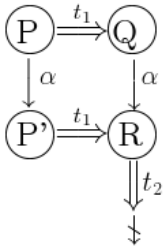
Proof. As a means of contradiction let P' be a strongly non-terminating process such that $P \stackrel{\alpha}{\longrightarrow} P'$ where $\alpha \neq \tau$. Let γ be an arbitrary maximal sequence of transitions from P. Since $P \stackrel{\alpha}{\longrightarrow} P'$, the action α will be performed in γ as a visible action or in a synchronisation with its complementary action $\bar{\alpha}$. In the synchronisation case, one can verify that there exists another maximal sequence γ' identical to γ except that in γ', α and $\bar{\alpha}$ appear as visible actions instead of their corresponding synchronisation. Therefore, there exists a sequence $P \stackrel{t_1}{\Longrightarrow} Q \stackrel{\alpha}{\longrightarrow} R \stackrel{t_2}{\Longrightarrow} \not\rightarrow$ (Fig. 3.2). From $P \stackrel{t_1}{\Longrightarrow} Q \stackrel{\alpha}{\longrightarrow} R$ and $P \stackrel{\alpha}{\longrightarrow} P'$, we can show that $P \stackrel{\alpha}{\longrightarrow} P' \stackrel{t_1}{\Longrightarrow} R \stackrel{t_2}{\Longrightarrow} \not\rightarrow$ (Fig. 3.3) thus contradicting the assumption that P' is a strongly non-terminating process. □

We conclude this section with a proposition which relates preservation of termination and the language of a process.



where $\alpha \neq \tau$

Figure 3.2: Alternative evolutions of P involving α



where $\alpha \neq \tau$

Figure 3.3: Confluence from P to R

Proposition 3.2.2 *Suppose that P is terminating-preserving and that $L(P) \neq \emptyset$. For every Q, if $P \stackrel{s}{\Longrightarrow} Q$ then $\exists s'$ such that $s.s' \in L(P)$.*

Proof. Let Q an arbitrary process such that $P \stackrel{s}{\Longrightarrow} Q$. Since $L(P) \neq \emptyset$ then P is weakly terminating. From Definition 3.2.2 and Proposition 3.2.1 it follows that Q is weakly terminating. Hence there exists a sequence s' such that $P \stackrel{s}{\Longrightarrow} Q \stackrel{s'}{\Longrightarrow} R \not\rightarrow$ and thus from Definition 2.4.6 we have $s.s' \in L(P)$ as wanted. □

3.3 $CCS_!$ without choice

In this section we show that the encoding proposed by Busi et al. in [22] of RAMs (Random Accesss machines) into $CCS_!$ with guarded summation can be easily adapted to the summation-free fragment.

First, let us recall the encoding of $RAMs$ into CCS$_!$ with guarded summation proposed in [22]. Hence the encoding of the instructions and of a register r_j

storing the value c_j is:

$$\begin{array}{ll}
[\![(i : Succ(r_j)]\!] = & !p_i.(\overline{inc_j} \mid inc.\overline{p_{i+1}}) \\
[\![(i : DecJump(r_j, s)]\!] = & !p_i.(dec_j \mid (dec.\overline{p_{i+1}} + zero.\overline{ps})) \\
[\![(r_j : c_j)]\!] = & \overline{nr_j} \mid \\
& !nr_j.(\nu\, m, i, d, u)(outm \mid !m.(inc_j.\bar{i} + dec_j.\bar{d}) \mid \\
& !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.(\overline{m} \mid \overline{dec})) \mid \\
& d.(\overline{zero} \mid u.DIV \mid \overline{nr_j}) \mid \\
& \textstyle\prod_{c_j}(\overline{u} \mid d.u.(\overline{m} \mid \overline{dec})))
\end{array}$$

where DIV is a process able to activate an infinite observable computation, for instance $\overline{w'} \mid !w'.\overline{w'}$.

Along the computation, some "garbage process" can appear:

$$G_j : (\nu\, m, i, d, u)(!m.(inc_j.\bar{i} + dec_j.\bar{d}) \mid !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.\overline{dec}) \mid u.DIV)$$

Definition 3.3.1 *[22] Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the configuration $(i, c_1, \ldots, c_n)$ of R, we define*

$$\begin{array}{rl}
[\![(i, c_1 \ldots c_n)]\!]_R = & (\nu\, p_1 \ldots p_m, nr_1, inc_1, dec_1 \ldots nr_n, inc_n, dec_n, inc, dec, zero) \\
& (\overline{p_1} \mid [\![(1 : I_1)]\!] \mid \ \ldots \ \mid [\![(m : I_m)]\!] \mid \prod_{i \in TI} p_i.\overline{w} \mid \\
& [\![r_1 = c_1]\!] \mid \ \ldots \ \mid [\![r_n = c_n]\!] \mid \prod_{k_1} G_1 \mid \ \ldots \ \mid \prod_{k_n} G_n)
\end{array}$$

where the modelling of program instructions $[\![(i : I_i)]\!]$, the modelling of registers $[\![r_j = c_j]\!]$, the set of terminating indexes TI, and the garbage $G_1, \ldots, G_n$ have been defined above, and $k_1 \ldots k_n$ are natural numbers.

We recall the following theorem from [22] where the correctness of the encoding is established.

Theorem 3.3.1 *[22] Let R be a RAM with program $(1 : I_1), \ldots, (m : I_m)$ and state $(i, c_1, \ldots, c_n)$ and let the process P be in $[\![(i, c_1, \ldots, c_n]\!]_R$. Then*

$(i, c_1, \ldots, c_n)$ terminates if and only if P converges. Moreover P converges if and only if $P \approx \tau.P + \overline{w}$.

This proves that convergence and weak bisimulation are undecidable for $CCS_!$ with guarded summation.

If we consider the CCS! fragment without choice it is still possible to adapt the encoding proposed in [22] to our language:

$$
\begin{array}{ll}
[\![(i : Succ(r_j)]\!]_m = & [\![(i : Succ(r_j)]\!] \\
[\![(i : DecJump(r_j, s)]\!]_m = & !p_i.(dec_j \mid (\langle\!\langle dec.\overline{p_{i+1}} + zero.\overline{ps}\rangle\!\rangle)) \\
\langle\!\langle dec.\overline{p_{i+1}} + zero.\overline{ps}\rangle\!\rangle := & (\nu\, lv_d, lv_z)(dec.(\overline{p_{i+1}} \mid \overline{lv_z} \mid lv_d.DIV) \mid \\
 & zero.(\overline{ps} \mid \overline{lv_d} \mid lv_z.DIV)) \\
[\![(r_j : c_j)]\!]_m = & \overline{nr_j} \mid \\
 & !nr_j.(\nu\, m, i, d, u)(\overline{m} \mid !m.\langle\!\langle (inc_j.\overline{i} + dec_j.\overline{d})\rangle\!\rangle \mid \\
 & !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.(\overline{m} \mid \overline{dec})) \mid \\
 & d.(\overline{zero} \mid u.DIV \mid \overline{nr_j}) \mid \\
 & \textstyle\prod_{c_j}(\overline{u} \mid d.u.(\overline{m} \mid \overline{dec}))) \\
\langle\!\langle inc_j.\overline{i} + dec_j.\overline{d}\rangle\!\rangle := & (\nu\, lv_d, lv_i)(inc_j.(\overline{i} \mid \overline{lv_d} \mid lv_i.DIV) \mid \\
 & dec_j.(\overline{d} \mid \overline{lv_i} \mid lv_d.DIV))
\end{array}
$$

Where DIV is a process able to activate an infinite observable computation, in particular we define DIV as $\overline{w'} \mid !w'.\overline{w'}$.

The translation of choice in both $\langle\!\langle dec.\overline{p_{i+1}} + zero.\overline{ps}\rangle\!\rangle$ and $\langle\!\langle inc_j.\overline{i} + dec_j.\overline{d}\rangle\!\rangle$ introduces more computations which do not follow the expected behaviour of the modeled RAM. However these computations are also infinite. Intuitively, once the choice has been done, e.g. inc ($zero$) or dec can still participate in the computation as they are in parallel, in this case the local variables lv_d, lv_i(lv_z) trigger divergence, ensuring that the computation cannot terminate.

In this way, Definition 3.3.1 and Theorem 3.3.1 can be adapted to $CCS_!$ without choice. Given an initial configuration $(1, 0, \ldots, 0)$ of a RAM, it is possible to provide an encoding $[\![(1, 0, \ldots, 0)]\!]_{R_m}$ similar to the one of the Definition 3.3.1 but using $[\![\cdot]\!]_m$. Clearly, $(i, 0, \ldots, 0)$ terminates if and only if $[\![(1, 0, \ldots, 0)]\!]_{R_m}$ converges and $[\![(1, 0, \ldots, 0)]\!]_{R_m}$ converges if and only if $[\![(1, 0, \ldots, 0)]\!]_{R_m} \approx \tau$ $[\![(1, 0, \ldots, 0)]\!]_{R_m} + \overline{w}$. Therefore convergence and weak bisimulation are unde-

cidable for $CCS_!$ without choice.

From now on the term $CCS_!$ will refer to the $CCS_!$ variant without choice.

3.4 Undecidability results for $CCS_!^{-\omega}$

A relevant question is whether a $CCS_!$ process preserves termination or not. In this section, we prove that this problem is undecidable. Recall that $CCS_!^{-\omega}$ is not a new language but a set of $CCS_!$ processes which satisfy a semantic property: termination-preserving. In Chapter 4, we study the expressive power of syntactic fragments of $CCS_!$.

Lemma 3.4.1 *Let P be a $CCS_!$ process, if $L(P) = \emptyset$ then P is termination-preserving.*

Proof. Let first observe that by definition $L(P) = \emptyset$ iff P is not weakly terminating. As a mean of contradiction, let $L(P) = \emptyset$ and P be non-termination preserving, since P is not termination-preserving then $P \stackrel{s}{\Longrightarrow} Q \stackrel{\tau}{\longrightarrow} R$ such that Q is weakly terminating and R is non-weakly terminating. Therefore Q recognises at least one sequence and consequently also P. As P recognises at least one sequence, $L(P) \neq \emptyset$, a contradiction. □

Notice that the notion of weakly termination and convergence are not the same, as the first one takes into account the visible actions whereas the second one not. For example $!a$ is not weakly terminating but it is convergent, and $(\nu\, a)(\overline{a} \mid !a.\overline{a} \mid b.a)$ is not convergent but it is weakly terminating, in fact $L((\nu\, a)(\overline{a} \mid !a.\overline{a} \mid b.a)) = \{b\}$. However when considering RAMs the two notion are equivalent, indeed we can prove the following:

Lemma 3.4.2 *$[\![M]\!]_{R_m}$ is weakly terminating iff $[\![M]\!]_{R_m}$ is convergent.*

Proof.

First , let us consider the case if $[\![M]\!]_{R_m}$ is convergent: from the encoding construction, then there exists only one maximal finite τ-sequence from $[\![M]\!]_{R_m}$. When this τ computation finishes a terminating instruction of the form $p_i.\overline{w}$ has been activated. Therefore at the end of the τ computation $\overline{w}$ is visible. There

is no further actions later on as $\overline{w}$ can not synchronise and hence a τ-action can not arise therefore DIV is not active. As DIV is not active $\overline{w'}$ and w' are not visible and as only one terminating instruction can be activated there is no other $\overline{w}$ visible. Therefore $[\![M]\!]_{R_m} \stackrel{\overline{w}}{\Longrightarrow} Q \stackrel{\alpha}{\not\Longrightarrow}$ for any $\alpha \in Act$, there is no other finite maximal sequence of visible actions from a finite maximal sequence of transitions of $[\![M]\!]_{R_m}$ as two $\overline{w}$ actions can not appear in any sequence and $\overline{w'}$ and w only can appear in infinite τ-sequences. Notice that $\overline{w}$, $\overline{w'}$ and w' are the only visible actions in $[\![M]\!]_{R_m}$. We conclude that if $[\![M]\!]_{R_m}$ is convergent then $L([\![M]\!]_{R_m}) = \{w\}$ hence $[\![M]\!]_{R_m}$ is weakly terminating.

Conversely if $[\![M]\!]_{R_m}$ is weakly terminating, there is at least one finite maximal sequence of visible actions generated from a finite maximal sequence of transitions of $[\![M]\!]_{R_m}$. In fact a finite maximal sequence of transitions from $[\![M]\!]_{R_m}$ only can generate sequences in which one occurrence of $\overline{w}$ is present and it is the unique visible action. Let us understand why other sequences from a finite maximal sequence of transitions from $[\![M]\!]_{R_m}$ can not be generated:

1. Let us consider a sequence with only τ actions: in this case $[\![M]\!]_{R_m}$ would be convergent but $[\![M]\!]_{R_m}$ would exhibit $\overline{w}$ as well . Thus a maximal finite sequence with only τ actions cannot exist.

2. Let us consider a sequence of the form $s.\overline{w'}.s'$ or $s.w'.s'$ where s and $s' \in \mathcal{L}^*$: if w' or $\overline{w'}$ appear in the sequence then DIV is activated therefore there is a process is of the form $Q \mid !w'.\overline{w'}$ in the computation, hence the process is non-weakly terminating and a maximal finite sequence with w' or $\overline{w'}$ cannot exist.

3. Let us consider a sequence of the form $s.\overline{w}.s'$ where s and $s' \in \mathcal{L}^*$ such that $|s'| \geq 1$: from item 1 and 2, it is left to study sequences made of only $\overline{w}$ actions. But from the encoding construction only one occurrence of $\overline{w}$ is visible along any sequence as only one terminating instruction, of the form $p_i.\overline{w}$, can be activated. Therefore a sequence with at least two $\overline{w}$ actions cannot exist.

Therefore if $[\![M]\!]_{R_m}$ is weakly terminating then $L([\![M]\!]_{R_m}) = \{\overline{w}\}$.

Hence we conclude that $[\![M]\!]_{R_m}$ is weakly terminating iff $[\![M]\!]_{R_m}$ is convergent.

□

Lemmas 3.4.1 and 3.4.2 are necessary to prove the following undecidability result:

Theorem 3.4.1 *The property of being termination-preserving is undecidable for* $CCS_!$.

Proof. Let us consider the encoding $[\![\cdot]\!]_{R_m}$ from RAM into $CCS_!$[1], it suffices to prove that a RAM M halts iff $(\nu\, a)(\overline{a} \mid\ a.[\![M]\!]_{R_m} \mid a.DIV)$ is not terminating preserving.

First, we prove that $(\nu\, a)(\overline{a} \mid\ a.[\![M]\!]_{R_m} \mid a.DIV)$ is not termination preserving iff $[\![M]\!]_{R_m}$ is weakly terminating.

If $[\![M]\!]_{R_m}$ is non-weakly terminating (hence $L([\![M]\!]_{R_m}) = \emptyset$) then $(\nu\, a)(\overline{a} \mid a.[\![M]\!]_{R_m} \mid a.DIV)$ is non-weakly terminating ($L((\nu\, a)(\overline{a} \mid a.[\![M]\!]_{R_m} \mid a.DIV)) = \emptyset$) and therefore by Lemma 3.4.1 $(\nu\, a)(\overline{a} \mid\ a.[\![M]\!]_{R_m} \mid\ a.DIV)$ is termination-preserving.

On the other hand, if $[\![M]\!]_{R_m}$ is weakly terminating, then $(\nu\, a)(\overline{a} \mid a.[\![M]\!]_{R_m} \mid a.DIV)$ is weakly terminating but notice that $(\nu\, a)\ (\overline{a} \mid a.[\![M]\!]_{R_m} \mid a.DIV) \xrightarrow{\tau} (\nu\, a)(a.[\![M]\!]_{R_m} \mid DIV)$ where $(\nu\, a)\ (a.[\![M]\!]_{R_m} \mid DIV)$ is non-weakly terminating therefore $(\nu\, a)(\overline{a} \mid\ a.[\![M]\!]_{R_m} \mid\ a.DIV)$ is not termination preserving .

We have that $[\![M]\!]_{R_m}$ is weakly terminating iff $[\![M]\!]_{R_m}$ is convergent by Lemma 3.4.2 and $(\nu\, a)(\overline{a} \mid\ a.[\![M]\!]_{R_m} \mid\ a.DIV)$ is not termination preserving iff $[\![M]\!]_{R_m}$ is weakly terminating. Therefore $[\![M]\!]_{R_m}$ is convergent iff $(\nu\, a)(\overline{a} \mid a.[\![M]\!]_{R_m} \mid\ a.DIV)$ is not termination preserving. Finally as M halts iff $[\![M]\!]_{R_m}$ is convergent, we conclude that M halts iff $(\nu\, a)(\overline{a} \mid\ a.[\![M]\!]_{R_m} \mid\ a.DIV)$ is not termination preserving. □

[1]Recall this is the $CCS_!$ variant without choice

3.5 $CCS_!$ and the Chomsky Hierarchy

In this section we study the expressiveness of termination-preserving CCS! processes in the Chomsky hierarchy. Recall that, in a strictly decreasing expressive order, Types 0, 1, 2 and 3 in the Chomsky hierarchy correspond, respectively, to unrestricted-grammars (Turing Machines), Context-Sensitive Grammars (Non-Deterministic Linear Bounded Automata), Context-Free Grammars (Non-Deterministic PushDown Automata), and Regular Grammars (Finite State Automata).

We assume that the reader is familiar with the notions and notations of formal grammars. A grammar is a quadruple $G = (\Sigma, N, S, P)$ where Σ are the terminal symbols, N the non-terminals, S the initial symbol, P the set of production rules. The language of (or generated by) a formal grammar G, denoted as $L(G)$, is defined as all those strings in Σ^* that can be generated by starting with the start symbol S and then applying the production rules in P until no more non-terminal symbols are present.

Notation 3.5.1 *In the remainder of this chapter we shall write the summation $P + Q$ as an abbreviation of the process $(\nu\, u)(\overline{u} \mid u.P \mid u.Q)$.*

3.5.1 Encoding Regular Languages

Regular Languages (REG) are those generated by grammars whose production rules can only be of the form $A \rightarrow a$ or $A \rightarrow a.B$. They can be alternatively characterised as those recognised by regular expressions which are given by the following syntax:

$$e = \emptyset \mid \epsilon \mid a \mid e_1 + e_2 \mid e_1.e_2 \mid e^*$$

where a is a terminal symbol.

Definition 3.5.1 *Given a regular expression e, the set of terminal symbols of e is defined inductively as follows: $Symb(\emptyset) = \emptyset$, $Symb(\epsilon) = \emptyset$, $Symb(a) = \{a\}$, $Symb(e_1 + e_2) = Symb(e_1) \cup Symb(e_2)$, $Symb(e_1.e_2) = Symb(e_1) \cup Symb(e_2)$, $Symb(e^*) = Symb(e)$.*

$$
\begin{array}{ll}
[\![\emptyset]\!]_m = & DIV \\
[\![\epsilon]\!]_m = & \overline{m} \\
[\![a]\!]_m = & a.\overline{m} \\
[\![e_1 + e_2]\!]_m = & \begin{cases} [\![e_1]\!]_m & \text{if } L(e_2) = \emptyset \\ [\![e_2]\!]_m & \text{if } L(e_1) = \emptyset \\ [\![e_1]\!]_m + [\![e_2]\!]_m & \text{otherwise} \end{cases} \\
[\![e_1.e_2]\!]_m = & \begin{cases} DIV & \text{if } L(e_1) = \emptyset \text{ or } L(e_2) = \emptyset \\ (\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m) \text{ with } m_1 \notin Symb(e_1) & \text{otherwise} \end{cases} \\
[\![e^*]\!]_m = & \begin{cases} \overline{m} & \text{if } L(e) = \emptyset \\ (\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \text{ with } m' \notin Symb(e) & \text{otherwise} \end{cases}
\end{array}
$$

where $DIV = !\tau$.

Figure 3.4: Encoding of regular expressions

Definition 3.5.2 *Given a regular expression e, we define $[\![e]\!]$ as the $CCS_!$ process $(\nu\, m)$ $([\![e]\!]_m)$ where $[\![e]\!]_m$, with $m \notin Symb(e)$, is inductively defined as in Figure 3.4.*

Remark 3.5.1 *The conditionals on language emptiness in Definition 3.5.2 are needed to make sure that the encoding of regular expressions always produce termination-preserving processes. To see this consider the case $a + \emptyset$. Notice that while $[\![a]\!] = a$ and $[\![\emptyset]\!] = DIV$ are termination-preserving, $a + DIV$ is not. Hence $[\![e_1 + e_2]\!]$ cannot be defined as $[\![e_1]\!] + [\![e_2]\!]$. Since the emptiness problem is decidable for regular expressions, it is clear that given e, $[\![e]\!]$ can be effectively constructed.*

Proposition 3.5.1 *Given a regular expression e and $[\![e]\!]_m$ as in Figure 3.4 with $m \notin Symb(e)$. Then $L([\![e]\!]_m) = \{s.\overline{m} | s \in L(e)\}$ and $[\![e]\!]_m$ is termination preserving.*

Proof. The proof will proceed by induction on the structure of regular expressions.

- if $e = \emptyset$: From Figure 3.4 it is straightforward that $L([\![\emptyset]\!]_m) = \emptyset$ and $[\![\emptyset]\!]_m = DIV$ is trivially termination-preserving.

- if $e = \epsilon$: From Figure 3.4 it is straightforward that $L([\![\epsilon]\!]_m) = \overline{m}$ and $[\![\epsilon]\!]_m = \overline{m}$ is trivially termination-preserving.

- if $e = a$: From Figure 3.4 it is straightforward that $L([\![a]\!]_m) = a.\overline{m}$ and $[\![a]\!]_m = a.\overline{m}$ is trivially termination-preserving.

- if $e = e_1 + e_2$:

  - if $L(e_2) = \emptyset$: $L([\![e_1 + e_2]\!]_m) = L([\![e_1]\!]_m)$ from Figure 3.4. By inductive hypothesis, $L([\![e_1]\!]_m) = \{s.\overline{m} | s \in L(e_1)\}$ and $L([\![e_1]\!]_m) = \{s.\overline{m} | s \in L(e_1 + e_2)\}$ as $L(e_1 + e_2) = L(e_1)$ when $L(e_2) = \emptyset$. As $[\![e_1 + e_2]\!]_m = [\![e_1]\!]_m$ and by inductive hypothesis $[\![e_1]\!]_m$ is termination-preserving then $[\![e_1 + e_2]\!]_m$ as well.

  - if $L(e_1) = \emptyset$: $L([\![e_1 + e_2]\!]_m) = L([\![e_2]\!]_m)$ from Figure 3.4. By inductive hypothesis, $[\![e_2]\!]_m = \{s.\overline{m} | s \in L(e_2)\}$ and $L([\![e_2]\!]_m) = \{s.\overline{m} | s \in L(e_1 + e_2)\}$ as $L(e_1 + e_2) = L(e_2)$ when $L(e_1) = \emptyset$. As $[\![e_1 + e_2]\!]_m = [\![e_2]\!]_m$ and by inductive hypothesis $[\![e_2]\!]_m$ is termination-preserving then $[\![e_1 + e_2]\!]_m$ as well.

  - in other case: $L([\![e_1 + e_2]\!]_m) = L([\![e_1]\!]_m + [\![e_2]\!]_m)$ from Figure 3.4. As $L([\![e_1]\!]_m + [\![e_2]\!]_m) = \{s | s \in L([\![e_1]\!]_m) \ \ or \ \ s \in L([\![e_2]\!]_m)\}$ and by inductive hypothesis $L([\![e_1]\!]_m) = \{s.\overline{m} | \ \ s \in L(e_1)\}$ and $L([\![e_2]\!]_m) = \{s.\overline{m} | s \in L(e_2)\}$, $L([\![e_1 + e_2]\!]_m) = \{s.\overline{m} | \ \ s \in L(e_1) \ \ or \ \ s \in L(e_2)\} = \{s.\overline{m} | \ s \in L(e_1 + e_2)\}$. As $L([\![e_1]\!]_m) \neq 0$ and $L([\![e_2]\!]_m) \neq 0$ and by using inductive hypothesis we know that $[\![e_1]\!]_m$ and $[\![e_2]\!]_m$ are weakly terminating and termination-preserving. From Definitions 2.4.6 and 3.2.2 and Proposition 3.2.1, whenever $[\![e_1]\!]_m \stackrel{s}{\Longrightarrow} P$, P is weakly-terminating and whenever $[\![e_2]\!]_m \stackrel{s}{\Longrightarrow} P'$, P' is weakly-terminating. From Notation 3.5.1, either $[\![e_1]\!]_m + [\![e_2]\!]_m = (\nu\, u)(\overline{u} \mid u.[\![e_1]\!]_m \mid u.[\![e_2]\!]_m) \stackrel{\tau}{\longrightarrow} (\nu\, u)([\![e_1]\!]_m \mid u.[\![e_2]\!]_m)$ or $[\![e_1]\!]_m + [\![e_2]\!]_m \stackrel{\tau}{\longrightarrow} (\nu\, u)(u.[\![e_1]\!]_m \mid [\![e_2]\!]_m)$. As there is no occurrence of $\overline{u}$ in $(\nu\, u)([\![e_1]\!]_m \mid u.[\![e_2]\!]_m)$ and $(\nu\, u)(u.[\![e_1]\!]_m \mid [\![e_2]\!]_m)$, whenever $[\![e_1]\!]_m + [\![e_2]\!]_m \stackrel{s}{\Longrightarrow} Q$, Q is either of the form $(\nu\, u)(P \mid u.[\![e_2]\!]_m)$ or $(\nu\, u)(u.[\![e_1]\!]_m \mid P')$ where $(\nu\, u)(P \mid u.[\![e_2]\!]_m)$ and $(\nu\, u)(u.[\![e_1]\!]_m \mid P')$ are weakly terminating as P and P' are weakly terminating because

$[\![e_1]\!]_m \stackrel{s'}{\Longrightarrow} P$ and $[\![e_2]\!]_m \stackrel{s''}{\Longrightarrow} P'$. From Definition 3.2.2 $[\![e_1]\!]_m + [\![e_2]\!]_m$ is termination-preserving.

- if $e = e_1.e_2$: $L([\![e_1.e_2]\!]_m) = L((\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m))$ with $m_1 \notin Symb(e_1)$ from Figure 3.4.

  - if $L(e_1) = \emptyset$ or $L(e_2) = \emptyset$: from the encoding in Figure 3.4 , $[\![e_1.e_2]\!]_m = DIV$, therefore $L([\![e_1.e_2]\!]_m) = \emptyset$. It is straightforward $[\![e_1.e_2]\!]_m$ is termination-preserving.

  - If $L(e_1) \neq \emptyset$ and $L(e_2) \neq \emptyset$: By inductive hypothesis a sequence s is in $L(e_1)$ iff there exists Q such that $[\![e_1]\!]_m \stackrel{s.\overline{m}}{\Longrightarrow} Q$ and $Q \stackrel{\alpha}{\not\longrightarrow}$ for any $\alpha \in Act$ and a sequence s' is in $L(e_2)$ iff there exists Q' such that $[\![e_2]\!]_m \stackrel{s'.\overline{m}}{\Longrightarrow} Q'$ and $Q' \stackrel{\alpha}{\not\longrightarrow}$ for any $\alpha \in Act$. Therefore for any s, s' in $L(e_1)$ and $L(e_2)$ respectively there exist Q and Q' such that $[\![e_1.e_2]\!]_m = (\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m) \stackrel{s.s'.m}{\Longrightarrow} (\nu\, m_1)Q \mid Q'$. where $Q \stackrel{\alpha}{\not\longrightarrow}$ and $Q' \stackrel{\alpha}{\not\longrightarrow}$ and therefore $(\nu\, m_1)Q \mid Q' \stackrel{\alpha}{\not\longrightarrow}$ for any $\alpha \in Act$. As a consequence, $\{s.\overline{m} | s \in L(e_1.e_2)\} \subseteq L([\![e_1.e_2]\!]_m)$. As for the other direction, i.e. $L([\![e_1.e_2]\!]_m) \subseteq \{s.\overline{m} | s \in L(e_1.e_2)\}$, it comes from the fact $[\![e_1.e_2]\!]_m = (\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m)$ cannot generate sequences apart from $\{s.\overline{m} | s \in L(e_1.e_2)\}$. It is because the restricted name m_1 allows to control that the sequences from $[\![e_1]\!]_{m_1}$ precede the sequences from $[\![e_2]\!]_m$. By inductive hypothesis $[\![e_1]\!]_{m_1}$ and $[\![e_2]\!]_m$ are termination-preserving and weakly terminating. By Definitions 2.4.6 and 3.2.2 and Proposition 3.2.1 whenever $[\![e_1]\!]_m \stackrel{s}{\Longrightarrow} P$, P is weakly-terminating and whenever $[\![e_2]\!]_m \stackrel{s}{\Longrightarrow} P'$, P' is weakly-terminating. We have whenever $(\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m) \stackrel{s}{\Longrightarrow} Q$, Q is either of the form $(\nu\, m_1)(P \mid m_1.[\![e_2]\!]_m)$ or $(\nu\, m_1)(P \mid P')$ and $[\![e_1]\!]_{m_1} \stackrel{s}{\Longrightarrow} P$ and $[\![e_2]\!]_m \stackrel{s}{\Longrightarrow} P'$. $(\nu\, m_1)(P \mid P')$ is weakly terminating as P and P' are weakly terminating and there is no possible synchronisation between them which can arise divergent behaviour, $(\nu\, m_1)(P \mid m_1.[\![e_2]\!]_m)$ is weakly terminating as P and $[\![e_2]\!]_m$ are weakly terminating and the only synchronisation is on m_1 which cannot arise divergent behaviour. As whenever $(\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m) \stackrel{s}{\Longrightarrow} Q$, Q is weakly

terminating, by using Definition 3.2.2 we have that $L([\![e_1.e_2]\!]_m) = (\nu\, m_1)([\![e_1]\!]_{m_1} \mid m_1.[\![e_2]\!]_m)$ is termination-preserving.

- if $e = e_1^*$:
  - if $L(e_1) = \emptyset$: it is trivial (similar to the case if $e = \epsilon$).
  - if $L(e_1) \neq \emptyset$: then $L(e_1^*) = \bigcup_{i\geq 0} L(e_1^i)$ and

$$e_1^i = \begin{cases} \epsilon & \text{if } i = 0 \\ e_1.e_1^{i-1} & \text{otherwise} \end{cases}$$

By inductive hypothesis we have that $L([\![e_1]\!]_m) = \{s.\overline{m} | s \in L(e_1\}$. We will first prove that $\{s.\overline{m}|s \in L(e_1^*)\} \subseteq L([\![e_1^*]\!]_m)$. Let $s \in L(e_1^*)$ by definition s is either ϵ or $s \in L(e_1^n) = L(\underbrace{e_1.\ldots.e_1}_{\text{i times}})$. For the case $s = \epsilon$, $(\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \xrightarrow{\tau} (\nu\, m')(!m'.[\![e]\!]_{m'} \mid \overline{m}) \xrightarrow{\overline{m}} (\nu\, m')(!m'.[\![e]\!]_{m'}) \not\xrightarrow{\alpha}$ for any $\alpha \in Act$. For the case $s = s_1.s_2.\ldots.s_n$, where $s_i \in L(e_1)$. By inductive hypothesis, $[\![e]\!]_{m'} \stackrel{s_i.\overline{m'}}{\Longrightarrow} Q$ where $Q \not\xrightarrow{\alpha}$ for any $\alpha \in Act$. Therefore, $(\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \xrightarrow{\tau} (\nu\, m')([\![e]\!]_{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \stackrel{s_1}{\Longrightarrow} (\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \xrightarrow{\tau} (\nu\, m')([\![e]\!]_{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \stackrel{s_2}{\Longrightarrow} (\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \ldots \stackrel{s_n}{\Longrightarrow} (\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \xrightarrow{\tau} (\nu\, m')(!m'.[\![e]\!]_{m'} \mid \overline{m}) \xrightarrow{\overline{m}} \xrightarrow{\tau} (\nu\, m')(!m'.[\![e]\!]_{m'}) \not\xrightarrow{\alpha}$ for any $\alpha \in Act$.

As for $L([\![e_1^*]\!]_m) \subseteq \{s.\overline{m}|s \in L(e_1^*)\}$. Any sequence from $(\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m})$ is controlled by the synchronisation on the bound name m':

  - Synchronisation of $\overline{m'}$ with $m'.\overline{m}$: we have that $(\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \xrightarrow{\tau} (\nu\, m')(!m'.[\![e]\!]_{m'} \mid \overline{m})$, in this case the sequence is $\overline{m'}$.
  - Synchronisation of $\overline{m'}$ with $!m'.[\![e]\!]_{m'}$: we have that $(\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \xrightarrow{\tau} (\nu\, m')[\![e]\!]_{m'} \mid !m'.[\![e]\!]_{m'} \mid m'\overline{m}) \stackrel{s}{\Longrightarrow} (\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m})$ where $s \in L(e_1)$, in this case the sequence is a concatenation of sequences of $L(e_1)$.

The concatenation is completed by a synchronisation of $\overline{m'}$ with $m'.\overline{m}$ and the generation of $\overline{m}$, i.e. the sequence generated is of the form $s_1.s_2.\ldots.s_n.m$ where $s_i \in L(e_1)$.

By inductive hypothesis $[\![e_1]\!]_m$ is termination preserving and weakly terminating (as $L(e_1) \neq \emptyset$). As the synchronisations on m' does not introduce divergent behaviour, whenever $(\nu\, m')(\overline{m'} \mid !m'.[\![e]\!]_{m'} \mid m'.\overline{m}) \stackrel{s}{\Longrightarrow} Q$, Q is weakly terminating. Finally by Definition 3.2.2 $[\![e_1^*]\!]_m$ is weakly terminating.

□

The following proposition states the correctness of the encoding.

Proposition 3.5.2 *Let $[\![e]\!]$ be as in Definition 3.5.2. We have $L(e) = L([\![e]\!])$ and furthermore $[\![e]\!]$ is termination-preserving.*

Proof.

$L(e) = L([\![e]\!])$ is straightforward from Proposition 3.5.1 and the fact that m is a local name, therefore it is no part of the sequences generated from $[\![e]\!] = (\nu\, m)$ $([\![e]\!]_m)$.

If $L(e) = \emptyset$, by Proposition 3.5.1 $L([\![e]\!]_m) = \emptyset$ and as m does not participate in any action from $[\![e]\!]_m$, $L([\![e]\!]) = \emptyset$. By Lemma 3.4.1 $[\![e]\!]$ is termination-preserving. If $L(e) \neq \emptyset$, $[\![e]\!]$ is termination-preserving as the restriction on m only prevents the action $\overline{m}$, it clearly does not introduce divergent behaviour.

□

From the standard encoding from Type 3 grammars to regular expressions and the above proposition we obtain the following result.

Theorem 3.5.2 *For every Type 3 grammar G, we can construct a termination-preserving $CCS_!$ process P_G such that $L(G) = L(P_G)$.*

Proof. Follows immediately from Proposition 3.5.2 □

The converse of the theorem above does not hold; Type 3 grammars are strictly less expressive.

Theorem 3.5.3 *There exists a termination-preserving CCS! process P such that $L(P)$ is not Type 3.*

Proof. The above statement can be shown by providing a process which generates the typical $a^n b^n$ context-free language. Namely, let us take

$$P = (\nu\, k, u)(\overline{k} \mid !(k.a.(\overline{k} \mid \overline{u})) \mid k.!(u.b)).$$

One can easily verify that P is termination-preserving and that $L(P) = a^n b^n$. □

3.5.2 Impossibility Result: Context Free Languages

Context-Free Languages (CFL) are those generated by Type 2 grammars: grammars where every production is of the form $A \rightarrow \gamma$ where A is a non-terminal symbol and γ is a string consisting of terminals and/or non-terminals.

We have already seen that termination-preserving CCS! process can encode a typical CFL language such as $a^n b^n$. Nevertheless, we shall show that they cannot in general encode Type 2 grammars.

The nesting of restriction processes plays a key role in the following results CCS!.

Definition 3.5.3 *The maximal number of nesting of restrictions $|P|_\nu$ can be inductively given as follows:*

$$|(\nu\, x)P|_\nu = 1 + |P|_\nu \qquad |P \mid Q|_\nu = max(|P|_\nu, |Q|_\nu)$$
$$|\alpha.P|_\nu = |!P|_\nu = |P|_\nu \qquad |0|_\nu = 0$$

A very distinctive property of CCS! is that the maximal nesting of restrictions is invariant during evolution.

Proposition 3.5.3 *Let P and Q be CCS! processes. If $P \stackrel{s}{\Longrightarrow} Q$ then $|P|_\nu = |Q|_\nu$.*

Proof. The proposition can be proved by induction on the reductions steps of the operational semantics:

- ACT $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$: from definition 3.5.3 $|\alpha.P|_\nu = |P|_\nu$.
- RES $\dfrac{P \xrightarrow{\alpha} P'}{(\nu\, a)P \xrightarrow{\alpha} (\nu\, a)P'}$ if $\alpha \notin \{a, \overline{a}\}$: by inductive hypothesis we have that $|P|_\nu = |P'|_\nu$ hence by definition 3.5.3 $|(\nu\, a)P|_\nu = |(\nu\, a)P'|_\nu$.
- PAR$_1$ $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$: by inductive hypothesis we have that $|P|_\nu = |P'|_\nu$ hence by definition 3.5.3 $|P \mid Q|_\nu = |P' \mid Q|_\nu$. (Similarly one can prove rule PAR_2)
- COM $\dfrac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$: by inductive hypothesis we have that $|P|_\nu = |P'|_\nu$ $|Q|_\nu = |Q'|_\nu$ and hence by definition 3.5.3 $|P \mid Q|_\nu = |P' \mid Q'|_\nu$.
- REP $\dfrac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$: by inductive hypothesis we have that $|P \mid !P|_\nu = |P'|_\nu$ hence by definition 3.5.3 $|P \mid !P|_\nu = max(|P|_\nu, |!P|_\nu)$ but $|P|_\nu = |!P|_\nu$ thus concluding that $|!P|_\nu = |P'|_\nu$.

□

Remark 3.5.2 *In CCS because of the* unfolding *of recursive definitions the nesting of restrictions can increase unboundedly during evolution*[2]*. E.g., consider* $A(a)$ *where* $A(x) \stackrel{\mathtt{def}}{=} (\nu\, y)(x.\overline{y}.R \mid y.A(x))$ *(see Section 2.3.3) which has the following sequence of transitions*

$$A(a) \stackrel{aaa\ldots}{\Longrightarrow} (\nu y)(R \mid (\nu y)(R \mid (\nu y)(R \mid \ldots)))$$

Another distinctive property of CCS$_!$ is that if a CCS$_!$ process can perform a given action β, it can always do it by performing a number of actions bounded by a value that depends only on the size of the process. In fact, as stated below, for a significant class of processes, the bound can be given solely in terms of the maximal number of nesting of restrictions.

[2]Also in the π-calculus [83], an extension of CCS$_!$ where names are communicated, the nesting of restrictions can increase during evolution due to its name-extrusion capability.

Now, the above statement may seem incorrect since as mentioned earlier CCS! is Turing expressive. One may think that β above could represent a termination signal in a TM encoding, then it would seem that its presence in a computation cannot be determined by something bounded by the syntax of the encoding. Nevertheless, recall that the Turing encoding in [22] may wrongly signal β (i.e., even when the encoded machine does not terminate) but it will diverge afterwards.

Now we introduce some lemmas needed for proving our impossibility results for CCS! processes.

Trios-Processes For technical reasons we shall work with a family of CCS! processes, namely *trios-processes*. These processes can only have prefixes of the form $\alpha.\beta.\gamma$. The notion of trios was introduced for the π-calculus by Parrow in [72]. We shall adapt trios and use them as a technical tool for our purposes.

We shall say that a CCS! process T is a *trios-process* iff all prefixes in T are *trios*; i.e., they all have the form $\alpha.\beta.\gamma$ and satisfy the following: If $\alpha \neq \tau$ then α is a *name* bound in T, and similarly if $\gamma \neq \tau$ then γ is a *co-name* bound in T. For instance $(\nu l)(\tau.\tau.\bar{l} \mid l.a.\tau)$ is a trios-process. We will view a trio $l.\beta.\bar{l}$ as linkable node with incoming link l from another trio, outgoing link $\bar{l}$ to another trio, and contents β.

Interestingly, the family of trios-processes can capture the behaviour of arbitrary CCS! processes via the following encoding:

Definition 3.5.4 *Given a CCS! process P, $[\![P]\!]$ is the trios-process $(\nu\, l)(\tau.\tau.\bar{l} \mid [\![P]\!]_l)$ where $[\![P]\!]_l$, with $l \notin n(P)$, is inductively defined as follows:*

$$
\begin{array}{ll}
[\![0]\!]_l = & 0 \\
[\![\alpha.P]\!]_l = & (\nu\, l')(l.\alpha.\overline{l'} \mid [\![P]\!]_{l'}) \text{ where } l' \notin n(P) \\
[\![P \mid Q]\!]_l = & (\nu\, l', l'')(l.\overline{l'}.\overline{l''} \mid [\![P]\!]_{l'} \mid [\![Q]\!]_{l''}) \text{ where } l', l'' \notin n(P) \cup n(Q) \\
[\![!P]\!]_l = & (\nu\, l')(!l.\overline{l'}.\bar{l} \mid ![\![P]\!]_{l'}) \text{ where } l' \notin n(P) \\
[\![(\nu\, x)P]\!]_l = & (\nu\, x)[\![P]\!]_l
\end{array}
$$

Notice that the trios-process $[\![\alpha.P]\!]_l$ encodes a process $\alpha.P$ much like a linked list. Intuitively, the trio $l.\alpha.\overline{l'}$ has an outgoing link l to its continuation $[\![P]\!]'_l$ and incoming link l from some previous trio. The other cases can be explained analo-

gously. Clearly the encoding introduces additional actions but they are all silent—i.e., they are synchronisations on the bound names l, l' and l''.

Unfortunately the above encoding is not invariant w.r.t. language equivalence because the replicated trio in $[\![!P]\!]_l$ introduces divergence. E.g, $L((\nu x)!x) = \{\epsilon\}$ but $L([\![(\nu x)!x]\!]) = \emptyset$. It has, however, a pleasant invariant property: *weak bisimilarity*, $\approx$.

Now, in order to prove that $P \approx [\![P]\!]$, we define a bisimulation in which we need to take care of the processes derivated from $[\![P]\!]$ by internal communications between non-essential prefixes, i.e. prefixes introduced by the encoding in order to satisfy the structure of trios-processes. These processes are weak bisimilar to $[\![P]\!]$ (and therefore to P), They are characterized in the bisimulation by the function $D_l(P)$.

Proposition 3.5.4 *For every CCS$_!$ process P, $P \approx [\![P]\!]$ where $[\![P]\!]$ is the trios-process constructed from P as in Definition 3.5.4.*

Proof.

We establish a weak bisimulation-up to strong bisimulation including $(P, (\nu\, l)(\bar{l} \mid [\![P]\!]_l))$, it is enough as $[\![P]\!] \approx (\nu\, l)(\bar{l} \mid [\![P]\!]_l)$. Let us define the set of agents $C_l(P)$ associated to P in the bisimulation : $C_l(P) = \{\bar{l} \mid [\![P]\!]_l\} \cup D_l(P)$ where $D_l(P)$ representing processes derivated by non-essential prefixes from $[\![P]\!]$ is defined as follows:

- $D_l(0) = \{0\}$
- $D_l(\alpha.Q) = \{(\nu\, l')\alpha.\overline{l'} | [\![Q]\!]_{l'}\}$ where $l' \notin n(Q)$.
- $D_l(Q \mid R) = \{(\nu\, l', l'')\overline{l'}.\overline{l''} | [\![Q]\!]_{l'} | [\![R]\!]_{l''}\} \cup \{(C_Q \mid C_R) : C_Q \in C_{l'}(Q), C_R \in C_{l''}(R)\}$ where $l', l'' \notin n(P) \cup n(Q)$.
- $D_l(!P) = \{(\nu\, l', l'')(!l.\overline{l'}.\bar{l} \mid \overline{l'}.\bar{l} \mid (\bar{l} \mid)^n \mid ![\![P]\!]_{l'} \mid C_1 \mid C_2 \mid \ldots \mid C_m) : n, m \geq 0, C_i \in C_{l'}(P)\}$ where $l', l'' \notin n(P) \cup n(Q)$.
- $D_l((\nu\, x)P) = \{(\nu\, x)C_P : C_P \in C_l(P)\}$

Now we can define the relation S by $S = \{(P, (\nu\, l)C_l(P))\}$, one can easily verify that S is a weak bisimulation up-to strong bisimulation.

□

Another property of trios is that if a trios-process T can perform an action α, i.e., $T \stackrel{s.\alpha}{\Longrightarrow}$, then $T \stackrel{s'.\alpha}{\Longrightarrow}$ where s' is a sequence of actions whose length bound can be given solely in terms of $|T|_\nu$. This property is not exclusive for these trios-processes. We shall prove this property for a more general kind of processes that we call *unrestricted trios processes*.

We say that a CCS! process T is an *unrestricted trios-process* iff all prefixes in T are *unrestricted trios*; i.e., they all have the form $\alpha.\beta.\gamma$ where α, β and γ can be any (co-)name either free or bound or τ. We also say that a CCS! process T is a *degenerate unrestricted trios-process* iff each of the prefixes in T is an unrestricted trio or is of the form α or $\alpha.\beta$ where α and β can be any (co-)name either free or bound or τ. Notice that if P is an unrestricted trios-process then any process Q such that $P \stackrel{s}{\Longrightarrow} Q$ is a degenerate unrestricted trios-process, i.e., an unrestricted trios-process only can evolve into a degenerate unrestricted trios-process.

Proposition 3.5.5 *Let T be an unrestricted trios-process such that $T \stackrel{s\cdot c}{\Longrightarrow}$ and $n = |T|_\nu$. There exists a sequence s', whose length is bounded by 2^{n+1}, such that $T \stackrel{s'\cdot c}{\Longrightarrow}$.*

Proof.

We know that there must be a minimum sequence (of visible actions) s' such that $T \stackrel{s'\cdot c}{\Longrightarrow}$. We can apply in T the replication law ($!P \equiv P|!P$) repeatedly in order to unfold a number of (non-replicated) occurrences from every replicated process (i.e., of the form $!Q$ for some process Q) enough to generate the sequence $s' \cdot c$ without using more replicated processes. This means there must be an unrestricted trios-process T' such that $T \equiv T'$, $|T|_\nu = |T'|_\nu$, $T' \stackrel{s'\cdot c}{\Longrightarrow}$ where $s' \cdot c$ can be generated without using any rule involving replication, obviously s' is also a minimum sequence (of visible actions) to reach c from T'.

From the above, w.l.o.g we can restrict ourselves to analyse the generation of the sequence $s' \cdot c$ from T'. Now we will prove that the length of this sequence s' is bounded by 2^{n+1} where $n = |T|_\nu = |T'|_\nu$; we will do it by using induction on

n. We can assume that all the bound names in T' are different and that only non-replicated processes participate in the generation of the sequence $s' \cdot c$, therefore we will only refer to prefixes that are not under the scope of replication:

- If $n = 0$ (i.e. there is no restricted declarations): We know that c must be present in some prefix in T'. As all the prefixes in T' are of the form $\alpha.\beta.\gamma$ it is easy to see that the length of s' is bound by 2 as s' is minimum and it is possible to perform the actions of this prefix directly from T'. Notice that the length of s' is 2 when c only appears in prefixes of the form $\alpha.\beta.c$ where α and β are not c.

- If $n \geq 1$: We know that c must be present in some prefix in T'. In particular we know that there is at least one prefix in T' that can eventually provide the occurrence of c in the generation of the sequence $s' \cdot c$. This prefix in T' can have one of the following forms:

  - $c.\beta.\gamma$ $\alpha.c.\gamma$ or $\alpha.\beta.c$ where α, β, and γ are free (co-) names: As in the case $n = 0$ it is easy to see that the length of s' is bound by 2.
  - $\alpha.\beta.c$ where α and β are bound (co-)names: As this prefix can provide the occurrence of c necessary to generate the sequence $s' \cdot c$ from T' there must be a degenerate unrestricted trios-process Q resulting from the evolution of T' after generating s' where the prefix $\alpha.\beta.c$ or $\beta.c$(if α has been consumed before from $\alpha.\beta.c$) can interact with the complementary of α or β respectively. We will consider the case when the prefix $\alpha.\beta.c$ (and not $\beta.c$) is present in Q. The case of the prefix $\beta.c$, as we will see, is easier. As the prefix $\alpha.\beta.c$ and a prefix of the form $\overline{\alpha}.P$ are present in Q and s' has been already generated the presence of the prefix of the form $\overline{\alpha}.P$ could depend on the generation of the whole sequence s', otherwise it would mean that α could be consumed before generating s' then there would be a process similar to Q but with $\beta.c$. Therefore we can assume that the generation of the sequence s' was necessary for $\overline{\alpha}.P$

    it is possible that this means that s' was ge

and immediately after with the complementary of β, $\overline{\alpha}$ and $\overline{\beta}$ [3].

and other(s) prefix(es) can interact with it

–

The interesting situations happen when c is only present in prefixes where c is guarded by at least one bound (co-) name, otherwise it is similar to the case $n = 0$. From now on (in this case) we assume that c is only present in T' in prefixes where c is guarded by at least one bound (co-) name. This implies that at least one of these prefixes participates by providing the occurrence of c for the generation of $s' \cdot c$. Let us consider that prefix.

If the prefix is of the form $\alpha.c.\gamma$ where α is a bound (co-) name, say $(\bar{l})\ l$, then we know that there must a process Q such that $T' \stackrel{s'}{\Longrightarrow} Q \stackrel{c}{\Longrightarrow}$ where a prefix of the form $l.c.\gamma$ and a prefix of the form $\bar{l}.P$ appears. We also know that the prefix $\bar{l}.P$ could not appear before generating s' (otherwise it would mean that there would be a process able to generate c before generating the whole sequence s'.). Therefore s' is necessarily generated before unguarding $\bar{l}$. A crucial observation at this point of the proof is that the actions necessary to unguard $\bar{l}.P$ do not depend on the fact that l is a bound name or not as l. Therefore, even if we take off the restriction declaration of l in T' the same sequence s' is generated before unguarding $\bar{l}.P$, therefore now l is not under the scope of any restriction declaration. Thus, to calculate the size of s' we can consider the case $n = 0$, i.e. the length of s' is bound by 2.

If the prefix is of the form $\alpha.\beta.c$ where α and β are (co-) bound names, we know that α and β refer to the same bound name, say l, because $n = 1$. In this case we consider two possibilities to unguard c according to the number of additional prefixes to interact with the prefix $\alpha.\beta.c$ to unguard c.

Let us suppose that this evolves only one additional prefix, then we know that there must a process Q such that $T' \stackrel{s'}{\Longrightarrow} Q \stackrel{c}{\Longrightarrow}$ where a prefix of the form $\alpha.\beta.c$ and a prefix of the form $\overline{\alpha}.\overline{\beta}.P$ appears. We also know that the prefix $\overline{\alpha}.\overline{\beta}.P$ could not appear before generating s' (otherwise it would mean that there would be a process able to generate c before generating

[3] Although $\alpha.\beta.c$ could interact with $\overline{\alpha}$ without the pre$\overline{\beta}$ could be

the whole sequence s'.). Therefore s' is necessarily generated before unguarding $\overline{\alpha}.\overline{\beta}.P$. We use the crucial observation that the actions necessary to unguard $\overline{\alpha}.\overline{\beta}.P$ do not depend on the fact that l is a bound name or not. Therefore, even if we take off the restriction declaration of l in T' the same sequence s' is generated before unguarding $\overline{\alpha}.\overline{\beta}.P$, therefore now $\overline{\alpha}.\overline{\beta}.P$ is not under the scope of any restriction declaration. Thus, to calculate the size of s' we can consider the case $n = 0$, i.e. the length of s' is bound by 2.

Let us suppose that this evolves two additional prefixes, then we know that there must a process Q such that $T' \stackrel{s'}{\Longrightarrow} Q \stackrel{c}{\Longrightarrow}$ where a prefix of the form $\alpha.\beta.c$, a prefix of the form $\overline{\alpha}.P$ and a prefix of the form $\overline{\beta}.P$ appears. We also know that both the prefix $\overline{\alpha}.P$ and the prefix $\overline{\beta}.P$ could not appear before generating s' (otherwise it would mean that there would be a process able to generate c before generating the whole sequence s'.). Therefore s' is necessarily generated before unguarding the prefix $\overline{\alpha}.P$ and the prefix $\overline{\beta}.P$. We use the crucial observation that the actions necessary to unguard the prefix $\overline{\alpha}.P$ and the prefix $\overline{\beta}.P$ do not depend on the fact that l is a bound name or not. Therefore, even if we take off the restriction declaration of l in T' the same sequence s' is generated before unguarding $\bar{l}$, therefore now l is not under the scope of any restriction declaration. Thus, to calculate the size of s' we can consider the case $n = 0$ for the both prefixes: $\overline{\alpha}.P$ and $\overline{\beta}.P$, i.e. the length of s' is bound by 4.

- $n \geq 1$: We know that c must be present in some non-replicated prefix in T'. The interesting situations happen when c is only present in prefixes where c is guarded by at least one bound (co-) name, otherwise it is similar to the case $n = 0$. From now on (in this case) we assume that c is only present in T' in prefixes where c is guarded by at least one bound (co-) name. This implies that at least one of these prefixes participates by providing the occurrence of c for the generation of $s' \cdot c$. Let us consider that prefix.

  If the prefix is of the form $\alpha.c.\gamma$ where α is a bound (co-) name, say $(\bar{l})\ l$, then we know that there must a process Q such that $T' \stackrel{s'}{\Longrightarrow} Q \stackrel{c}{\Longrightarrow}$ where a prefix of the form $l.c.\gamma$ and a prefix of the form $\bar{l}.P$ appears. We also know that the prefix $\bar{l}.P$ could not appear before generating s' (otherwise it would

mean that there would be a process able to generate c before generating the whole sequence s'.). Therefore s' is necessarily generated before unguarding $\bar{l}$. A crucial observation at this point of the proof is that the actions necessary to unguard $\bar{l}$ do not depend on the fact that l is a bound name or not. Therefore, even if we take off the restriction declaration of l in T' the same sequence s' is generated before unguarding $\bar{l}$, therefore now l is not under the scope of any restriction declaration. Thus, to calculate the size of s' we can consider the case $n-1$, i.e. the length of s' is bound by 2^n.

If the prefix is of the form $\alpha.\beta.c$ where α and β are (co-) bound names, say l and k. In this case we consider two possibilities to unguard c according to the number of additional prefixes to interact with the prefix $\alpha.\beta.c$ to unguard c.

Let us suppose that this evolves only one additional prefix, then we know that there must a process Q such that $T' \stackrel{s'}{\Longrightarrow} Q \stackrel{c}{\Longrightarrow}$ where a prefix of the form $\alpha.\beta.c$ and a prefix of the form $\overline{\alpha}.\overline{\beta}.P$ appears. We also know that the prefix $\overline{\alpha}.\overline{\beta}.P$ could not appear before generating s' (otherwise it would mean that there would be a process able to generate c before generating the whole sequence s'.). Therefore s' is necessarily generated before unguarding $\overline{\alpha}.\overline{\beta}.P$. We use the crucial observation that the actions necessary to unguard $\overline{\alpha}.\overline{\beta}.P$ do not depend on the fact that l, the name associated to α, is a bound name or not. Therefore, even if we take off the restriction declaration of l in T' the same sequence s' is generated before unguarding $\bar{l}$, therefore now l is not under the scope of any restriction declaration. Thus, to calculate the size of s' we can consider the case $n = 0$, i.e. the length of s' is bound by 2.

Let us suppose that this evolves two additional prefixes, then we know that there must a process Q such that $T' \stackrel{s'}{\Longrightarrow} Q \stackrel{c}{\Longrightarrow}$ where a prefix of the form $\alpha.\beta.c$, a prefix of the form $\overline{\alpha}.P$ and a prefix of the form $\overline{\beta}.P$ appears. We also know that both the prefix $\overline{\alpha}.P$ and the prefix $\overline{\beta}.P$ could not appear before generating s' (otherwise it would mean that there would be a process able to generate c before generating the whole sequence s'.). Therefore s' is necessarily generated before unguarding the prefix $\overline{\alpha}.P$ and the prefix

$\overline{\beta}.P$. We use the crucial observation that the actions necessary to unguard the prefix $\overline{\alpha}.P$ and the prefix $\overline{\beta}.P$ do not depend on the fact that l is a bound name or not. Therefore, even if we take off the restriction declaration of l in T' the same sequence s' is generated before unguarding $\bar{l}$, therefore now l is not under the scope of any restriction declaration. Thus, to calculate the size of s' we can consider the case $n = 0$ for the both prefixes: $\overline{\alpha}.P$ and $\overline{\beta}.P$, i.e. the length of s' is bound by 4.

□

Proposition 3.5.6 *Let T be a trios-process such that $T \stackrel{s\cdot\beta}{\Longrightarrow}$. There exists a sequence s', whose length is bounded by a value depending only on $|T|_\nu$, such that $T \stackrel{s'\cdot\beta}{\Longrightarrow}$.*

Proof.

Our approach is to consider a minimal sequence of visible actions $t = \beta_1 \dots \beta_m$ performed by T leading to β (i.e., $P \stackrel{t}{\Longrightarrow}$ and $\beta_m = \beta$) and analyse the *causal dependencies* among the (*occurrences* of) the actions in this t. Intuitively, β_j depends on β_i if T, while performing t, could not had performed β_j without performing β_i first. For example in

$$T = (\nu l)(\nu l')(\nu l'')(\tau.a.\bar{l} \mid \tau.b.\overline{l'} \mid l.l'.\overline{l''} \mid l''.c.\tau)$$

$\beta = c$, $t = abc$, we see that c depends on a and b, but b does not depend on a since T could had performed b before a.

We then consider the unique directed acyclic graph G_t arising from the transitive reduction[4] of the partial ordered induced by the dependencies in t. Because t is minimal, β is the only sink of G_t.

We write $\beta_i \rightsquigarrow_t \beta_j$ (β_j depends directly on β_i) iff G_t has an arc from β_i to β_j. The crucial observation from our restrictions over trios is that if $\beta_i \rightsquigarrow_t \beta_j$ then (the trios corresponding to the occurrences of) β_i and β_j must occur in the scope of a restriction process R_{ij} in T (or in some evolution of T while generating t).

[4]The transitive reduction of a binary relation r on X is the smallest relation r' on X such that the transitive closure of r' is the same as the transitive closure of r.

Take e.g, $T = \tau.a.\tau \mid (\nu\, l)(\tau.b.\bar{l} \mid l.c.\tau)$ with $t = a.b.c$ and $b \rightsquigarrow c$. Notice that the trios corresponding to the actions b and c appear within the scope of the restriction in T.

To give an upper bound on the number of nodes of G_t (i.e., the length of t), we give an upper bound on its length and maximal in-degree. Take a path $\beta_{i_1} \rightsquigarrow_t \beta_{i_2} \ldots \rightsquigarrow_t \beta_{i_u}$ of size u in G_t. With the help of the above observation, we consider sequences of restriction processes $R_{i_1 i_2} R_{i_2 i_3} \ldots R_{i_{u-1} i_u}$ such that for every $k < u$ the actions β_{i_k} and $\beta_{i_{k+1}}$ (i.e., the trios where they occur) must be under the scope of $R_{i_k i_{k+1}}$. Note that any two different restriction processes with a common trio under their scope (e.g. $R_{i_1 i_2}$ and $R_{i_2 i_3}$) must be nested, i.e., one must be under the scope of the other. This induces tree-like nesting among the elements of the sequence of restrictions. E.g., for the restrictions corresponding to $\beta_{i_1} \rightsquigarrow_t \beta_{i_2} \rightsquigarrow_t \beta_{i_3} \rightsquigarrow_t \beta_{i_4}$ we could have a tree-like situation with $R_{i_1 i_2}$ and $R_{i_3 i_4}$ being under the scope of $R_{i_2 i_3}$ and thus inducing a nesting of at least two. Because of the tree-structure, for a sequence of restriction processes, the number m of nesting of them should satisfy $u \leq 2^m$. Since the nesting of restrictions remains invariant during evolution (Proposition 3.5.3) then $u \leq 2^{|T|_\nu}$. Similarly, we give an upper bound $2^{|T|_\nu}$ on the indegree of each node β_j of G_t (by considering sequences $R_{i_1 j}, \ldots, R_{i_m j}$ such that $\beta_{i_k} \rightsquigarrow \beta_j$, i.e having common trio corresponding to β_j under their scope). We then conclude that the number of nodes in G_t is bounded by $2^{|T|_\nu \times 2^{|T|_\nu}}$. □

Main Impossibility Result. We can now prove our main impossibility result.

Theorem 3.5.4 *There exists a Type 2 grammar G such that for every termination-preserving CCS$_!$ process P, $L(G) \neq L(P)$.*

Proof. It suffices to show that no process in CCS$_!^{-\omega}$ can generate the CFL $a^n b^n c$. Suppose, as a mean of contradiction, that P is a CCS$_!^{-\omega}$ process such that $L(P) = a^n b^n c$.

Pick a sequence $\rho = P \stackrel{a^n}{\Longrightarrow} Q \stackrel{b^n c}{\Longrightarrow} T \not\rightarrow$ for a sufficiently large n. From Proposition 3.5.4 we know that for some R, $[\![P]\!] \stackrel{a^n}{\Longrightarrow} R \stackrel{b^n c}{\Longrightarrow}$ and $R \approx Q$. Notice that R may not be a trios-process as it could contain prefixes of the form $\beta.\gamma$ and

γ. However, such prefixes into $\tau.\beta.\gamma$ and $\tau.\tau.\gamma$, we obtain a trios-process R' such that $R \approx R'$ and $|R|_\nu = |R'|_\nu$. We then have $R' \stackrel{b^n c}{\Longrightarrow}$ and, by Proposition 3.5.6, $R' \stackrel{s'\cdot c}{\Longrightarrow}$ for some s' whose length is bounded by a constant k that depends only on $|R'|_\nu$. Therefore, $R \stackrel{s'\cdot c}{\Longrightarrow}$ and since $R \approx Q$, $Q \stackrel{s'\cdot c}{\Longrightarrow} D$ for some D. With the help of Proposition 3.5.3 and from Definition 3.5.4 it is easy to see that $|R'|_\nu = |R|_\nu = |[\![P]\!]|_\nu \leq 1 + |P| + |P|_\nu$ where $|P|$ is the size of P. Consequently the length of s' must be independent of n, and hence for any $s'' \in \mathcal{L}^*$, $a^n s' c s'' \notin L(P)$. Nevertheless $P \stackrel{a^n}{\Longrightarrow} Q \stackrel{s'\cdot c}{\Longrightarrow} D$ and therefore from Proposition 3.2.2 there must be at least one string $w = a^n s' c w' \in L(P)$; a contradiction. □

It turns out that the converse of Theorem 3.5.4 also holds: termination-preserving $CCS_!$ processes can generate non CFL's.

Theorem 3.5.5 *There exists a termination-preserving $CCS_!$ process P such that $L(P)$ is not a CFL.*

Proof. Take

$$P = (\nu\, k, u)(\overline{k} \mid !k.a.(\overline{k} \mid \overline{u})) \mid k.!u.(b \mid c))$$

One can verify that P is termination-preserving. Furthermore, $L(P) \cap a^*b^*c^* = a^n b^n c^n$, hence $L(P)$ is not a CFL since CFL's are closed under intersection with regular languages. □

Now, notice that if we allow the use of $CCS_!$ processes which are not termination-preserving, we can generate $a^n b^n c$ straightforwardly by using a process similar to that of Example 3.2.1.

Example 3.5.1 *Consider the process P below:*

$$\begin{aligned} P &= (\nu\, k_1, k_2, k_3, u_b)(\ \overline{k_1} \mid \overline{k_2} \mid Q_a \mid Q_b \mid Q_c) \\ Q_a &= !k_1.a.(\overline{k_1} \mid \overline{k_3} \mid \overline{u_b}) \\ Q_b &= k_1.!k_3.k_2.u_b.b.\overline{k_2} \\ Q_c &= k_2.(c \mid u_b.DIV) \end{aligned}$$

where $DIV = !\tau$. One can verify that $L(P) = \{a^n b^n c\}$.

Termination-Preserving CCS. Type 0 grammars can be encoded by using the termination-preserving encoding of RAMs in CCS given in [21]. However, the fact that preservation of termination is not as restrictive for CCS as it is for CCS$_!$ can also be illustrated by giving a simple termination-preserving encoding of Context-Free grammars.

Theorem 3.5.6 *For every type 2 grammar G, there exists a termination-preserving CCS process P_G, such that $L(P_G) = L(G)$.*

Proof. For simplicity we restrict ourselves to Type 2 grammars in Chomsky normal form. All production rules are of the form $A \rightarrow B.C$ or $A \rightarrow a$. We can encode the productions rules of the form $A \rightarrow B.C$ as the recursive definition $A(d) \stackrel{\texttt{def}}{=} (\nu\, d')(B(d') \mid d'.C(d))$ and the terminal production $A \rightarrow a$ as the definition $A(d) \stackrel{\texttt{def}}{=} a.\overline{d}$. Rules with the same head can be dealt using the summation $P + Q$. One can verify that, given a Type 2 grammar G, the suggested encoding generates the same language as G.

Notice, however, that there can be a grammar G with a non-empty language exhibiting derivations which do not lead to a sequence of terminal (e.g., $A \rightarrow B.C$, $A \rightarrow a$, $B \rightarrow b$, $C \rightarrow D.C$,$D \rightarrow d$). The suggested encoding does not give us a termination-preserving process. However one can show that there exists another grammar G', with $L(G) = L(G')$ whose derivations can always lead to a final sequence of terminals . The suggested encoding applied to G' instead, give us a termination-preserving process. □

3.5.3 Inside Context Sensitive Languages (CSL)

Context-Sensitive Languages (CSL) are those generated by Type 1 grammars. We conjecture that every language generated by a termination-preserving CCS$_!$ process is context sensitive.

Our conjecture relies on the following claim: suppose that P generates a sequence s of size n. We believe that there must be a trace of P that generates s with a total number of τ actions bounded by kn where k is a constant associated to the size of P. We think that a constant number of τ actions is enough to produce each of the symbols in s.

Now recall that context-sensitive grammars are equivalent to linear bounded non-deterministic Turing machines. That is a non-deterministic Turing machine with a tape with only kn cells, where n is the size of the input and k is a constant associated with the machine. Given P, we can define a non-deterministic machine which simulates the runs of P using the semantics of CCS$_!$ and which uses as many cells as the total number of performed actions, silent or visible, multiplied by a constant associated to P. Now, we define the conjecture:

Conjecture 3.5.7 *If P is a termination-preserving CCS$_!$ process then $L(P)$ is a context sensitive language.*

Notice that from the above conjecture and Theorem 3.5.4, we can conjecture that the languages generated by termination-preserving CCS$_!$ processes form a proper subset of context sensitive languages.

3.6 Summary and Related Work

In this chapter, we studied the expressiveness of encodings in CCS$_!$ that do not allow unfaithful computations that move from a (weakly) terminating state into a (strongly) non-terminating state that do not correspond to any configuration of the encoded process. It is known that only unfaithful encodings, i.e. encodings with unfaithful computations, can encode Turing Machines into CCS$_!$ [22]. We extended the work in [22] by considering the existence of faithful encodings of models of computability strictly less expressive than Turing Machines in CCS$_!$. We proved that CCS$_!$ can faithfully encode Regular Languages but it is not possible to provide a faithful encoding of Context Free Languages. Finally we conjectured that languages generated by using only faithful computations are Context Sensitive.

The closest related work is that in [21, 22] already discussed in Section 3.1. Furthermore in [21] the authors also provide a discrimination result between CCS$_!$ and CCS by showing that the divergence problem (i.e., given P, whether P has an infinite sequence of τ moves) is decidable for the former calculus but not for the latter.

In [38] Giambiagi et al. study replication and recursion in CCS focusing on the role of name scoping. In particular they show that CCS$_!$ is equivalent to CCS with recursion with static scoping. The standard CCS in [59] is shown to have dynamic scoping. A survey on the expressiveness of replication vs recursion is given in [71] where several decidability results about variants of π, CCS and Ambient calculi can be found. None of these works study replication with respect to computability models less expressive than Turing Machines.

In [66] Nielsen et al. showed a separation result between replication and recursion in the context of temporal concurrent constraint programming (tccp) calculi. They show that the calculus with replication is no more expressive than finite-state automata while that with recursion is Turing Powerful. The semantics of tccp is rather different from that of CCS. In particular, unlike in CCS, processes interact via the shared-memory communication model and communication is asynchronous.

In the context of calculi for security protocols, the work in [49] uses a process calculus to analyse the class of ping-pong protocols introduced by Dolev and Yao. Huttel and Srba show that all nontrivial properties, in particular reachability, become undecidable for a very simple recursive variant of the calculus. The authors then show that the variant with replication renders reachability decidable. The calculi considered are also different from CCS. For example no restriction is considered and communication is asynchronous.

There is extensive work in process algebras and rewriting transition systems providing expressiveness hierarchies similar to that of Chomsky as well as results closely related to those of formal grammars. For example works involving characterisation of regular expression w.r.t. bisimilarity include [51, 58] and more recently [10]. An excellent description is provided in [19]. These works do not deal with replication nor the restriction operator which are fundamental to our study.

As for future work, we plan to provide a proof for Conjecture 3.5.7 or to find a counterexample. Moreover a somewhat complementary study to the one carried in this paper would be to investigate what extension to CCS$_!$ is needed for providing faithful encoding of RAMs. Clearly the extension with recursion does the job but there may be simpler process constructions from process algebra which also do

the job.

The classification of $\mathrm{CCS}_!^{-\omega}$ in the Chomsky Hierarchy presented in this chapter was originally published as [7]. In addition to the work in [7], in this chapter we proved that the set of $\mathrm{CCS}_!^{-\omega}$ processes is undecidable.

Chapter 4

On the Expressive Power of Restriction and Priorities in CCS with replication

In the previous chapter, we presented a classification of $\mathrm{CCS}_!$ variants based on semantic properties: That of being termination preserving. In this chapter, we shall provide instead a classification based on syntactic properties. More precisely, we study the expressive power of restriction and its interplay with replication. We do this by considering several syntactic variants which differ from each other in the use of restriction with respect to replication. In particular, we consider three syntactic variations of $\mathrm{CCS}_!$ which do not allow the use of an unbounded number of restrictions: $\mathrm{CCS}_!^{-!\nu}$ is the fragment of $\mathrm{CCS}_!$ not allowing restrictions under the scope of a replication. $\mathrm{CCS}_!^{-\nu}$ is the restriction-free fragment of $\mathrm{CCS}_!$. The third variant is $\mathrm{CCS}_{!+pr}^{-!\nu}$ which extends $\mathrm{CCS}_!^{-!\nu}$ with Phillips' priority guards.

We show that the use of unboundedly many restrictions in $\mathrm{CCS}_!$ is necessary for obtaining Turing expressiveness in the sense of Busi et al [22]. We do this by showing that there is no encoding of RAMs into $\mathrm{CCS}_!^{-!\nu}$ which preserves and reflects convergence. We also prove that up to failures equivalence, there is no encoding from $\mathrm{CCS}_!$ into $\mathrm{CCS}_!^{-!\nu}$ nor from $\mathrm{CCS}_!^{-!\nu}$ into $\mathrm{CCS}_!^{-\nu}$. As lemmata for the above results we prove that convergence is decidable for $\mathrm{CCS}_!^{-!\nu}$ and that language equivalence is decidable for $\mathrm{CCS}_!^{-\nu}$. As corollary it follows that con-

vergence is decidable for restriction-free CCS. Finally, we show the expressive power of priorities by providing an encoding of RAMs in $\text{CCS}^{-!\nu}_{!+pr}$.

The results in this chapter were published as [8].

4.1 Introduction

Recall that [22] states that, in spite of its being less expressive than CCS, $\text{CCS}_!$ is in fact Turing powerful. This is done by encoding Random Access Machines (RAMs) [63]. The fundamental property of the encoding is that it *preserves (and reflects) convergence*; i.e., the RAM converges if and only if its encoding converges.

The $\text{CCS}_!$ encoding of RAMs in [22] uses an unbounded number of restrictions arising from having restriction operators *under the scope* of a replication operator as for example in $!(\nu x)P$. Similarly, the CCS encoding of RAMs in [21] involves also an unbounded number of restrictions arising from having restrictions under the scope of recursive expressions as for example in $\mu X.(\nu x)(P \mid X)$. One then may wonder if the generation of unboundedly many names is necessary for Turing Expressiveness.

In this chapter we study the expressiveness of restriction and its interplay with replication. We do this by considering two syntactic fragments of $\text{CCS}_!$, namely $\text{CCS}^{-!\nu}_!$ and $\text{CCS}^{-\nu}_!$ which differ from $\text{CCS}_!$ in the occurrences of restriction under the scope of replication. These fragments and a variant of $\text{CCS}_!$, $\text{CCS}^{-!\nu}_{!+pr}$, as well as our classification criteria are described and motivated below.

Although different in nature, our work was inspired by the study of decidable classes (wrt satisfiability) of formulae involving the occurrence of existential quantifiers *under the scope* of universal quantification. E.g., Skolem showed that the class of formulae of the form $\forall y_1 \ldots y_n \exists z_1 \ldots z_m F$, where F is quantifier-free formula, is undecidable while from Gödel we know that its subclass $\forall y_1 y_2 \exists z_1 \ldots z_m F$ is decidable [14].

The $\text{CCS}_!$ Variants. As explained above $\text{CCS}_!$ allows processes with restriction under the scope of replication and hence they can generate an unbounded number

of restricted names. In order to allow only processes with a number of restricted names bounded by their size, we consider two variants of $\text{CCS}_!$ not allowing restrictions under the scope of replications.

Definition 4.1.1 ($\text{CCS}_!^{-!\nu}$ and $\text{CCS}_!^{-\nu}$) *The processes of $\text{CCS}_!^{-!\nu}$ are those $\text{CCS}_!$ processes which do not have occurrences of a process of the form $(\nu x)P$ within a process of the form $!R$. The processes of $\text{CCS}_!^{-\nu}$ are those $\text{CCS}_!$ processes with no occurrences of processes of the form $(\nu x)P$.*

To illustrate the expressiveness of $\text{CCS}_!^{-!\nu}$ take for example

$$P = (\nu k)(\nu u)(\bar{k} \mid\ !(k.a.(\bar{k} \mid \bar{u})) \mid k.!(u.b))$$

which uses only two restricted names. The reader familiar with CCS can verify that the set of (maximal) finite sequences of visible actions performed by P corresponds to the *context-free* language $a^n b^n$. A similar but slightly more complex example involves a $\text{CCS}_!^{-!\nu}$ process with only five restricted names whose set of (maximal) finite sequences of visible actions corresponds to the *context-sensitive* language $a^n b^n c^n$–see [7].

Now, one may wonder whether a process that uses only a number of restricted names bounded by its size, can be encoded, perhaps by introducing some additional non-restricted names, into one which uses none. For this purpose we also consider $\text{CCS}_!^{-\nu}$ defined above.

Finally, we may also wonder whether some other natural process construct can replace the use in $\text{CCS}_!$ of unboundedly many restrictions in achieving Turing expressiveness. For this purpose we shall consider a third variant $\text{CCS}_{!+pr}^{-!\nu}$, introduced later on in this chapter, corresponding to $\text{CCS}_!^{-!\nu}$ extended with Phillips' *priority guards* construct [76].

Classifying Criteria. Our main comparison criteria for the above variants are the decidability of *convergence* and their relative expressiveness wrt *failures equivalence* [18, 60].

As mentioned before, convergence is a fundamental property of processes and its preservation and reflection are also fundamental properties of the encoding of

RAMs in $CCS_!$. Furthermore, we choose it over divergence because the former is undecidable for $CCS_!$ while the latter is already known to be decidable for $CCS_!$.

Failures equivalence is a well-established notion of process equivalence and we choose it over other equivalences because of its sensitivity to convergence. In fact unlike failures equivalence, other standard equivalences for observable behaviour such as weak bisimilarity, must testing, trace equivalence and language equivalence may actually equate a convergent process with a non-convergent one. We already proved this claim about sensitivity to convergence in Section 2.4.2.

4.1.1 Contributions

Our main contributions are the following:

- We show that convergence is decidable for $CCS_!^{-!\nu}$ and thus that there is no (computable) encoding, which preserves and reflects convergence, of RAMs using only a bounded number of restricted names. We do this by encoding $CCS_!^{-!\nu}$ into Petri Nets. Thus convergence is also decidable for the fragment of CCS with no restrictions within recursive expressions, here refered to as $CCS^{-\mu\nu}$, because of the convergence preserving and reflecting encoding into $CCS_!^{-!\nu}$ given in [38] (Section 4.2).

- We show that, up to failures equivalence, $CCS_!$ is strictly more expressive than $CCS_!^{-!\nu}$ and, similarly, that $CCS_!^{-!\nu}$ is strictly more expressive than $CCS_!^{-\nu}$. Thus up to failures equivalence, we cannot encode a process with an unbounded number of restrictions into one with a bounded number of restrictions, nor one with a bounded number of restrictions into a restriction-free process. (Section 4.4) .

- We show that priorities confer significant expressive power to $CCS_!^{-!\nu}$, it is done by showing that adding Phillips' priority guards to $CCS_!^{-!\nu}$ renders the resulting calculus capable of encoding RAMs. Furthermore, unlike the encoding into $CCS_!$ and just like the encoding into CCS, the encoding of RAMs into $CCS_{!+pr}^{-!\nu}$ preserves and reflects both convergence and divergence. This bears witness to the expressive power of Phillips' priority guards (Section 4.6).

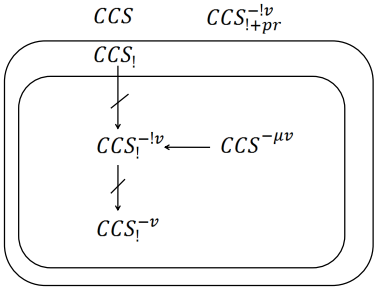


Figure 4.1: A (crossed) arrow from $\mathcal{C}$ to $\mathcal{C}'$ represents the (non) existence of an encoding from $\mathcal{C}$ into $\mathcal{C}'$ preserving and reflecting failures equivalence. Convergence is/isn't decidable for each $\mathcal{C}$ in/outside the inner rectangle. Divergence is/isn't decidable for each $\mathcal{C}$ in/outside the outer rectangle.

The classification of the various fragments mentioned above are summarized in Figure 4.1.1. The undecidability of convergence and decidability of divergence for $\text{CCS}_!$ as well as the undecidability of both divergence and convergence for CCS were shown in [21, 22]. The other results are derived from the work here presented.

Outline of this chapter. The remainder of this chapter is organized as follows. In Section 4.2 we prove the decidability of convergence for $\text{CCS}_!^{-!\nu}$. In Section 4.3 we show that language equivalence is decidable in $\text{CCS}_!^{-\nu}$. In Section 4.4 we provide the separation results between $\text{CCS}_!$ and $\text{CCS}_!^{-!\nu}$ and between $\text{CCS}_!^{-!\nu}$ and $\text{CCS}_!^{-\nu}$ up to failures equivalence. In Section 4.5 we prove the correctness of an encoding from $\text{CCS}^{-\mu\nu}$ into $\text{CCS}_!^{-\nu}$ up to $\sim_F$. In Section 4.6 we show that $\text{CCS}_!^{-!\nu}$ with Phillips' priority guards is Turing Expressive. In Section 4.7, we conclude by summarising this chapter and discussing some related work.

4.2 Decidability of Convergence for $\text{CCS}_!^{-!\nu}$

In this section we show the decidability of convergence for $\text{CCS}_!^{-!\nu}$ by a reduction to the same problem for a fragment of Petri Nets.

4.2.1 Convergence-invariant properties in fragments of $\text{CCS}_!^{-!\nu}$

We use reduction bisimilarity and strong bisimilarity in order to prove Propositions 4.2.2 and 4.2.3.

Proposition 4.2.1 *Let P and Q be CCS$_!$ processes. If $P \sim_r Q$ then P converges if and only if Q converges.*

Proof. If P converges, there exists a finite sequence of τ-actions as follows:

$$P = P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots P_n \not\xrightarrow{\tau}$$

By Definition 2.4.1, there must be a sequence of τ-actions as follows:

$$Q = Q_1 \xrightarrow{\tau} Q_2 \xrightarrow{\tau} \ldots Q_n$$

Where $P_i \sim_r Q_i$ $1 \leq i \leq n$. As $P_n \sim_r Q_n$, $Q_n \not\xrightarrow{\tau}$. Therefore Q is convergent. □

Notice that decidability of convergence for $\text{CCS}_!^{-!\nu}$ can be reduced to the decidability of convergence for $\text{CCS}_!^{-\nu}$.

Proposition 4.2.2 *For every P in CCS$_!^{-!\nu}$ one can effectively construct a CCS$_!^{-\nu}$ process P', such that P converges if only if P' converges.*

Proof. First α-convert P into a process P' so that each bound name in P is replaced with a distinct bound name. Then remove from the process P' each occurrence of a "(νx)". Let P'' be the resulting restriction-free process. As it is known that α-conversion preserves convergence, it remains to show that P' converges if and only if P'' converges. This can be obtained by using Proposition 4.2.1 and showing that $P' \sim_r P''$. For this latter one can easily verify that $\{(Q, f(Q) \mid Q \in \text{CCS}_!^{-!\nu} with\ distinct\ bound\ names\}$ where $f(Q)$ determines the process after removing each occurrence of a "(νx)" in the process Q is a reduction bisimulation and that $(P', P'') \in \{(Q, f(Q) \mid Q \in \text{CCS}_!^{-!\nu} with\ distinct\ bound\ names\}$. □

Consequently, in what follows we reduce the convergence problem for $\text{CCS}_!^{-\nu}$ to convergence problem in Petri nets.

In order to simplify the reduction to Petri Nets, we shall consider the fragment $\text{CCS}_{s!}^{-\nu}$ of those $\text{CCS}_!^{-\nu}$ processes in which replication can only be applied to prefix or summation processses.

Proposition 4.2.3 *For every $CCS_!^{-\nu}$ process P, one can effectively construct a $CCS_{s!}^{-\nu}$ process Q such that P converges iff Q converges.*

Proof.

In [83] it is shown that for any processes P, Q in the π-calculus (hence in $\text{CCS}_!^{-\nu}$), $!!P$ and $!P$ are strongly bisimilar ($!!P \sim !P$), similarly $!(P \mid Q) \sim !P \mid !Q$ and $!0 \sim 0$. From the well-known fact that $\sim$ is a congruence, systematically we can replace in any $\text{CCS}_!^{-\nu}$ process P every occurrence of the form $!!R$ with $!R$, $!(R|R')$ with $!R \mid !R'$, and $!0$ with 0. The resulting $\text{CCS}_{s!}^{-\nu}$ process Q is strongly bisimilar to P, as $\sim$ preserves convergence then P converges iff Q converges.

□

4.2.2 The Reduction to Petri Nets

Here we shall provide a (Unlabelled Place/Transition) Petri Net semantics for $\text{CCS}_{s!}^{-\nu}$ which considers only the τ moves. For these Petri Nets convergence is decidable [33].

Petri nets were already introduced in Chapter 2, however, we find it convenient to recall the definition here.

Definition 4.2.1 (Petri Nets) *A Petri net is a tuple (S, T), where S is a set of places, T is a set of transitions $\mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(S)$ with $\mathcal{M}_{fin}(S)$ being a finite multiset of S called a marking.*

A transition (c, p) is written in the form $c \Longrightarrow p$. A transition is enabled at a marking m if $c \subseteq m$. The execution of the transition produces the marking m' $= (\ m \setminus c\ ) \oplus p$ (where $\setminus$ and $\oplus$ are the difference and the union operators on multisets). This is written as $m \rhd m'$. If no transition is enable at m we say that

m *is a* dead marking. *A marked Petri net is a tuple* (S, T, m_0)*, where* (S, T) *is a Petri net and* m_0 *is the initial marking.*

We say that the marked Petri net (S, T, m_0) converges *iff there exists a dead marking* m' *such that* $m_0(\triangleright)^* m'$.

Intuitively, we will associate to each $\text{CCS}_{s!}^{-\nu}$ process a Petri net so that:

- Places are identified as syntactic components reachable from P,
- Markings are descriptions of processes reachable from P through τ-actions. The places and tokens in the marking represent different syntactic components and their number of occurrences in the process described.
- Transitions represent the τ-actions enabled to be performed at certain process. Input places correspond to the components in the process involved in the τ-action and Output places are the components to be enabled once the τ-action has been executed.

Given a Petri net for P the elements of $Sub(P)$ below will be the syntactic components represented by places in the Petri net.

Definition 4.2.2 *Define* $Sub(P)$*, where* $P \in CCS_{s!}^{-\nu}$*, as* $Sub(0) = \{0\}$, $Sub(\Sigma_{i\in I} P_i) = \{\Sigma_{i\in I} P_i\} \cup (\bigcup_{i\in I} Sub(P_i))$, $Sub(\alpha.P) = \{\alpha.P\} \cup Sub(P)$, $Sub(!P) = \{!P\} \cup Sub(P)$, $Sub(P \mid Q) = Sub(P) \cup Sub(Q)$.

$Sub(P)$ denotes the set of all null, replicated, summation, prefix processes occurring in P. Since a process P may have several parallel occurrences of an element in $Sub(P)$ we use a multi-set $Occur(P)$ take into account its number of occurrences.

Definition 4.2.3 (Occurrence) *Let* $P \in CCS_{s!}^{-\nu}$*. The multiset of processes which occur in* P*,* $Occur(P)$*, is given by the following rule:* $Occur(P) = Occur(Q) \oplus Occur(R)$ *if* $P = Q \mid R$ *else* $Occur(P) = \{P\}$*. Furthermore, we say that* Q *is an occurrence of a process* P *if and only if* $Q \in Occur(P)$.

$Occur(P)$ associates to a $\text{CCS}_{s!}^{-\nu}$ process P the multiset of its immediate parallel components (occurrences) and will be identified as the marking of P in the Petri net.

We are now ready to define our Petri net encoding of $\text{CCS}_{s!}^{-\nu}$ processes.

Definition 4.2.4 (Nets for $\text{CCS}_{s!}^{-\nu}$) *Given a $CCS_{s!}^{-\nu}$ process P, we define its Petri net $N_P = (S, T)$ where $S = \{Q \mid Q \in Sub(P)\}$ and $T = T_1 \cup T_2$ where: $T_1 = \{\{Q\} \Longrightarrow Occur(Q') \mid Q \xrightarrow{\tau} Q' \text{ where } Q \in Sub(P)\}$ and $T_2 = \{\{Q, R\} \Longrightarrow Occur(Q') \oplus Occur(R') | Q \xrightarrow{\alpha} Q' \text{ and } R \xrightarrow{\overline{\alpha}} R' \text{ where } Q \text{ and } R \in Sub(P)\}$.*

The corresponding marked Petri net is $N_P(Occur(P)) = (S, T, Occur(P))$.

Clearly, given P, N_P and its marked one can be effectively constructed.

Roughly speaking, the set of transitions T represents the possible τ moves to be performed and the initial marking $Occur(P)$ is the one which identifies the process P. In particular:

- T_1 : this type of transition reflects a τ move coming from one of the components, it is referred as P, going to the process P'. Notice as a token representing P is consumed and the tokens representing P', there might be more than one component, are added, in this way the transition reflects the evolution from the component P into the process P' .

- T_2 : this type of transition reflects the τ-actions resulting from the synchronisation of two components P and Q, as a result of the synchronisation the processes P' and Q' are reached, in this case, a token associated to both P and another one associated to Q are consumed, the tokens representing P' and Q' are added.

We can now state the correctness of the encoding of $\text{CCS}_!^{-\nu}$ into Petri nets.

Proposition 4.2.4 *Consider two $CCS_{s!}^{-\nu}$ processes P and Q, if $P \xrightarrow{\alpha} Q$ then $Sub(Q) \subseteq Sub(P)$.*

Proof. This proposition can be proven by induction on the depth of the inference of $P \xrightarrow{\alpha} P'$. Let $P \xrightarrow{\alpha} P'$ be the conclusion of the last step in the inference. We show below some of the cases, the rest of the cases can be proven in a similar way:

- If $P \xrightarrow{\alpha} P'$ has been introduced by Rule PAR_1, then P is of the form $P = P_1 \mid P_2$ and $P' = P_1' \mid P_2$: so we know that $P_1 \xrightarrow{\alpha} P_1'$. Since $Sub(P_1' \mid P_2) = Sub(P_1') \cup Sub(P_2)$ and by induction hypothesis $Sub(P_1') \subseteq Sub(P_1)$, then $Sub(P_1' \mid P_2) \subseteq Sub(P_1) \cup Sub(P_2) = Sub(P_1 \mid P_2)$.

- If $P \xrightarrow{\tau} P'$ has been introduced by Rule COM, then P is of the form $P_1 \mid P_2$ and $P' = P_1' \mid P_2'$: so we know that $P_1 \xrightarrow{l} P_1'$ and $P_2 \xrightarrow{\bar{l}} P_2'$. Since $Sub(P_1' \mid P_2') = Sub(P_1') \cup Sub(P_2')$ and by induction hypothesis $Sub(P_1') \subseteq Sub(P_1)$ and $Sub(P_2') \subseteq Sub(P_2)$, then $Sub(P_1' \mid P_2') \subseteq Sub(P_1) \cup Sub(P_2) = Sub(P_1 \mid P_2)$.

- if $P \xrightarrow{\tau} P'$ has been introduced by Rule REPL2, then P is of the form $!P_1$ and $P' = P_2 \mid P_3 \mid !P_1$, where $P_1 \xrightarrow{l} P_2$ and $R \xrightarrow{\bar{l}} P_3$: since $Sub(P_2 \mid P_3 \mid !P_1) = Sub(P_2) \cup Sub(P_3) \cup Sub(!P_1)$ and by induction hypothesis $Sub(P_2) \subseteq Sub(P_1)$ and $Sub(P_3) \subseteq Sub(P_1)$, then $Sub(P_2 \mid P_3 \mid !P_1) \subseteq Sub(P_1) \cup Sub(!P_1)$, as $Sub(!P_1) = \{!P_1\} \cup Sub(P_1)$ therefore $Sub(P_2 \mid P_3 \mid !P_1) \subseteq Sub(!P_1)$.

□

From Proposition 4.2.4, we obtain the following helpful corollary.

Corollary 4.2.1 *Consider two $CCS_{s!}^{-\nu}$ processes P and Q, if $P\ (\xrightarrow{\tau})^*\ Q$ then $Sub(Q) \subseteq Sub(P)$.*

Now we shall prove some additional statements.

Proposition 4.2.5 *Let us consider two $CCS_{s!}^{-\nu}$ processes P and Q and its corresponding Petri nets $N_P = (S, T)$ and $N_Q = (S', T')$ such that $P(\xrightarrow{\tau})^*Q$. Then $S' \subseteq S$ and $T' \subseteq T$.*

Proof.

From Corollary 4.2.1 we know that $S' \subseteq S$. By using Definition 4.2.4, we prove by cases that $T' \subseteq T$. Let t be a transition in T' in one of the following forms:

- $t = \{R\} \Longrightarrow Occur(R')$ and $R \xrightarrow{\tau} R'$ for some process $R \in Sub(Q)$. By Corollary 4.2.1 we know that $Sub(Q) \in Sub(P)$ and by using the definition of N_P there must be a transition in T of the form $\{R\} \Longrightarrow Occur(R')$ and $R \xrightarrow{\tau} R'$ for any $R \in Sub(P)$, therefore $t \in T$.

- $t = \{R, S\} \Longrightarrow Occur(R') \oplus Occur(S')$ and $R \xrightarrow{\alpha} R'$ and $S \xrightarrow{\overline{\alpha}} S'$ for two processes R and $S \in Sub(Q)$. By Corollary 4.2.1 we know that $Sub(Q) \in Sub(P)$ and by using the definition of N_P there must be a transition in T of the form $\{R, S\} \Longrightarrow Occur(R') \oplus Occur(S')$ and $R \xrightarrow{\alpha} R'$ and $S \xrightarrow{\overline{\alpha}} S'$ for any two processes R and $S \in Sub(P)$, therefore $t \in T$.

Hence $T' \subseteq T$.

□

Proposition 4.2.6 *Let us consider a $CCS_{s!}^{-\nu}$ processes P and its corresponding Petri net $N_P = (S, T)$. If $P \xrightarrow{\tau} Q$ then $Occur(P) \rhd Occur(Q)$.*

Proof.

There are two cases for $P \xrightarrow{\tau} Q$:

- $P = P_1 \mid P_2 \ldots \mid P_j \mid \ldots P_n$ and $Q = P_1 \mid P_2 \ldots \mid P'_j \mid \ldots P_n$, where $P_i \in Sub(P)\ \forall\ i \in \{1, \ldots, n\}$ and $P_j \xrightarrow{\tau} P'_j$ for some $j \in \{1, \ldots, n\}$. From Definition of N_P, there is a transition $t = \{P_j\} \Longrightarrow Occur(P'_j)$ as $P_j \xrightarrow{\tau} P'_j$. As $P_j \in \{P_1, \ldots, P_n\} = Occur(P)$, t is enabled at the marking $Occur(P)$. By applying t at $Occur(P)$, the next marking is $m' = (\ Occur(P) \setminus \{P_j\}\ ) \oplus \{Occur(P'_j)\} = Occur(Q)$. Therefore, $Occur(P) \rhd Occur(Q)$.

- $P = P_1 \mid P_2 \ldots \mid P_j \mid \ldots \mid P_k \mid \ldots P_n$ and $Q = P_1 \mid P_2 \ldots \mid P'_j \mid \ldots \mid P'_k \mid \ldots P_n$ where $P_i \in Sub(P)\ \forall\ i \in$

$\{1, \ldots, n\}$ and $P_j \xrightarrow{l} P'_j$ and $P_k \xrightarrow{\bar{l}} P'_k$ for some j, $k \in \{1, \ldots, n\}$ where $j < k$. From Definition of N_P, there must be a transition $t = \{P_j, P_k\} \Longrightarrow Occur(P'_j) \oplus Occur(P'_k)$ as $P_j \xrightarrow{l} P'_j$ and $P_k \xrightarrow{\bar{l}} P'_k$. As P_j and $P_k \in \{P_1, \ldots, P_n\} = Occur(P)$, t is enabled at the marking $Occur(P)$. By applying t at $Occur(P)$, the next marking is $m' = (Occur(P) \setminus \{P_j, P_k\}) \oplus \{Occur(P'_j)\} \oplus \{Occur(P'_k)\} = Occur(Q)$. Therefore, $Occur(P) \rhd Occur(Q)$.

□

Proposition 4.2.7 *If $m \rhd m'$ in a Petri net (S, T), then $m \rhd m'$ in any petri net (S', T') where $T \subseteq T'$.*

Proof.

If $m \rhd m'$ in (S,T), then there must be a transition $t \in T$ where $t = c \Longrightarrow p$ where $c \subseteq m$ and the marking $m' = (m \setminus c) \oplus p$. In any petri net (S', T') where $T \subseteq T'$, t is enabled if the marking is m, hence the marking m' can be produced from m applying t in (S', T') therefore $m \rhd m'$.

□

Proposition 4.2.8 *Consider a $CCS^{-\nu}_{s!}$ process P and the corresponding Petri Net $N_P = (S, T)$. We have the following:*

1. *for any Q such that $P(\xrightarrow{\tau})^*Q$, if $Q \xrightarrow{\tau} R$ then we have $Occur(Q) \rhd Occur(R)$ in N_P.*

2. *for any Q such that $P(\xrightarrow{\tau})^*Q$ and any marking m in N_P , if $Occur(Q) \rhd m$ in N_P then there exists a R such that $Q \xrightarrow{\tau} R$ and $m = Occur(R)$.*

Proof.

We divide the proof into two parts according to the items in the proposition:

Proof of Item 1 : Let $N_Q = (S', T')$ be the petri net associated to Q. From Proposition 4.2.6 and $Q \xrightarrow{\tau} R$, we know that $Occur(Q) \rhd Occur(R)$ in N_Q. From Proposition 4.2.5 and $P(\xrightarrow{\tau})^*Q$ we obtain that $T' \subseteq T$. By using Proposition 4.2.7 $Occur(Q) \rhd Occur(R)$ in N_P.

Proof of item 2 : If $Occur(Q) \rhd m$ in N_P, there must be a transition t in T in one of the following forms:

- $t = \{R\} \implies Occur(R')$ and $R \xrightarrow{\tau} R'$ for some process $R \in Sub(P)$ where $\{R\} \subseteq Occur(Q)$ and $m = (\ Occur(Q) \setminus \{R\}\ ) \oplus Occur(R')$: As $\{R\} \subseteq Occur(Q)$, $R \in Sub(Q)$ and the multiset $Ocurr(R') = \{R'_1, R'_2, \ldots, R'_m\}$ for some m where $R'_i \in Sub(Q)$ for $1 \leq i \leq m$. In fact for some m,n $Occur(Q) = \{Q_1, Q_2, \ldots, Q_{j-1}, R, Q_{j+1}, \ldots, Q_n\}$ where $Q_i \in Sub(Q)$ for $1 \leq i \leq n$ and $m = \{Q_1, Q_2, \ldots, Q_{j-1}, R'_1, R'_2, \ldots, R'_m, Q_{j+1}, \ldots, Q_n\}$, therefore $m = Occur(Q_1 \mid Q_2 \mid \ldots \mid Q_{j-1} \mid R'_1 \mid R'_2 \mid \ldots \mid R'_m \mid Q_{j+1} \mid \ldots \mid Q_n)$ and $Q = Q_1 \mid Q_2 \ldots Q_{j-1} \mid R \mid Q_{j+1} \mid \ldots \mid Q_n \xrightarrow{\tau} Q_1 \mid Q_2 \ldots \mid Q_{j-1} \mid R'_1 \mid R'_2 \mid \ldots \mid R'_m \mid Q_{j+1} \mid \ldots \mid Q_n$ as $R \xrightarrow{\tau} R' = R'_1 \mid R'_2 \mid \ldots \mid R'_m$.

- $t = \{R, S\} \implies Occur(R') \oplus Occur(S')$ and $R \xrightarrow{\alpha} R'$ and $S \xrightarrow{\overline{\alpha}} S'$ for some processes $R, S \in Sub(P)$ where $\{R\}, \{S\} \subseteq Occur(Q)$ and $m = (\ Occur(Q) \setminus (\{R\} \oplus \{S\})) \oplus Occur(R') \oplus Occur(S')$: As $\{R\}$, $\{S\} \subseteq Occur(Q)$, R, $S \in Sub(Q)$ and the multiset $Ocurr(R') = \{R'_1, R'_2, \ldots, R'_l\}$ for some l where $R'_i \in Sub(Q)$ for $1 \leq i \leq l$ and the multiset $Ocurr(S') = \{S'_1, S'_2, \ldots, S'_m\}$ for some m where $S'_i \in Sub(Q)$ for $1 \leq i \leq m$. In fact for some l, m,n $Occur(Q) = \{Q_1, Q_2, \ldots, Q_{j-1}, R, Q_{j+1}, \ldots, QS, Q_{j+1}, \ldots, Q_n\}$ where $Q_i \in Sub(Q)$ for $1 \leq i \leq n$ and $m = \{Q_1, Q_2, \ldots, Q_{j-1}, R'_1, R'_2, \ldots, R'_l, Q_{j+1}, \ldots, Q_{k-1}, S'_1, S'_2, \ldots, S'_m , Q_{k+1}, \ldots, Q_n\}$, therefore $m = Occur(Q_1 \mid Q_2 \mid \ldots \mid Q_{j-1} \mid R'_1 \mid R'_2 \mid \ldots \mid R'_l \mid Q_{j+1} \mid \ldots \mid Q_{k-1} \mid S'_1 \mid S'_2 \mid \ldots \mid S'_m \mid Q_{k+1} \mid \ldots \mid Q_n)$ and $Q = Q_1 \mid Q_2 \ldots Q_{j-1} \mid R \mid Q_{j+1} \mid \ldots \mid Q_{k-1} \mid S \mid Q_{k+1} \ldots \mid Q_n \xrightarrow{\tau} Q_1 \mid Q_2 \ldots \mid Q_{j-1} \mid R'_1 \mid R'_2 \mid \ldots \mid R'_l \mid Q_{j+1} \mid \ldots \mid Q_{k-1} \mid S'_1 \mid S'_2 \mid \ldots \mid S'_m \mid Q_{k+1} \mid \ldots Q_n$ as $R \xrightarrow{\alpha} R' = R'_1 \mid R'_2 \mid \ldots \mid R'_l$ and $S \xrightarrow{\overline{\alpha}} S' = S'_1 \mid S'_2 \mid \ldots \mid S'_m$.

□

From Proposition 4.2.8, we have the following corollary:

Corollary 4.2.2 *Let P, Q be two $CCS_{s!}^{-\nu}$ processes such that $P(\xrightarrow{\tau})^*Q$. Then, $Q \xrightarrow{\tau}$ if and only if in the corresponding Petri net N_P we have $Occur(Q)\triangleright$.*

With the help of the above propositions and corollaries we can now state that our Petri net encoding preserves and reflects convergence.

Lemma 4.2.1 (Convergence-invariant property) *For any $CCS_{s!}^{-\nu}$ process P, P converges if and only if the Petri net N_P converges.*

Proof.

- If P is convergent then $P(\xrightarrow{\tau})^*Q$ where $Q \not\xrightarrow{\tau}$. From Proposition 4.2.6 and Corollary 4.2.2, we have $Occur(P) \triangleright m_1 \ldots \triangleright m_n$ where $m_n = Occur(Q)$ and $Occur(Q)$ is a dead marking. Therefore, N_P is convergent.
- If N_P is convergent, then $Occur(P) \triangleright m_1 \ldots \triangleright m_n$ where m_n is a dead marking. From Proposition 4.2.8 there exists a process Q such that $m_n = Occur(Q)$ and $P(\xrightarrow{\tau})^*Q$. From Corollary 4.2.2 we know that $Q \not\xrightarrow{\tau}$, therefore P is convergent.

□

Since convergence is decidable for Petri nets [33], we conclude from the above lemma and our effective construction of Petri Nets that convergence is also decidable for $CCS_{s!}^{-\nu}$. Thus, from Propositions 4.2.2 and 4.2.3, we obtain the following corollary.

Theorem 4.2.1 *Convergence is a decidable property for $CCS_{!}^{-!\nu}$ processes.*

4.3 Decidability of Language Equivalence for $CCS_{!}^{-\nu}$

We now prove that decidability of language equivalence for $CCS_{!}^{-\nu}$. The crucial observation is that up to language equivalence every occurrence of a replicated process $!R$ in a $CCS_{!}^{-\nu}$ process can be replaced with $!\tau.0$ if R can perform at least an action, otherwise it can be replaced with 0. More precisely, let $P[Q/R]$ the process that results from replacing in P every occurrence of R with Q.

Proposition 4.3.1 *Let P be a CCS$_!^{-\nu}$ process and suppose that $!R$ occurs in P. Then $L(P) = L(P[Q/!R])$ where $Q = !\tau.0$ if there exists α s.t., $R \xrightarrow{\alpha}$ else $Q = 0$.*

Given any R in CCS$_!$ one can effectively decide whether there exists α such that $R \xrightarrow{\alpha}$, it can be proved by using the fact that the semantics is finitely-branching. We can then use the above proposition for proving the following statement.

Lemma 4.3.1 *Let P be a CCS$_!^{-\nu}$ process. One can effectively construct a process P' such that $L(P) = L(P')$ and P' is either $!\tau.0$ or a replication-free CCS$_!^{-\nu}$ process.*

Proof. Notice that we can use systematically Proposition 4.3.1 to transform any CCS$_!^{-\nu}$ process P into an language equivalent process Q whose replicated occurrences are all of the form $!\tau.0$. Now a $!\tau.0$ can occur either in a parallel composition, a summation or prefix process. Observe that (1) $P \mid !\tau.0 \sim_L !\tau.0$, (2) $!\tau.0 \mid P \sim_L !\tau.0$, (3) $\alpha.!\tau.0 \sim_L !\tau.0$, (4) $P + !\tau.0 \sim_L P$, (5) $!\tau.0 + P \sim_L P$. One can apply (1-5) from left to right to systematically transform Q into the process P' as required in the lemma. □

From the above lemma, we conclude that every CCS$_!^{-\nu}$ process can be effectively transformed into a language equivalent finite-state process. Hence,

Theorem 4.3.1 *Given P and Q in CCS$_!^{-\nu}$, the question of whether $L(P) = L(Q)$ is decidable.*

4.4 Impossibility results for failure-preserving encodings

In this section, we shall state the impossibility results about the existence of computable encodings from CCS$_!$ into CCS$_!^{-!\nu}$ and from CCS$_!^{-!\nu}$ into CCS$_!^{-\nu}$ which preserve and reflect failures equivalence. The separation results follow from our previous decidability results and the undecidability results in the literature.

The non-existence of failure-preserving encoding from $\text{CCS}_!$ into $\text{CCS}_!^{-!\nu}$ follows from Proposition 2.4.1, Theorem 4.2.1 and the undecidability of convergence for $\text{CCS}_!$ [22].

Theorem 4.4.1 *There is no computable function $[\![\cdot]\!] : CCS_! \to CCS_!^{-!\nu}$ s.t $[\![P]\!] \sim_F P$.*

Proof. As a means of contradiction, let us suppose that there is a computable encoding $[\![\cdot]\!] : \text{CCS}_! \to \text{CCS}_!^{-!\nu}$ s.t $[\![P]\!] \sim_F P$. By Proposition 2.4.1, $[\![P]\!]$ is convergent if and only if P is convergent, from Theorem 4.2.1 and the fact $[\![\cdot]\!]$ is computable, it is obtained that convergence is decidable for $\text{CCS}_!$. However, the undecidability of convergence for $\text{CCS}_!$ was shown in [22], a contradiction. □

To state the non-existence of failures-preserving encoding from $\text{CCS}_!^{-!\nu}$ into $\text{CCS}_!^{-\nu}$ we appeal to the undecidability of language equivalence for BPP processes [28, 46]. BPP processes form a subset of restriction-free CCS processes. Now we can use the encoding of [38] to transform a restriction-free CCS processes into $\text{CCS}_!^{-!\nu}$—the encoding is correct up to failures equivalence, Section 4.5 is devoted to prove the correctness of this encoding. We can therefore conclude, with the help of Proposition 2.4.2, that language-equivalence for $\text{CCS}_!^{-!\nu}$ processes is undecidable.

Proposition 4.4.1 *Given P and Q in $CCS_!^{-!\nu}$, the problem of whether $P \sim_L Q$ is undecidable.*

Proof.

As a means of contradiction, let us assume that given two $\text{CCS}_!^{-!\nu}$ processes P and Q, the problem of whether $P \sim_L Q$ is decidable. Let us consider any two BPP processes P' and Q'. By using the encoding $[\![\cdot]\!]$ of [38] to transform a restriction-free CCS processes into $\text{CCS}_!^{-!\nu}$, we can translate P' and Q' into two $\text{CCS}_!^{-!\nu}$ processes $[\![P']\!]$ and $[\![Q']\!]$ respectively. By Proposition 2.4.2 and Theorem 4.5.2, it followed that $P' \sim_L [\![P']\!]$ and $Q' \sim_L [\![Q']\!]$. From the fact that $[\![\cdot]\!]$ is computable, the problem of whether $P' \sim_L Q'$ is decidable. However, it contradicts the undecidability of language equivalence for BPP processes [28, 46]. □

From the above proposition, the decidability of language equivalence for CCS$_!^{-\nu}$ (Theorem 4.3.1) and Proposition 2.4.2 we can conclude the following.

Theorem 4.4.2 *There is no computable function* $[\![\cdot]\!]$: *CCS*$_!^{-!\nu}$ $\rightarrow$ *CCS*$_!^{-\nu}$ *s.t.* $[\![P]\!] \sim_F P$.

Proof. As a means of contradiction, let us suppose that there is a computable function $[\![\cdot]\!]$: CCS$_!^{-!\nu}$ $\rightarrow$ CCS$_!^{-\nu}$ s.t. $[\![P]\!] \sim_F P$. By Proposition 2.4.2, the problem of whether two CCS$_!^{-!\nu}$ processes P and Q are language equivalent can be transformed into the problem of whether $[\![P]\!]$ and $[\![Q]\!]$ are language equivalent. By Theorem 4.3.1, we obtain that the problem of whether two CCS$_!^{-!\nu}$ processes P and Q are language equivalent is decidable. But it contradicts Proposition 4.4.1. □

In the following section we shall show that restriction-free CCS processes can be translated into CCS$_!^{-!\nu}$ reflecting and preserving failures equivalence. In particular, we use the encoding proposed in [83] to translate π-calculus with recursive definitions into π-calculus with replication and we show that this is correct up to failures equivalence.

4.5 Encoding from CCS$^{-\mu\nu}$ into CCS$_!^{-\nu}$

In this section we shall show an encoding from CCS without restrictions within constant definitions, henceforth called CCS$^{-\mu\nu}$, into CCS$_!^{-!\nu}$ preserving and reflecting failures equivalence. Therefore, BPP, a restriction-free fragment of CCS, can be translated into $ccsrTwo$ up to failures equivalence. Thus extending the undecidability of languages equivalence for BPP to CCS$_!^{-!\nu}$. Another consequence of the correctness of the encoding up to failures equivalence is that convergence is also decidable for CCS$^{-\mu\nu}$.

Except for not including $!P$, CCS$^{-\mu\nu}$ extends the syntax of CCS$_!$ as follows:

$$P, Q \ldots := \ldots \mid A \tag{4.1}$$

Here A is an constant (also call, or invocation). We assume that every such an identifier has a unique, possibly recursive, definition $A \stackrel{\texttt{def}}{=} P_A$. and the intuition is that a call A behaves as its body P_A. Processes in CCS$^{-\mu\nu}$ are those defined by using the syntax above which do not have occurrences of a process of the form $(\nu x)P$ within the body of a constant.

The operational rules for CCS$^{-\mu\nu}$ are those in Table 2.3 plus the following rule:

$$\text{TR-CONS}\frac{P_A \stackrel{\alpha}{\longrightarrow} P'}{A \stackrel{\alpha}{\longrightarrow} P'} \text{ if } A \stackrel{\texttt{def}}{=} P_A$$

We make a typical assumption on calls in CCS$^{-\mu\nu}$: they need to be guarded in the body of constant definitions. We say that an expression is guarded in P if only if it lies within some sub-expression of P of the form $\alpha.Q$.

Convention 4.5.1 *We shall confine ourselves to CCS$^{-\mu\nu}$ processes where all the calls appearing in the bodies of constant definitions are guarded.*

First, we introduce the encoding from CCS$^{-\mu\nu}$ into CCS$_!^{-!\nu}$, and then we shall prove that the encoding preserves and reflects failures equivalence.

4.5.1 The Encoding

The main idea of this encoding is to associate a replicated process $!x.P'$ to each variable definition and any occurrence of the variables in the process term could be modelled as a name which communicates with its corresponding replicated process. To do this, a local name could be helpful to guarantee the proper behaviour. The encoding is basically the same used in [83].

Definition 4.5.1 *Let $[\![\cdot]\!]$: CCS$^{-\mu\nu}$ $\longrightarrow$ CCS$_!^{-!\nu}$ be an encoding function that is homomorphic over all operators in the sub-calculus defining finite behaviour and is otherwise defined as follows:*

$$\begin{aligned} [\![X_i]\!]_{aux} &= \overline{x_i}.0 \\ [\![X_i \stackrel{\texttt{def}}{=} P_i]\!]_{aux} &= !x_i.[\![P_i]\!]_{aux} \end{aligned}$$

Where each x_i is a fresh name.

Let $[\![\cdot]\!] : CCS^{-\mu\nu} \longrightarrow CCS_!^{-!\nu}$ be the encoding (main) function which translates a CCS$^{-\mu\nu}$ process P where $\{X_i \stackrel{\text{def}}{=} P_i : i = 1, 2, \ldots, n\}$ is its corresponding set of constant definitions, as follows:

$$[\![P]\!] \;=\; \nu x_1, x_2, \ldots, x_n([\![P]\!]_{aux} \mid [\![X_1 \stackrel{\text{def}}{=} P_1]\!]_{aux} \mid \ldots \mid [\![X_n \stackrel{\text{def}}{=} P_n]\!]_{aux})$$

In comparing CCS$^{-\mu\nu}$ and CCS$_!^{-!\nu}$, we find it convenient to consider another variant calculus, as an intermediate step, which we call CCS$_\tau^{-\mu\nu}$. The syntax of CCS$_\tau^{-\mu\nu}$ agrees entirely with CCS$^{-\mu\nu}$. Its transition relation $\longrightarrow_\tau$ is obtained by replacing $\longrightarrow$ with $\longrightarrow_\tau$ in the rules in Table 2.3 and by adding the following rule:

$$\text{TR-APP}\frac{}{A \stackrel{\tau}{\longrightarrow}_\tau P_A} \text{ if } A \stackrel{\text{def}}{=} P_A$$

Rule TR-APP performs a τ-action when unfolding A's definition, hence the sub-index τ.

In order to witness an application of TR-APP, we introduce a relation $\longrightarrow_{TR-APP}$. Define $P \longrightarrow_{TR-APP} Q$ iff the derivation of $P \longrightarrow_\tau Q$ includes an application of Rule TR-APP. Similarly, we define $P \longrightarrow_{NTR-APP} Q$ iff $P \not\longrightarrow_{TR-APP} Q$, i.e., the derivation of $P \longrightarrow_\tau Q$ does not include applications of TR-APP.

As CCS$_\tau^{-\mu\nu}$ can be seen as a restriction of the π-calculus with constants definitions, referred π^c in [83], by requiring all inputs and outputs to have empty subjects only and without local names definitions and the encoding above is the same as the one defined in [83] for translating from π^c into π-calculus. The bisimilarity property for the encoding showed in [83] also holds for CCS$_\tau^{-\mu\nu}$.

Proposition 4.5.1 *[83] For $P \in CCS_\tau^{-\mu\nu}$ we have $P \sim [\![P]\!]$.*

As corollary from Proposition 4.5.1 and Proposition 2.4.3 we have the following result:

Theorem 4.5.1 *Let P be a CCS$_\tau^{-\mu\nu}$ process, $P \sim_F [\![P]\!]$.*

It remains to study the relation between $\text{CCS}^{-\mu\nu}$ and $\text{CCS}_{\tau}^{-\mu\nu}$. We shall prove that a process P in both $\text{CCS}^{-\mu\nu}$ and $\text{CCS}_{\tau}^{-\mu\nu}$ has the same failures.

First of all, we shall prove that $\text{CCS}_{\tau}^{-\mu\nu}$ preserves the failures of the processes in $\text{CCS}^{-\mu\nu}$.

Lemma 4.5.1 *For $P \in CCS^{-\mu\nu}$, if $P \stackrel{s}{\Longrightarrow} Q$ then $P \stackrel{s}{\Longrightarrow}_{\tau} Q$.*

Proof.

By induction on the number of transitions.

□

Convention 4.5.2 *From now on, we shall represent with $P\{P_{A_1}/A_1, P_{A_2}/A_2, P_{A_3}/A_3, \ldots, P_{A_n}/A_n\}$ the process resulting after applying the rule $TR - APP$ to all the unguarded occurrences of the constants $A_1, A_2, \ldots, A_n$ in the $CCS_{\tau}^{-\mu\nu}$ process P.*

The next two lemmas show that a $\text{CCS}^{-\mu\nu}$ process can be equated to the respective process in $\text{CCS}_{\tau}^{-\mu\nu}$ once Rule TR-APP has been applied as much as possible, in thes sense, both perform the same immediate actions.

Lemma 4.5.2 *Given a process $P \in CCS^{-\mu\nu}$ whose constants are $A_1, A_2, \ldots, A_n$, there exists Q such that $P \stackrel{\epsilon}{\Longrightarrow}_{\tau} Q$ where $Q = P\{P_{A_1}/A_1, \ldots, P_{A_n}/A_n\}$.*

Proof.

It is straightforward from the repetitive application of Rule TR-APP until reaching a process without unguarded occurrences of the constants. Notice that it is possible to reach this process as $\text{CCS}_{\tau}^{-\mu\nu}$ is guarded recursive, i.e., there is no unguarded occurrences of a constant in the body of the constants.

□

Lemma 4.5.3 *Given a process $P \in CCS^{-\mu\nu}$ whose constants are $A_1, A_2, A_3, \ldots, A_n$, $\{\alpha \mid P \stackrel{\alpha}{\longrightarrow}\} = \{\alpha \mid P\{P_{A_1}/A_1, \ldots, P_{A_n}/A_n\} \stackrel{\alpha}{\longrightarrow}_{\tau}\}$.*

Proof.

It is straightforward from Rules TR-CONS and TR-APP in CCS$^{-\mu\nu}$ and CCS$_\tau^{-\mu\nu}$ respectively and the fact that CCS$^{-\mu\nu}$ (CCS$_\tau^{-\mu\nu}$) is guarded.

□

From now on, $Failures_\tau(P)$ represents the set of failures of a CCS$_\tau^{-\mu\nu}$ process P. The failures for CCS$_\tau^{-\mu\nu}$ are defined in the usual way but using $\longrightarrow_\tau$ instead of $\longrightarrow$.

Lemma 4.5.4 *Given a process $P \in$ CCS$^{-\mu\nu}$, $Failures(P) \subseteq Failures_\tau(P)$.*

Proof.

Let $\langle s, L\rangle$ be a failure in P. By definition of failure, there must be a process Q whose constants are $A_1, A_2, A_3, \ldots, A_n$ such that $P \stackrel{s}{\Longrightarrow} Q$ and $\{\alpha \mid Q \stackrel{\alpha}{\longrightarrow}\} \cap (\ L \cup \{\tau\}\ ) = \emptyset$. By Lemmata 4.5.1 and 4.5.2, $P \stackrel{s}{\Longrightarrow}_\tau Q \stackrel{\epsilon}{\Longrightarrow}_\tau Q\{P_{A_1}/A_1, \ldots, P_{A_n}/A_n\}$. Finally considering Lemma 4.5.3, it is clear that $\{\alpha \mid Q\{P_{A_1}/A_1, \ldots, P_{A_n}/A_n\} \stackrel{\alpha}{\longrightarrow}_\tau\} \cap (\ L \cup \{\tau\}\ ) = \emptyset$, therefore $\langle s, L\rangle \in Failures_\tau(P)$ as well.

□

Now, it is left to prove that extending CCS$^{-\mu\nu}$ into CCS$_\tau^{-\mu\nu}$ does not introduce new failures.

Lemma 4.5.5 *Given a process $P \in$ CCS$^{-\mu\nu}$, $Failures_\tau(P) \subseteq Failures(P)$.*

Proof.

Let $\langle s, L\rangle \in Failures_\tau(P)$. By definition of failure, there must be a process Q such that $P \stackrel{s}{\Longrightarrow}_\tau Q$ and $\{\alpha \mid Q \stackrel{\alpha}{\longrightarrow}\} \cap (\ L \cup \{\tau\}\ ) = \emptyset$. It is possible to generate an evolution from P into Q, where $P \stackrel{s}{\Longrightarrow}_\tau Q$, just by reordering the actions such that a τ-action from Rule TR-APP is to be performed only immediately before the actions which are enabled once the τ-action has been performed. The actions which do not depend on τ-actions from the application of Rule TR-APP are performed at the beginning and the τ-actions involving the application of Rule TR-APP whose enabled actions are not involved in the evolution are left at the end. Considering this, the evolution can be seen as follows:

$$P(\xrightarrow{\alpha}_{NTR-APP})^{+} \xrightarrow{\tau}_{TR-APP} (\xrightarrow{\alpha}_{NTR-APP})^{+} \dots$$

$$\dots \xrightarrow{\tau}_{NTR-APP} (\xrightarrow{\alpha}_{NTR-APP})^{+} P'(\xrightarrow{\tau}_{TR-APP})^{*} Q$$

Where $\xrightarrow{\tau}_{TR-APP}$ denotes the τ- action involving the rule TR-APP and $Q = P'\{P_{A_1}/A_1, \dots, P_{A_n}/A_n\}$ where $A_1, A_2, A_3, \dots, A_n$ are the constants in P'.

It is easy to see that by using Rule TR-CONS instead of the rule TR-APP in the previous evolution, it is possible to generate an evolution from P into P' in CCS$^{-\mu\nu}$ such that $P \stackrel{s}{\Longrightarrow} P'$. From $Q = P'\{P_{A_1}/A_1, \dots, P_{A_n}/A_n\}$ and by using Lemma 4.5.3 we conclude that $\{\alpha \mid Q \xrightarrow{\alpha}\} \cap (\ L \cup \{\tau\}\ ) = \emptyset$, hence $\langle s, L\rangle \in Failures(P)$.

□

As a direct consequence of the fact that CCS$_{\tau}^{-\mu\nu}$ preserves failures-equivalence and Theorem 4.5.1, it is shown that $[\![\cdot]\!]$ preserves failures-equivalence .

Theorem 4.5.2 (Encoding preserving failures-equivalence) *Let P be a CCS$^{-\mu\nu}$ process, $P \sim_F [\![P]\!]$.*

From the correctness of the encoding up to $\sim_F$ we can state the decidability results for $\sim_F$ in CCS$_{!}^{-!\nu}$ and convergence in CCS$^{-\mu\nu}$ as follows:

Theorem 4.5.3 *$\sim_F$ is undecidable in CCS$_{!}^{-!\nu}$.*

Proof. It is straightforward from Theorem 4.5.2 and the fact that failures equivalence is undecidable for CCS$^{-\mu\nu}$, this latter is a consequence of the decidability of failures equivalence for BPP [48, 52].

□

By Proposition 2.4.1, Theorem 4.2.1 and Theorem 4.5.2 we obtain the following corollary.

Theorem 4.5.4 *Convergence is a decidable property for CCS$^{-\mu\nu}$.*

4.6 Expressiveness of Priorities

In this section we add Phillips' priority guards [76] to $\text{CCS}_!^{-!\nu}$. We shall refer to the resulting calculus as $\text{CCS}_{!+pr}^{-!\nu}$. This calculus corresponds to Phillips' Calculus of Priority Guards (CPG) with replication rather than recursion and no restrictions within the scope of replication—hence it cannot use an unbounded number of restrictions.

We show that $\text{CCS}_{!+pr}^{-!\nu}$ turns out to be Turing powerful in the sense of Busi et al [22] (i.e., preserving and reflecting convergence), thus bearing witness to computational expressiveness of priority guards. Recall that from the previous sections $\text{CCS}_!$, and even CCS, cannot encode Turing machines, in the sense above, without using an unbounded number of restrictions (Theorem 4.2.1 and Theorem 4.5.4).

4.6.1 $\text{CCS}_!$ with priorities

In CPG there are two sets of names: N which corresponds to the set of names used to represent the visible actions in $\text{CCS}_!^{-!\nu}$ and a set of priority names U. Each set has a set of complementary actions : $\bar{N}$ and $\bar{U}$, where $Std = N \cup \bar{N}$ (the standard visible actions), $Pri = U \cup \bar{U}$ (the priority actions), $Vis = Std \cup Pri$ (the visible actions), and $Act = Vis \cup \tau$ (all actions). We let $a, b, \ldots$ range over $N \cup U$; u, v, $\ldots$ over Pri; λ, $\ldots$ over Vis; and α, β, $\ldots$ over Act. Also S, T, $\ldots$ range over finite subsets of Vis, and U, V, $\ldots$ over finite subsets of Pri.

The syntax of processes in $\text{CCS}_{!+pr}^{-!\nu}$ is like that of $\text{CCS}_!^{-\nu}$, except for the summations which now take the form of *priority-guarded* summations: $\Sigma_{i\in I} S_i : \alpha_i.P_i$ where I and each S_i are finite. The meaning of the priority guard $S : \alpha$ is that α can only be performed if the environment does not offer any action in $\bar{S} \bigcap Pri$ (see [76] for details).

Labelled Transition and Offers

We recall the set $off(P)$ of "higher priority" actions "offered" by P.

Definition 4.6.1 (Offers) *Let P be a $\text{CCS}_{!+pr}^{-!\nu}$ process and $u \in Pri$. The relation*

P off u (P offers u) is given by the rules in Table 4.1 . We define $off(P) = \{u \in Pri : P \ \ off \ \ u\}$. Finally, we say that P eschews *U iff $off(P) \cap \bar{U} = \emptyset$.*

$$M + S : u.P + N \ off \ u \quad if \quad u \notin S$$

$$\frac{P \ \ off \ \ u}{P \mid Q \ \ off \ \ u} \qquad \frac{Q \ \ off \ \ u}{P \mid Q \ \ off \ \ u}$$

$$\frac{P \ \ off \ \ u}{(\nu a) \ \ P \ \ off \ \ u} \ if \ \ a \neq name(u) \qquad \frac{P \ \ off \ \ u}{!P \ \ off \ \ u}$$

Table 4.1:

The transitions are conditional on offers from the environment. Intuitively, a transition of the form $P \xrightarrow{\alpha}_U P'$ means that P may perform α as long as the environment does not offer $\bar{u}$ for any $u \in U$ (i.e., the environment "eschews" U).

Example 4.6.1 *The transition $a : b.P \xrightarrow{b}_{\{a\}} P$ means that $a : b.P$ may perform b as long as the environment does not offer $\overline{a}$. Thus, $a : b.P \mid \overline{b}.Q$ could evolve into $P \mid Q$, i.e. $a : b.P \mid \overline{b}.Q \xrightarrow{\tau}_{\{a\}} P \mid Q$, however the system $a : b.P \mid \overline{b}.Q \mid \overline{a}$ could not evolve into $P \mid Q \mid \overline{a}$, i.e. $a : b.P \mid \overline{b}.Q \mid \overline{a} \not\xrightarrow{\tau}_{\{a\}} P \mid Q \mid \overline{a}$, as the presence of $\overline{a}$ prevents the execution of b and thus the τ-action resulting from $(b, \overline{b})$ communication.*

This capability of processes to test the presence or the absence of a channel ready to be performed will be fundamental to represent the test for $zero$ in the encoding of RAMs in $CCS^{-!\nu}_{!+pr}$ presented in the next subsection. Transitions are determined by the rules in Table 4.2.

Convention 4.6.1 *We write $P \xrightarrow{\alpha}_{\emptyset} P'$ as $P \xrightarrow{\alpha} P'$ (i.e., α is not constrained on offers from the environment thus corresponding to a standard $CCS_!$ transition). Thus, the notions of divergence and convergence for $CCS^{-!\nu}_{!+pr}$ are obtained as in Definition 2.4.9 by replacing $\xrightarrow{\tau}$ with $\xrightarrow{\tau}_{\emptyset}$.*

4.6.2 Encoding RAMs with priorities

Now we shall show that $CCS^{-!\nu}_{!+pr}$ is Turing powerful by providing an encoding from Random Access Machines into $CCS^{-!\nu}_{!+pr}$ preserving and reflecting divergence and convergence.

$$\text{SUM}\ M + S : \alpha.P + N \xrightarrow{\alpha}_{S \cap Pri} P \ \ if \ \ \alpha \ \ \in S \cap Pri$$

$$\text{PAR}_1\ \frac{P \xrightarrow{\alpha}_U P' \ \ Q \ \ eschews \ \ U}{P \mid Q \xrightarrow{\alpha}_U P' \mid Q} \qquad \text{PAR}_2\ \frac{Q \xrightarrow{\alpha}_U Q' \ \ P \ \ eschews \ \ U}{P \mid Q \xrightarrow{\alpha}_U P \mid Q'}$$

$$\text{REACT}\ \frac{P \xrightarrow{\lambda}_{U_1} P' \quad Q \xrightarrow{\overline{\lambda}}_{U_2} Q' \ \ P \ \ eschews \ \ U_2 \ \ Q \ \ eschews \ \ U_1}{P \mid Q \xrightarrow{\tau}_{U_1 \cup U_2} P' \mid Q'}$$

$$\text{REP}\ \frac{P \mid !P \xrightarrow{\alpha}_U P'}{!P \xrightarrow{\alpha}_U P'}$$

$$\text{RES}\ \frac{P \xrightarrow{\alpha}_U P' \ \ if \ \ \alpha \ \ \notin \{a, \overline{a}\}}{(\nu\, a)P \xrightarrow{\alpha}_{U - \{a,\overline{a}\}} (\nu\, a)P'}$$

$$\text{SUM}\ \frac{}{\Sigma_{i \in I} \alpha_i.P_i \xrightarrow{a_j} P_j}\ if\ j \ \in \ I$$

Table 4.2: An operational semantics for $\text{CCS}^{-!\nu}_{!+pr}$.

The Encoding. A register r_j with value c_j (written $r_j : c_j$) is modeled by a corresponding number of processes of the form $\overline{u}_j$.

$$[\![(r_j : c_j)]\!] \ = \ \prod_{1}^{c_j} \overline{u}_j$$

The program counter is modeled with the *absence* of $\overline{p}_i$ (i.e., the action p_i is eschwed by the encoding) indicating that the i-th instruction is the next to be executed. The initial value of the program counter is 1 so by using $\prod_{i=2}^{m+1} \overline{p}_i$ we indicate the absence of $\overline{p}_1$.

The increasing instruction is modelled with a process $[\![(i : Succ(r_j))]\!]$ which is guarded by a τ-action which is only performed when there is an absence of $\overline{p}_i$.

$$[\![(i : Succ(r_j))]\!] \ = !(\{p_i\} : \tau.(\overline{p_i} \mid p_{i+1} \mid \overline{u}_j))$$

Once activated, the instruction increases the register r_j by offering $\overline{u}_j$, and goes to the next instruction by both disallowing the current one by offering $\overline{p}_i$ and allowing the next one by performing p_{i+1} so that $\overline{p}_{i+1}$ can be consumed.

The decreasing instruction is defined similarly. In addition we consider the absence of $\overline{u}_j$ to test for zero.

$$[\![(i : DecJump(r_j, l))]\!] = !(\{p_i\} : u_j.(\overline{p_i} \mid p_{i+1}))|!(\{p_i, u_j\} : \tau.(\overline{p_i} \mid p_l))$$

The encoding of a RAM is given below.

Definition 4.6.2 *Let R be a RAM with program instructions* $(1 : I_1), \ldots, (m : I_m)$ *and registers* $r_1, \ldots, r_n$*. Given the configuration* $(i, c_1, \ldots, c_n)$ *we define its encoding into* $CCS^{-!\nu}_{!+pr}$ *as:*

$[\![(i, c_1, \ldots, c_n)]\!]_R = (\nu p_1, \ldots, p_{m+1}, u_1, \ldots, u_n)(\prod_{j=1}^{m} [\![(j : I_j)]\!] \mid \prod_{j=1}^{n} [\![(r_j : c_j)]\!] \mid \prod_{j=1}^{i-1} \overline{p}_j \mid \prod_{j=i+1}^{m+1} \overline{p}_j)$

In order to prove formally the correctness of the encoding. We will reason up-to the following structural congruence $\equiv_R$ used to remove terminated processes equal to 0 as well as for the reordering of processes in parallel compositions. Formally, $\equiv_R$ is the least congruence relation satisfying the following axioms:

$P \mid 0 \equiv_R P$

$P \mid Q \equiv_R Q \mid P$

$P \mid (Q \mid R) \equiv_R (P \mid Q) \mid R$

It is easy to see that $\equiv_R$ preserves the operational semantics of $\text{CCS}^{-!\nu}_{!+pr}$ up to strong bisimulation. It is a direct consequence of Lemma 5.3 in [76] where $\equiv$ was proved to respect the transitions of CPG, which uses recursion instead of replication, and the fact that replication can be encoded by using recursive definitions up to strong bisimulation.

Corollary 4.6.1 *Let* $P, Q \in CCS^{-!\nu}_{!+pr}$ *with* $P \sim \equiv_R \sim Q$*. If* $P \xrightarrow{\alpha}_U P'$ *then there exists* Q' *such that* $Q \xrightarrow{\alpha}_U Q'$ *and* $P' \sim \equiv_R \sim Q'$*.*

In the rest of this section, we will only focus on computations of τ-actions and we will say that a computation $P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n[\ldots]$ is deterministic if there is no other possible computation (of τ-actions) from P_1. Notice that this implies that there is no other possible computation (of τ-actions) from each P_i where $i \geq 1$.

The following proposition shows that the encoding of RAM behaves deterministically.

Proposition 4.6.1 *Let R be a RAM with program instructions* $(1 : I_1), \ldots, (m : I_m)$ *and registers* $r_1, \ldots, r_n$. *Given a configuration* $(i, c_1, \ldots, c_n)$ *of R, we have that, if* $i > m$ *then* $[\![(i, c_1, \ldots, c_n)]\!]_R$ *is a stable process, otherwise:*

1. *if* $(i, c_1, \ldots, c_n) \longrightarrow_R (i', c'_1, \ldots, c'_n)$ *then we have* $[\![(i, c_1, \ldots, c_n)]\!]_R \xrightarrow{\tau}{}^{+} \equiv_R [\![(i', c'_1, \ldots, c'_n)]\!]_R$

2. *there exists a non-zero length deterministic computation* $[\![(i, c_1, \ldots, c_n)]\!]_R \xrightarrow{\tau} Q_1 \xrightarrow{\tau} Q_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} \equiv_R [\![(i', c'_1, \ldots, c'_n)]\!]_R$ *such that* $(i, c_1, \ldots, c_n) \longrightarrow_R (i', c'_1, \ldots, c'_n)$.

Proof.

It is immediate to see that $[\![(i, c_1, \ldots, c_n)]\!]_R$ is stable if $i > m$, because all processes (that compose it by means of parallel) are stuck on inputs or τ-actions that can not be triggered as they can perform as long as the environment does not offer $\overline{p_i}$ for some $1 \leq i \leq m$.

Otherwise, if $i \leq m$, let us suppose $(i, c_1, \ldots, c_n) \longrightarrow_R (i', c'_1, \ldots, c'_n)$. We have two cases:

- If I_i is a $Succ(r_j)$ instruction, the process $[\![(i, c_1, \ldots, c_n)]\!]_R$ proceeds deterministically by performing a reduction sequence composed of two reduction steps that leads to a process which is structurally equivalent to $[\![(i', c'_1, \ldots, c'_n)]\!]_R$: the first reduction corresponds to the performing of a τ-action from $[\![(i : Succ(r_j))]\!]$, the second one is caused by the synchronisation on p_{i+1}. Thus both statements 1 and 2 are satisfied.

- If I_i is a $DecJump(r_j, l)$ instruction, we have two sub-cases depending on $c_j = 0$ or $c_j \geq 0$.

  - If $c_j = 0$ then $[\![(i, c_1, \ldots, c_n)]\!]_R$ proceeds deterministically by performing a reduction sequence composed of two reduction steps that leads to a process which is structurally equivalent to $[\![(i', c'_1, \ldots, c'_n)]\!]_R$: the first reduction corresponds to the performing of a τ-action from $[\![(i : DecJump(r_j, l))]\!]$, the second one is caused by the synchronisation on p_{i+1}. Thus both statements 1 and 2 are satisfied.

– If $c_j > 0$ then $[\![(i, c_1, \ldots, c_n)]\!]_R$ proceeds deterministically by performing a reduction sequence composed of two reduction steps that leads to a process which is structurally equivalent to $[\![(i', c'_1, \ldots, c'_n)]\!]_R$: the first reduction is caused by a synchronisation on u_j, the second one by a synchronisation on p_{i+1}.

□

Theorem 4.6.1 *Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the initial configuration $(i, 0, \ldots, 0)$ of R we have that R terminates if and only if the process $[\![(1, 0, \ldots, 0)]\!]_R$ converges.*

Proof.

As a means of contradiction, assume that the RAM computation does not terminate and the corresponding encoding converge. However by using Proposition 4.6.1 (statement 1) it is immediate to see that the encoding has an infinite computation and by statement 2 we obtain that this computation is the only one possible, so the encoding of the RAM is not convergent, a contradiction.

Assume now that RAM terminates. By using induction on the length of the computation of RAM, the statement 1 and the first part of Proposition 4.6.1, we prove that the encoding reaches a process structural congruent to $[\![(m, c'_1, \ldots, c'_n)]\!]_R$, so this reached process is a stable one. Thus obtaining the convergence of the encoding.

□

As the process $[\![(1, 0, \ldots, 0)]\!]_R$ generated by the encoding is deterministic, we have that it converges if and only if it does not diverge (i.e., all computations are terminating). Henceforth, also divergence is undecidable.

4.7 Summary and Related Work

In this chapter, we studied the expressiveness of restriction and its interplay with replication in $\text{CCS}_!$. We proved that Turing expressiveness in the sense of [22] is lost when not using restriction under the scope of replication. We also showed

that even more expressive power is lost when considering only restriction-free processes. We have proved and/or used the decidability of properties such as convergence, language equivalence and failures equivalence to establish these negative results. Finally, we showed the expressive power of priorities by providing an encoding of RAMs in $\text{CCS}^{-!\nu}_{!+pr}$ preserving and reflecting convergence and divergence. This implies that $\text{CCS}^{-!\nu}_{!+pr}$ can not be encoded into $\text{CCS}_{!}$. Similar to [21], we gave formal evidence that recursion cannot be replaced by replication when considering fragments of the π-calculus without mobility. In particular, we showed that a restriction-free fragment of the π-calculus with recursive definitions without mobility and without synchronisation (BPP) is more expressive than a restriction-free fragment of the π-calculus with replication without mobility ($\text{CCS}^{-\nu}_{!}$).

The work in [21, 22] was already discussed in Section 4.1. In [38] the authors study replication and recursion in CCS focusing on the role of restriction and name scoping. In particular they show that $\text{CCS}_{!}$ is equivalent to CCS with recursion with *static scoping*. The standard CCS is shown to have *dynamic* scoping precisely because the use of restriction within recursive definitions. However, if no restriction appears within recursive expressions then there is no distinction between static and dynamic scoping. Hence, if no restriction is allowed within recursive expressions then we know from [38] that CCS can be encoded in $\text{CCS}_{!}$, without restriction under replication, while preserving and reflecting convergence. As for the other direction, clearly $\nu X.(P|X)$ behaves as $!P$. Nevertheless, if recursion is required to be *prefix guarded*, it is not clear how to produce an encoding which preserves and reflects convergence—without appealing to the decidability results for $\text{CCS}_{!}$ here presented. Consider e.g., $E = \nu X.(P|\alpha.X)$ and $!P$. If $\alpha = \tau$ then E does not converge and $!P$ may—take $P = a.0$. If $\alpha \neq \tau$ then E may converge and $!P$ may not—take $P = \tau.0$.

The authors in [29] also pointed out the role of restriction in the expressiveness of CCS. They showed that strong bisimilarity is decidable for restriction-free CCS, in contrast with the undecidability result for CCS [87]. It is not clear to us how to relate strong bisimilarity with convergence or failures equivalence.

The authors of [6] studied a fragment of the asynchronous π-calculus with restricted forms of bound name generation. A closely related result in of [6] is the

decidability of the control reachability problem for restriction-free asynchronous π-calculus. This implies the decidability of the same problem for the restriction-free fragment of asynchronous $CCS_!$ (i.e., only 0 can be prefixed with an output action). It is not obvious how to relate control reachability to failures equivalence or convergence. Also it is not clear how to encode our $CCS_!$ fragment into restriction-free asynchronous $CCS_!$.

In [42] a Petri net semantics is proposed for a subset of CCS without restriction and with guarded choice. Also in [87] it was shown that the subset studied in [42] can not be extended significantly. These works also presuppose guarded recursion in their fragments which seem crucial for their Petri net constructions. We do not restrict our Petri net construction to guarded sums. Furthermore, as explained above, it is not clear how to translate $CCS_!$ into CCS with guarded recursion while preserving convergence.

In [93] the authors show the decidability of convergence for a restriction-free calculus for the compositional description of chemical systems, called *CFG* which seems closely related to CCS. The calculus, however, presupposes guarded summation and guarded recursion and thus, as argued before, it is not clear how to encode $CCS_!$ into such a calculus while preserving convergence.

As for related work dealing with the expressiveness of priorities in Process Calculi, in [3] it was shown that the priority operator of Baeten, Bergstra and Klop cannot be expressed using positive rule formats for operational semantics, we think that this inexpressibility result could be valid for other priorities operators due to the non-monotonicity of these kind of operators. In [76] it was shown that priorities add expressive power to CCS by modelling electoral systems that cannot be modelled in CCS. Also [91] studies two process algebras enriched with different priority mechanisms. The work reveals the gap between the two prioritised calculi and the two non prioritised ones by modeling electoral systems. Both [76] and [91] state the impossibility of the existence of an encoding subject to certain structural requirements such as homomorphism wrt parallel composition and name invariance. Our derived impossibility result about the non-existence of convergent preserving encodings makes no structural assumptions on the encodings. Finally, we claim that our expressivity results involving priorities are also held by using other priority approaches as they provide the capability of processes

to know if another process is ready to perform a synchronisation on some channel or not.

The results in this chapter were originally published as [8].

Part II

Chapter 5

Linearity, Persistence and Testing Semantics in the Asynchronous Pi-Calculus

In this chapter we continue our study on the expressive power of variants of the π-calculus, in particular of a significant subcalculus: the asynchronous π-calculus (Aπ). As before, we consider the special role of replication and properties sensitive to infinite computations as an issue to compare the expressiveness in different subcalculi of Aπ.

In [70] the authors studied the expressiveness of persistence in the asynchronous π-calculus (Aπ) wrt weak barbed congruence. The study is incomplete because it ignores the issue of divergence. In this chapter we shall present an expressiveness study of persistence in the asynchronous π-calculus (Aπ) wrt De Nicola and Hennessy's testing scenario, which is sensitive to divergence.

Following [70], we consider Aπ and three sub-languages of it, each capturing one source of persistence: the *persistent-input* calculus (PIAπ), the *persistent-output* calculus (POAπ) and the *persistent* calculus (PAπ). In [70] the authors showed encodings from Aπ into the semi-persistent calculi (i.e., POAπ and PIAπ) correct wrt weak barbed congruence.

In this chapter we prove that, under some general conditions, there cannot be an encoding from Aπ into a (semi)-persistent calculus preserving the $must$ testing

semantics.

The separation result between Aπ and its semi-persistent subcalculi in Section 5.5.1 and the separation result between Aπ and PAπ in Section 5.6 were published as [25].

The separation results between Aπ and its semi-persistent subcalculi in Section 5.5.2 and the decidability of convergence and divergence for POAπ in Section 5.7 have not been published.

5.1 Introduction

In [70] the authors present an expressiveness study of linearity and persistence of processes. Since several calculi presuppose persistence on their processes, the authors address the expressiveness issue of whether such persistence restricts the systems that we can specify, model or reason about in the framework. Their work is conducted using the standard notion of weak barbed congruence and hence it ignores divergence issues. Since divergence plays an important role in expressiveness studies, particularly in those studies involving persistence, in this work we aim at extending and strengthening their study by using the standard notion of testing equivalences. As elaborated below, our technical results contrast and complement those in [70]. More importantly, our results also clarify and support informal expressiveness claims in the literature.

Linearity is present in process calculi such as CCS, CSP, the π-calculus [62] and Linear CCP [85, 34], where messages are consumed upon being received. In the π-calculus the system $\bar{x}z \mid x(y).P \mid x(y).Q$ represents a message with a datum z, tagged with x, that can be *consumed* by either $x(y).P$ or $x(y).Q$. *Persistence of messages* is present in several process calculi. Perhaps the most prominent representative of such calculi is Concurrent Constraint Programming (CCP) [84]. Here the messages (or items of information) can be read but, unlike in Linear CCP, they cannot be consumed. Other prominent examples can be found in the context of calculi for analyzing and describing security protocols: Crazzolara and Winskel's SPL [31], the Spi Calculus variants by Fiore and Abadi [35] and by Amadio et all [5], and the calculus of Boreale and Buscemi [13] are operationally defined in terms of configurations containing messages which cannot

be consumed. *Persistent receivers* arise, e.g. in the notion of *omega receptiveness* [82], where the input of a name is always available—but always with the same continuation. In the π-calculus persistent receivers are used, for instance, to model functions, objects, higher-order communications, or procedure definitions. Furthermore, persistence of *both* messages and receivers arise in the context of CCP with universally-quantified persistent ask operations. In the context of calculi for security, persistent receivers can be used to specify protocols where principals are willing to run an unbounded number of times (and persistent messages to model the fact that every message can be remembered by the spy). In fact, the approach of specifying protocols in a persistent setting, with an unbounded number of sessions, has been explored in [12] by using a classic logic Horn clause representation of protocols (rather than a linear logic one).

Expressiveness of Persistence: Drawbacks and Conjectures. The study in [70] is conducted in the *asynchronous* π-calculus (Aπ), which naturally captures the persistent features mentioned above. Persistent messages (and receivers) can simply be specified using the *replication* operator of the calculus which creates an unbounded number of copies of a given process. In particular, the authors in [70] investigate the existence of encodings from Aπ into three sub-languages of it, each capturing one source of persistence: the *persistent-input* calculus (PIAπ), defined as Aπ where inputs are replicated; the *persistent-output* calculus (POAπ), defined dually, i.e. outputs rather than inputs are replicated; the *persistent* calculus (PAπ), defined as Aπ but with all inputs and outputs are replicated.

The main result in [70] basically states that we need one source of linearity, i.e. either on inputs (PIAπ) or outputs (POAπ) to encode the behavior of arbitrary Aπ processes via weak barbed congruence.

Nevertheless, the main drawback of the work [70] is that the notion of correct encoding is based on weak barbed bisimulation (congruence), which is not sensitive to *divergence*. In particular, the encoding provided in [70] from Aπ into PIAπ is weak barbed congruent preserving but not divergence preserving. Although in some situations divergence may be ignored, in general it is an important issue to consider in the correctness of encodings [26, 44, 43, 55, 24].

In fact, the informal claims of extra expressivity of Linear CCP over CCP in

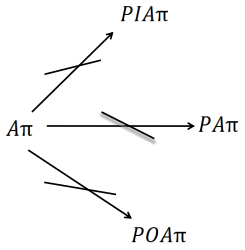
[11, 34] are based on discrimination introduced by divergence that is clearly ignored by the standard notion of weak bisimulation. Furthermore, the author of [30] suggests as future work to extend SPL, which uses only persistent messages and replication, with recursive definitions to be able to program and model recursive protocols such as those in [4, 73]. One can, however, give an encoding of recursion in SPL from an easy adaptation of the composition between the Aπ encoding of recursion [83] (where recursive calls are translated into *linear* Aπ outputs and recursive definitions into persistent inputs) and the encoding of Aπ into POAπ in [70]. The resulting encoding is correct up-to weak bisimulation. The encoding of Aπ into POAπ, however, introduces divergence and hence the composite encoding does not seem to invalidate the justification for extending SPL with recursive definitions. The above works suggest that the expressiveness study of persistence is relevant but incomplete if divergence is not taken into account.

In this chapter we shall therefore study the existence of encodings from Aπ into the persistent sub-languages mentioned above using testing semantics [65] which takes divergence into account.

5.1.1 Contributions

Our main contribution is to show formally that (semi-) persistent subcalculi of A$\pi$are not as expressive as Aπ. It is done by providing a uniform and general negative result stating that, under some reasonable conditions, Aπ cannot be encoded into any of the above (semi-) persistent calculi while preserving the $must$ testing semantics (Section 5.5). The general conditions involve compositionality on the encoding of constructors such as parallel composition, prefix, and replication. The main result contrasts and completes the ones in [70]. Furthermore, we prove that convergence and divergence are decidable in POAπ (hence in PAπ) unlike Aπ (Section 5.7). The decidability is a direct consequence of the absence of continuation of the persistent output prefixes in POAπ. Thus, strengthening the separation result between Aπ and POAπ.

It also supports the informal claims of extra expressivity mentioned above. We shall also state other more specialized impossibility results for must preserving encodings from Aπ into the semi-persistent calculi, focusing on specific proper-



	Aπ	POAπ	PIAπ	PAπ
Decid. Converg	no	yes	?	yes
Decid. Diverg.	no	yes	?	yes

Figure 5.1: Separation results between Aπ and its semi-persistent calculi (PIAπ, POAπ, PAπ). A crossed arrow from C to C' represents the non-existence of an encoding from C to C' preserving the $must$ testing semantics . The table summarises the decidability results for Aπ and its semi-persistent subcalculi.

ties of each target calculus. This helps clarifying some previous assumptions on the interplay between syntax and semantics in encodings of process calculi. We believe that, since the study is conducted in Aπ with well-established notions of equivalence, we can easily adapt our results to other asynchronous frameworks such as CCP languages and the above-mentioned calculi for security.

The contributions are summarized in Figure 5.1.1.

Remark 5.1.1 *Convergence and divergence are undecidable in Aπ. In [21], it was proved the undecidability of convergence and divergence for CCS. That result is extended directly to Aπ by using the encoding from π-calculus with recursive functions into replication showed in [83], the encoding from guarded-choice π-calculus into choice-free π-calculus given in [64], and either Honda and Tokoro's encoding or Boudol's encoding from π-calculus into Aπ proposed in [47] and [15] respectively. All of these encodings preserve and reflect divergence and convergence.*

Outline of this chapter. The remainder of this chapter is organized as follows. In Section 5.2, we present the semi-persistent subcalculi we are going to compare.

In Section 5.3 we recall some properties of encodings. In Section 5.4 we recall the encodings in [70] and study them by using Testing Semantics. Section 5.5 is the core of this chapter, we present the separability results between Aπ and its semi-persistent subcalculi by showing the non-existence of encodings preserving must-testing. In Sections 5.6 and 5.7 we provide some specialized separation results between Aπ and PAπ and Aπ and POAπ by showing the non-existence of a larger class of encodings from Aπ into PAπ and the decidability of convergence and divergence in POAπ. In Section 5.8 we conclude by summarising and discussing some related work.

5.2 Semi-persistence in Aπ

Here we define the syntactic restrictions of Aπ. The reader may find it useful to look at the notions and notations given in Sections 2.2 and 2.4.

5.2.1 The (semi-)persistent calculi

The *persistent-input* calculus PIAπ results from Aπ by requiring all input processes to be replicated. Processes in PIAπ are generated by the following grammar:

$$P, Q, \ldots \quad := \quad 0 \;\Big|\; !\,x(y).P \;\Big|\; \bar{x}y \;\Big|\; P \mid Q \;\Big|\; (\nu x)P \;\Big|\; !\,P$$

The *persistent-output* calculus POAπ arises as from Aπ by requiring all outputs to be replicated. Processes in POAπ are generated by the following grammar:

$$P, Q, \ldots \quad := \quad 0 \;\Big|\; x(y).P \;\Big|\; !\,\bar{x}y \;\Big|\; P \mid Q \;\Big|\; (\nu x)P \;\Big|\; !\,P$$

Finally, we have the *persistent* calculus PAπ , a subset of Aπ where output and input processes must be replicated. Processes in PAπ are generated by the following grammar:

$$P, Q, \ldots \quad := \quad 0 \;\Big|\; !\,x(y).P \;\Big|\; !\,\bar{x}y \;\Big|\; P \mid Q \;\Big|\; (\nu x)P \;\Big|\; !\,P$$

The relation $\xrightarrow{\alpha}$ for PIAπ, POAπ and PAπ can be equivalently defined as in Table 2.2, with Input replaced with Input(PIAπ), Output replaced with Output(POAπ), and Input and Output replaced with Input(PIAπ) and Output(PIAπ) rules respectively. (Table 5.1). The new rules reflect the *persistent-input* nature of PIAπ (Rule Input(PIAπ)), the *persistent-output* nature of POAπ

(Rule Output(POAπ)), and the *persistent* nature of PAπ (Rules Input and Output(PAπ)). Notice that these new rules can be derived directly from the application of the Rules Input, Output, and Rep-Act in Table 2.2.

Input(PIAπ)	$!x(y).P \xrightarrow{xz} P\{z/y\} \mid !x(y).P$ where $x,\ y \in \mathcal{N}$
Output(POAπ)	$!\,\bar{x}z \xrightarrow{\bar{x}y} 0 \mid !\,\bar{x}z$
Input(PAπ)	$!x(y).P \xrightarrow{xz} P\{z/y\} \mid !x(y).P$ where $x,\ y \in \mathcal{N}$
Output(PAπ)	$!\,\bar{x}z \xrightarrow{\bar{x}y} 0 \mid !\,\bar{x}z$

Table 5.1: Transition Rules.

Notation 5.2.1 *We shall use $\mathcal{P}$ to range over the set of the calculi in this chapter $\{A\pi, PIA\pi, POA\pi, PA\pi\}$.*

5.3 Reasonable Properties of Encodings

As mentioned earlier, an encoding is a mapping from the terms of a calculus into the terms of another. In general a "good" encoding satisfies some additional requirements, but as discussed in the introduction there is no agreement on a general notion of "good" encoding. Perhaps indeed there should not be a unique notion, but several, depending on the purpose.

In this section we shall introduce and justify the requirements used in the forthcoming sections.

5.3.1 Contexts, Compositionality and Homomorphism

Let us begin by recalling the notion of (multi-hole) process *contexts* [83] to describe compositionality.

Recall that a $\mathcal{P}$ *context* C with k holes is a term with occurrences of k distinct *holes* $[\,]_1, \ldots, [\,]_k$ such that a $\mathcal{P}$ process must result from C if we replace all the occurrences of each $[\,]_i$ with a $\mathcal{P}$ process. The context C is *singularly-structured* if each hole occurs exactly once. For example, $[\,]_1 \mid x(y).([\,]_2 \mid [\,]_1)$ is an Aπ non singularly-structured context with two holes. Given $P_1, \ldots, P_k \in \mathcal{P}$ and a

context C with k holes, $C[P_1, \ldots, P_k]$ is the process that results from replacing the occurrences of each $[\,]_i$ with P_i. The names of a context C with k holes, $n(C)$, are those of $C[Q_1, \ldots, Q_k]$ where each Q_i is 0. The free and bound names of a context are defined analogously. We can regard the input prefix $x(y)$, $|$ and $!$ as the operators of arity 1, 2 and 1 respectively in Aπ in the obvious sense.

Definition 5.3.1 (Compositionality w.r.t. an operator) Let *op* be an n-ary operator of Aπ. An encoding $[\![\cdot]\!] : \text{A}\pi \to \mathcal{P}$ is *compositional* w.r.t. *op* iff there is a $\mathcal{P}$ context C_{op} with n holes such that $[\![op(P_1, .., P_n)]\!] = C_{op}[[\![P_1]\!], .., [\![P_n]\!]]$.

Furthermore, given an encoding $[\![\cdot]\!] : \text{A}\pi \to \mathcal{P}$, we define ${C_{op}}^{[\![\cdot]\!]}$ as the context C such that $[\![op(P_1, \ldots, P_n)]\!] = C[[\![P_1]\!], \ldots, [\![P_n]\!]]$. We shall often omit the "$[\![\cdot]\!]$" in ${C_{op}}^{[\![\cdot]\!]}$ since it is easy to infer from the context. $C_{op}[\cdot]$ refer to the context encoding the operator op with a hole with any number of occurrences. $C_{op}[P]$ will refer to the context encoding the operator op and P represents the term placed in every occurrence of the hole in the context. Thus, for example $C_{a(x)}[[\![\bar{a}z]\!]]$ $= [\![a(x).\bar{a}z]\!]$ if the encoding is compositional wrt input prefix. $C_{a(x)}[0]$ is the resulting process after replacing every occurrence of the hole in $C_{a(x)}[\cdot]$ with 0. Notice that $C_{a(x)}[P]$ and $C_{a(x)}[[\![P]\!]]$ are not the same as the in the first case the hole is replaced with P and in the second case it is replaced with $[\![P]\!]$, if the encoding is compositional wrt input prefix $C_{a(x)}[[\![P]\!]] = [\![a(x).P]\!]$.

An interesting case of compositionality is *homomorphism* w.r.t a given operator op.

Definition 5.3.2 (Homomorphism) *Let op be an n-ary operator of Aπ. An encoding $[\![\cdot]\!] : \text{A}\pi \to \mathcal{P}$ is homomorphic w.r.t. op iff $[\![op(P_1, \ldots, P_n)]\!] = op([\![P_1]\!], \ldots, [\![P_n]\!])$.*

It is worth pointing out that homomorphism w.r.t parallelism, also called distribution-preserving [92], can arguably be considered as a reasonable requirement for an encoding. In particular, the works [92, 69, 27, 43, 44] support the distribution-preserving hypothesis by arguing that it corresponds to requiring that the degree of distribution of the processes is maintained by the translation, i.e. no coordinator is added. Some of these works are in the context of solving electoral problems and some others in more general scenarios [43, 44]. Other works

[64, 79], however, argue that the requirement can be quite demanding as it rules out practical implementation of distributed systems.

Some of our impossibility results will appeal to the distribution-preserving hypothesis. If we accept this hypothesis then it is also reasonable to require homomorphism for replication, since ! represents an infinite parallel composition. In the following section, however, we will argue for weaker, and probably less controversial, requirements over replication.

5.3.2 Preservation of infinite behaviour

Now we introduce a requirement involving replication, named preservation of infinite behaviour w.r.t !. From our point of view, this requirement is rather natural and relies on the ability to preserve the behaviour of !, i.e., the ability of replication to provide certain behaviour unlimitedly. Since $!P$ can be seen as an unbounded number of copies of P, therefore the behaviour associated to P would be offered permanently . Furthermore, this requirement could be used to compare a language involving ! with other language with another mechanism to simulate infinite behaviour, e.g , the encoding of replication into recursion showed in [83] preserves infinite behaviour wrt replication as presented in Definition 5.3.3.

Definition 5.3.3 (Preservation of infinite behaviour wrt !) An encoding $[\![\cdot]\!] : \mathrm{A}\pi \to \mathcal{P}$ preserves infinite behaviour wrt ! iff:

- If $[\![P]\!] \stackrel{\alpha}{\Longrightarrow} Q$ for some process Q and $\alpha \neq \tau$ then $[\![!P]\!] \stackrel{\alpha}{\Longrightarrow} Q$ for some process Q
- If $[\![!P]\!] \stackrel{\alpha}{\Longrightarrow} Q$ for some process Q and $\alpha \neq \tau$ then $[\![!P]\!] \stackrel{\alpha}{\Longrightarrow} \equiv P' \mid [\![!P]\!]$ for some process P'.

The intuition behind the first item in Definition 5.3.3 is that an encoding should take into account that $!P$ is able to interact with the environment as P does. Therefore all the observable actions (after an arbitrary number of τ-actions) of $[\![P]\!]$ should be present in $[\![!P]\!]$. As for the second item, the idea is that the persistent behaviour associated to $!P$, which is inherent of replication, should be preserved at least partially by the encoding. Therefore a process encoding $!P$, i.e. $[\![!P]\!]$, should be preserved along its evolution.

Remark 5.3.1 *It is worth noticing that an encoding with homomorphism w.r.t* !*, i.e.* $[\![!P]\!] = ![\![P]\!]$*, satisfies the requirements in Definition 5.3.3 since trivially* $[\![P]\!] \Downarrow_{\bar{x}}$ *iff* $![\![P]\!] \Downarrow_{\bar{x}}$ *and* $![\![P]\!] \xrightarrow{\alpha} Q \mid ![\![P]\!]$*. Clearly, the same happens when the encoding is homomorphic wrt* ! *up to* $\equiv$*.*

With preservation of infinite behaviour, we want to capture preservation of the behaviour of $!P$. If $!P \xrightarrow{\alpha} Q$ with $\alpha \neq \tau$ then $Q \equiv R \mid !P$ and hence we see that the infinite behaviour is preserved after any interaction of the process with its environment, thus any observable action remains able to be performed, i.e., *observable infinite behaviour*. Now, if $!P \xrightarrow{\tau} Q$ and $\alpha = \tau$ then we also have $Q \equiv R \mid !P$ so the infinite behaviour is also preserved after any internal action, i.e. the internal actions will also be present along the evolution, i.e., *internal infinite behaviour*. Homomorphism wrt replication preserves both kinds of infinite behaviour. Encodings satisfying the property from Definition 5.3.3 preserve a weaker form of observable infinite behaviour.

In the following sections we shall study the existence of encodings $[\![\cdot]\!] : \mathrm{A}\pi \to \mathcal{P}$ from π into $\mathcal{P} \in \{\mathrm{PA}\pi, \mathrm{PIA}\pi, \mathrm{POA}\pi\}$ satisfying some of the properties described in this section and the standard testing semantics defined in Section 2.4.3.

As mentioned earlier in the introduction to this chapter, the focus on testing semantics is due to its treatment of divergency which was ignored in previous work.

5.4 Previous encodings of Aπ into semi-persistent subcalculi

In this section we shall state some properties of existing encodings of Aπ into its semi-persistent subcalculi. Let us recall the following encoding from Aπ to PIAπ, defined in [70].

Definition 5.4.1 *[70] The encoding* $[\![\cdot]\!] : A\pi \to PIA\pi$ *is a homomorphism for* 0*, parallel composition, restriction and replication, otherwise is defined as*

- $[\![\bar{x}z]\!] = \bar{x}z$*, and*

- $[\![x(y).P]\!] = (\nu tf)(\bar{t} \mid !x(y).(\nu l)(\bar{l} \mid !t.!l.([\![P]\!] \mid !\bar{f}) \mid !f.!l.\bar{x}y))$

  where $t, f, l \notin fn(P) \cup \{x, y\}$. *(The lifted version is given adding* $[\![\omega.P]\!] = \omega.[\![P]\!]$.*)*

In [70], it was shown that this encoding enjoys a strong property: namely, for any P, $[\![P]\!] \approx_a P$,

Proposition 5.4.1 *[70] Let* $[\![\cdot]\!] : A\pi \rightarrow PIA\pi$ *as in Definition 5.4.1. For any* $A\pi$ *process* P, $[\![P]\!] \approx_a P$.

Proposition 5.4.1 implies, in a testing scenario, that a process passes a test if and only if the encoding of the process passes the encoding of the test as long as may and fair testing are considered. First, we need to consider ω as a barb.

Remark 5.4.1 *Let us extend the notion of barb in Definition 2.4.4 to include* ω*, where* ω*, in contrast to the rest of the barbs (co-names), has the following restrictions :*

- *It can be only present in processes of* $\mathcal{O}$.
- *It does not have a corresponding co-action, i.e., it is not possible to synchronise via* ω.
- *It can not be the subject of an action, therefore it can not be transmitted.*

As a straightforward consequence, $P \xrightarrow{\omega}$ *and* $P \downarrow_\omega$ *coincide.*

Proposition 5.4.2 *For any* P *in* $\mathcal{O}$, $P \xrightarrow{\omega}$ *if and only if* $P \downarrow_\omega$.

We can now prove that the encoding given [70] is both "may" and "fair" preserving.

Proposition 5.4.3 *Let* $[\![\cdot]\!] : A\pi \rightarrow PIA\pi$ *as in Definition 5.4.1.* $\forall\, P \in \mathrm{A}\pi$, $\forall\, o \in \mathcal{O}$ P *sat* o iff $[\![P]\!]$ *sat* $[\![o]\!]$, where *sat* can be respectively *may* and *fair* .

Proof.

First, we show that $[\![\cdot]\!]$ is may-preserving. Consider any $P \in \mathrm{A}\pi$ and any $o \in \mathcal{O}$. We know that $P \mid o \dot{\approx}_a [\![P \mid o]\!]$ by Proposition 5.4.1. As a consequence of Definition 2.4.5 and the fact that $[\![\cdot]\!]$ is homormorphic wrt parallelism we have that $(P \mid o) \Downarrow_\omega$ iff $([\![P]\!] \mid [\![o]\!]) \Downarrow_\omega$. Hence, there exists a maximal computation:

$$P \mid o = T_o \mid o_0 \xrightarrow{\tau} T_1 \mid o_1 \xrightarrow{\tau} T_2 \mid o_2 \xrightarrow{\tau} \ldots T_i \mid o_i \xrightarrow{\tau} \ldots$$

such that $T_i \mid o_i \downarrow_\omega$, for some $i \geq 0$ if and only if there exists a maximal computation:

$$[\![P]\!] \mid [\![o]\!] = T'_o \mid o'_0 \xrightarrow{\tau} T'_1 \mid o'_1 \xrightarrow{\tau} T'_2 \mid o'_2 \xrightarrow{\tau} \ldots T'_j \mid o'_j \xrightarrow{\tau} \ldots$$

such that $T'_i \mid o'_i \downarrow_\omega$, for some $i \geq 0$. By Proposition 5.4.2 and Definition 2.4.12 we have that $P \; may \; o$ iff $[\![P]\!] \; may \; [\![o]\!]$.

As for fair-testing, let us suppose P *fair* o. Then for every maximal computation $P \mid o = E_0 \xrightarrow{\tau} E_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} E_i \, [\xrightarrow{\tau} \ldots]$ we have $E_i \Longrightarrow E'_i \downarrow_\omega$, for every $i \geq 0$. Since $P \mid o \dot{\approx}_a [\![P \mid o]\!] = [\![P]\!] \mid [\![o]\!]$, every process Q such that $[\![P]\!] \mid [\![o]\!] \; (\xrightarrow{\tau})^* Q$ is asynchronous barbed bisimilar to at least one process reachable from $P \mid o$ by τ-actions. Therefore for every maximal computation $[\![P]\!] \mid [\![o]\!] = A_0 \xrightarrow{\tau} A_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} A_i \, [\xrightarrow{\tau} \ldots] \; A_i \Downarrow_\omega$, for every $i \geq 0$. I.e. $[\![P]\!]$ *fair* $[\![o]\!]$.

□

To see that the above statement is not satisfied in the case of $must$ semantics, consider $P = (a.0 \mid !\bar{a})$ and $o = a.\omega$. We have $P \; must \; o$ but $[\![P]\!] \; \not{must} \; [\![o]\!]$.

In [70] the encoding in Definition 5.4.1 is used to get an encoding of Aπ into POAπ, by composing it with the following mapping from PIAπ into POAπ.

Definition 5.4.2 The encoding $f = [\![\cdot]\!] : \mathrm{PIA}\pi \to \mathrm{POA}\pi$ is a homomorphism for 0, parallel composition, restriction, and replication, otherwise is defined as

- $[\![\bar{x}z]\!] = (\nu s)(!\bar{x}s \mid s(r).!\bar{r}z)$, and
- $[\![!x(y).P]\!] = !x(s).(\nu r)(!\bar{s}r \mid r(y).[\![P]\!])$

where $s, r \notin fn(P) \cup \{x, z\}$. (The lifted version is given adding $[\![\omega.P]\!] = \omega.[\![P]\!]$.)

Let g be $[\![\cdot]\!] : \mathrm{A}\pi \rightarrow \mathrm{PIA}\pi$ in Definition 5.4.1. The encoding $h = [\![\cdot]\!] : \mathrm{A}\pi \rightarrow \mathrm{POA}\pi$ is the composite function $f \circ g$.

Since this encoding maps a linear output into a replicated one with the same barb, the composite encoding $h = [\![\cdot]\!] : \mathrm{A}\pi \rightarrow \mathrm{POA}\pi$ in Definition 5.4.2 does not satisfy $[\![P]\!] \approx_a P$. It has a weaker property: namely, $P \approx_a Q$ iff $[\![P]\!] \approx_a \underset{[\cdot]}{\mathrm{POA}_\pi}[\![Q]\!]$, where $[\![P]\!] \approx_a \underset{[\cdot]}{\mathrm{POA}_\pi}[\![Q]\!]$ means that $\forall C$ context in $\mathrm{A}\pi$, $[\![C]\!][\![P]\!]$ and $[\![C]\!][\![Q]\!]$ (assuming $[\![[\cdot]]\!] = [\cdot]$) are weak barbed bisimilar [83].

As for the testing scenario, the above encoding satisfies the following:

Proposition 5.4.4 Let $h = [\![\cdot]\!] : \mathrm{A}\pi \rightarrow \mathrm{POA}\pi$ as in Definition 5.4.2. $\forall\, P \in \mathrm{A}\pi$, $\forall\, o \in \mathcal{O}$, P *sat* o if and only if $[\![P]\!]$ *sat* $[\![o]\!]$, where *sat* can be respectively *may* and *fair*.

Proof. Similar to the proof of the *may* (*fair*)-preservation of Boudol's encoding from π into $\mathrm{A}\pi$ in [23]. □

The above proposition would not hold if *sat* were *must*. Consider $P = !\bar{a}$ and $o = a.\omega$: then P *must* o but $[\![P]\!]$ *must* $[\![o]\!]$.

5.5 Uniform impossibility results for the semi-persistent calculi

In the previous section we stated that previous encodings from $\mathrm{A}\pi$ into the semi-persistent subcalculi are not must preserving. We shall now demonstrate that under some general conditions such encodings must not exist.

This section is the core of the chapter and it focuses on general and uniform negative results for encodings of $\mathrm{A}\pi$ into $\mathrm{PIA}\pi$, $\mathrm{POA}\pi$ and $\mathrm{PA}\pi$. We identify some reasonable conditions which will guarantee that none of these encodings can be must-preserving. Namely, compositionality, homomorphism wrt replication, and preservation of infinite behaviour wrt replication.

Besides them we use some additional hypothesis such as $[\![\omega]\!] \xrightarrow{\omega}$ and $fn([\![0]\!]) = 0$ which we also consider reasonable. The condition $[\![\omega]\!] \xrightarrow{\omega}$ seems to be reasonable as it can follow from the existence of a divergent process in the range of the encoding, which is necessary if the encoding preserves divergence—recall that P *diverges*, $P\uparrow$, if there is an infinite sequence of reductions from P. However, the hypothesis $[\![\omega]\!] \xrightarrow{\omega}$ can be also obtained in a purely syntactic way, i.e without divergence. Now, the condition $fn([\![0]\!]) = 0$ since it is natural that 0, the process which does nothing, is translated in a way such that it does not express any external behaviour.

We shall show two general impossibility results from Aπ into PIAπ, POAπ and PAπ, differing slightly in their hypotheses. The results relies on the fact that in a testing scenario, i.e. an observer and a test in parallel, any encoding introduces divergence. It arises as a translation forces some kind of action, either input or output, to be persistent, thus an interaction between a linear component and a persistent component turns out to be an interaction between two persistent components, hence the introduction of divergence. This divergence can delay indefinitely the execution of some actions in the target language which were expected to be performed in order to be consistent with the behaviour of the original process in the source language. In term of testing semantics it means that there is no must-preserving encoding as the divergence introduced in the testing scenario by the encoding delays indefinitely the report of success, i.e the observation of ω.

Roughly speaking, the impossibility results in this section will use the fact that for any encoding $[\![\cdot]\!]$ from Aπ into a semi-persistent calculi we have $C_!{[}C_{x(y)}[0]{]} \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$ or $C_{x(y)}[0] \mid C_!{[}[\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$, i.e for any term T, $[\![!x(y).T]\!]|[\![\bar{x}z]\!]|\bar{x}'z'[\![\uparrow$ or $[\![x(y).T]\!]|[\![!(!\bar{x}z|\bar{x}'z')]\!] \uparrow$ if the encoding is compositional. In this way any such encoding would introduce divergent behaviour that can be used to delay an action which can be expected to be performed. Thus replacing T with $[\![x'z'.\omega]\!]$ we can prove that $[\![!x(y).x'(y').\omega]\!]|[\![\bar{x}z]\!]|\bar{x}'z'[\![$ or $[\![x(y).x'(y').\omega]\!]|[\![!(!\bar{x}z|\bar{x}'z')]\!]$ can delay forever an action that may *unguard* ω—i.e., causing ω to become observable. Consequently, we must obtain $[\![!x(y).x'(y').\omega]\!]$ / *must* $[\![\bar{x}z]\!]|\bar{x}'z'[\![$ or $[\![x(y).x'(y').\omega]\!]$ *m/ust* $[\![!(!\bar{x}z|\bar{x}'z')]\!]$.

In the first general impossibility result (Section 5.5.1) , we show that there does not exist a must-preserving compositional encoding, homomorphic wrt repli-

cation, from π-calculus into any semi-persistent calculus, assuming that $[\![\omega]\!] \xrightarrow{\omega}$.

In the second general impossibility result (Section 5.5.2), we prove a similar result to the one in Section 5.5.1 but by considering a weaker notion: preservation of infinite behaviour wrt replication. However, additionally it is assumed that $fn([\![0]\!]) = 0$.

In the next two subsections we show the two general impossibility results mentioned above in that order.

5.5.1 Non-existence of encodings homomorphic wrt !

In order to prove the impossibility result wrt must-preservation, we first prove that the encoding introduces divergent behaviours. As said before, must-preservation is sensitive to divergence.

We first need an auxiliary result about the occurrence of the hole in $C_{x(y)}[\cdot]$ being prefixed.

Definition 5.5.1 *We say that the hole in the context $C[\cdot]$ is prefixed iff every occurrence of $[\,]$ in $C[\cdot]$ is within a context of the form $x(y).C'[\cdot]$.*

Proposition 5.5.1 Let $[\![\cdot]\!] : \mathrm{A}\pi \to \mathcal{P} \in \{\mathrm{PIA}\pi, \mathrm{POA}\pi, \mathrm{PA}\pi\}$ be an encoding satisfying:

1. compositionality w.r.t. input prefix,
2. must-preservation,
3. $[\![\omega]\!] \xrightarrow{\omega}$.

Then $\forall x, y \in \mathcal{N}$, the hole in $C^{[\![\cdot]\!]}_{x(y)}$ is prefixed.

Proof. By definition we have that $0 \not\mathit{must}\ x(y).\omega$, and since $[\![\cdot]\!]$ is must-preserving, we have that $[\![0]\!] \not\mathit{must}\ [\![x(y).\omega]\!]$. Hence, $[\![0]\!] \not\mathit{must}\ C_{x(y)}[[\![\omega]\!]]$ by compositionality wrt input prefix. Since $[\![\omega]\!] \xrightarrow{\omega}$ by hypothesis, every occurrence of $[\![\omega]\!]$ has to be prefixed in $C_{x(y)}[[\![\omega]\!]]$. □

We can now prove that any must-preserving, compositional wrt input prefix, homomorphic wrt ! encoding must introduce divergence under the natural

assumption that $[\![\omega]\!] \xrightarrow{\omega}$. This is done by showing that $\forall x, y, z, x', z' \in \mathcal{N}$, $C_!{[C_{x(y)}[0]]} \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$ or $C_{x(y)}[0] \mid C_!{[[\![\bar{x}z \mid \bar{x}'z']\!]]} \uparrow$.

Lemma 5.5.1 Let $[\![\cdot]\!] : \mathrm{A}\pi \to \mathcal{P} \in \{\mathrm{PIA}\pi, \mathrm{POA}\pi, \mathrm{PA}\pi\}$ be an encoding satisfying:

1. compositionality wrt input prefix,
2. homomorphism wrt replication,
3. must-preservation,
4. $[\![\omega]\!] \xrightarrow{\omega}$,

Then $\forall x, y, z, x', z' \in \mathcal{N}$, $C_!{[C_{x(y)}[0]]} \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$ or $C_{x(y)}[0] \mid C_!{[[\![\bar{x}z \mid \bar{x}'z']\!]]} \uparrow$

Proof.

By homomorphism wrt !, $C_!{[C_{x(y)}[0]]} \mid [\![\bar{x}z \mid \bar{x}'z']\!] = !{[C_{x(y)}[0]]} \mid [\![\bar{x}z \mid \bar{x}'z']\!]$ and $C_{x(y)}[0] \mid C_!{[[\![\bar{x}z \mid \bar{x}'z']\!]]} = C_{x(y)}[0] \mid !{[[\![\bar{x}z \mid \bar{x}'z']\!]]}$. We consider different cases according to the behaviour of the encoded processes:

- $C_{x(y)}[0] \xrightarrow{\tau}$ or $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{\tau}$: Trivially $!{[C_{x(y)}[0]]} \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$ if $C_{x(y)}[0] \xrightarrow{\tau}$, and $C_{x(y)}[0] \mid !{[[\![\bar{x}z \mid \bar{x}'z']\!]]} \uparrow$ if $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{\tau}$.
- $C_{x(y)}[0] \not\xrightarrow{\tau}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \not\xrightarrow{\tau}$: As the encoding is must-preserving we know that $[\![x(y).\omega]\!] \ must \ [\![\bar{x}z \mid \bar{x}'z']\!]$. Therefore by compositionality wrt input prefix

$$(\star)\ [\![x(y).\omega]\!] = C_{x(y)}[[\![\omega]\!]] \ must \ [\![\bar{x}z \mid \bar{x}'z']\!].$$

  Since the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) and $C_{x(y)}[0] \not\xrightarrow{\tau}$ we conclude that $C_{x(y)}[[\![\omega]\!]] \not\xrightarrow{\tau}$ and $C_{x(y)}[[\![\omega]\!]] \not\xrightarrow{\omega}$. From this together with $(\star)$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \not\xrightarrow{\tau}$ we conclude that there must be at least one interaction between $C_{x(y)}[[\![\omega]\!]]$ and $[\![\bar{x}z \mid \bar{x}'z']\!]$, i.e., $C_{x(y)}[[\![\omega]\!]] \mid [\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{\tau}$ where either $C_{x(y)}[[\![\omega]\!]] \xrightarrow{\bar{x}''z''}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{x''z''}$, $C_{x(y)}[[\![\omega]\!]] \xrightarrow{x''z''}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{\bar{x}''z''}$, $C_{x(y)}[[\![\omega]\!]] \xrightarrow{\bar{x}''(z'')}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{x''z''}$, or $C_{x(y)}[[\![\omega]\!]] \xrightarrow{x''z''}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{\bar{x}''(z'')}$. Here we consider the first two cases, the others are similar.

– If $C_{x(y)}[[\![\omega]\!]] \xrightarrow{\bar{x}''z''}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{x''z''}$: Now let us consider the possible target calculi: if $\mathcal{P} =$ PIAπ, then $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(\bar{x}''z''|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \equiv (\nu z_1)..(\nu z_m)(!x''(y'').P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. Since the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) and $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(\bar{x}''z''|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ we can conclude that $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_n)(\bar{x}''z''|Q'')$ where $\forall i \in [1..m], x'' \neq z_i$. Therefore $![C_{x(y)}[0]] \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$. If $\mathcal{P} =$ POAπ, then $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!\bar{x}''z''|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \equiv (\nu z_1)..(\nu z_m)(x''(y'').P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. We can again use Proposition 5.5.1 and $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!\bar{x}''z''|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ to verify that $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_n)(!\bar{x}''z''|Q'')$ where $\forall i \in [1..m], x'' \neq z_i$. Therefore $C_{x(y)}[0] \mid ![[\![\bar{x}z \mid \bar{x}'z']\!]] \uparrow$.The case $\mathcal{P} =$ PAπ is analogous to (and easier than) the previous two cases.

– If $C_{x(y)}[[\![\omega]\!]] \xrightarrow{x''z''}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \xrightarrow{\bar{x}''z''}$: Now let us consider the possible target calculi: if $\mathcal{P} =$ PIAπ, then $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!x''(y'').P|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \equiv (\nu z_1)..(\nu z_m)(\bar{x}''z''.P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. Since the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) and $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!x''(y'').P|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ we can conclude that $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_n)(!x''(y'').P''|Q'')$ where $\forall i \in [1..m], x'' \neq z_i$, therefore $C_{x(y)}[0] \mid ![[\![\bar{x}z \mid \bar{x}'z']\!]] \uparrow$. If $\mathcal{P} =$ POAπ, then $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(x''(y'').P|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \equiv (\nu z_1)..(\nu z_m)(!\bar{x}''z''.P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. We can again use the fact the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) and $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(x''(y'').P|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ to conclude that $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_n)(x''(y'').P''|Q'')$ where $\forall i \in [1..n], x'' \neq z_i$, therefore $![C_{x(y)}[0]] \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$. The case $\mathcal{P} =$ PAπ is analogous to (and easier than) the previous two cases.

□

Now, with the help of Lemma 5.5.1 and Proposition 5.5.1 we can prove the first impossibility result in this Section. We show that there is no encoding from

Aπ into PIAπ, POAπ and PAπ by assuming compositionality, homomorphism wrt replication, $[\![\omega]\!] \xrightarrow{\omega}$ and must-preservation:

Theorem 5.5.1 Let $[\![\cdot]\!] : \text{A}\pi \to \mathcal{P}$, with $\mathcal{P} \in \{\text{PIA}\pi, \text{POA}\pi, \text{PA}\pi\}$, be an encoding satisfying:

1. compositionality wrt. input prefix,
2. homomorphism wrt replication,
3. $[\![\omega]\!] \xrightarrow{\omega}$.

Then $[\![\cdot]\!]$ is not must-preserving.

Proof.

Let $C[\cdot] = C_!'[C_{x(y)}[\cdot]] \mid [\![\bar{x}z \mid \bar{x}'z']\!]$ and $C'[\cdot] = C_{x(y)}[\cdot] \mid C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]]$. By Lemma 5.5.1, for any x, y, z x', $z' \in \mathcal{N}$, $C[0] \uparrow$ or $C'[0] \uparrow$. Now if $C[0]$ or $C'[0]$ may diverge with 0 replacing the hole, then $C[T]$ or $C'[T]$ may also diverge with a term T replacing the hole without T being ever involved in the generation of the divergency. Therefore, there is at least one divergent computation from $C[T]$ or $C'[T]$ where any term prefixed in T remains prefixed along the computation. By Proposition 5.5.1 $[\![\omega]\!]$ is prefixed in $T = C_{x'(y')}[[\![\omega]\!]]$. Hence, there is at least one maximal infinite computation of $C_!'[C_{x(y)}[T]] \,|\, [\![\bar{x}z \,|\, \bar{x}'z']\!]$ or $C_{x(y)}[T] \,|\, C_!'[[\![\bar{x}z \,|\, \bar{x}'z']\!]]$ where any occurrence of ω is prefixed, thus ω is not observable. Consequently, by compositionalilty wrt input prefix and replication and the definition of must-preservation $[\![!x(y).x'(y').\omega]\!]$ *must* $[\![\bar{x}z \mid \bar{x}'z']\!]$ or $[\![x(y).x'(y').\omega]\!]$ *must* $[\![!\bar{x}z \mid \bar{x}'z']\!]$ even if $x \neq x'$. Therefore $[\![\cdot]\!]$ is not must-preserving.

□

5.5.2 Non-existence of encodings preserving infinite behaviour

We now show that the assumption on homomorphism wrt replication can be weakened by using the property described in Definition 5.3.3 instead (Theorem 5.5.2).

Our separation result in this section states that there is no must-preserving, compositional wrt input prefix, preserving infinite behaviour wrt ! encoding from Aπ into PIAπ, POAπ and PAπ.

Similar to the previous impossibility results in this section. We first prove that any must-preserving, compositional wrt input prefix, preserving infinite behaviour wrt ! encoding introduces divergent behaviours (Lemma 5.5.4) . However, unlike the previous section, we need more auxiliary results as the requirement of preservation of infinite behaviour is weaker. The notion of preservation of infinite behaviour does not consider internal infinite behaviour, as a consequence we can not infer that if $[\![!P]\!] \xrightarrow{\tau}$ then $[\![!P]\!]\uparrow$ by using preservation of infinite behaviour.

From the above, we first introduce four auxiliary results before showing any must-preserving, compositional wrt input prefix, preserving infinite behaviour wrt ! encoding encoding introduces divergent behaviours.

The following three auxiliary results, presented in Propositions 5.5.2 and 5.5.3 and Lemma 5.5.2, are used in the proof of the fourth auxiliary result: Lemma 5.5.3, that in turn will be used to prove that any encoding introduces divergent behaviours (Lemma 5.5.4). Finally with the help of Lemma 5.5.4 we obtain the separation result Theorem 5.5.2.

Proposition 5.5.2 *Let $[\![\cdot]\!] : A\pi \to \mathcal{P}$ with $\mathcal{P} \in \{PIA\pi, POA\pi, PA\pi\}$, satisfying must-preserving. Then $[\![0]\!] \not\downarrow$.*

Proof.

As a means of contradiction, let us suppose that $[\![0]\!]\uparrow$. By must-preservation $[\![\bar{a}z \mid a(x).\omega]\!] \not\xrightarrow{\omega}$ as otherwise $[\![a(x)]\!]$ *must* $[\![\bar{a}z \mid a(x).\omega]\!]$, a contradiction. Then $[\![0]\!] \mid [\![\bar{a}z \mid a(x).\omega]\!]$ can diverge following an infinite computation of τ-actions from $[\![0]\!]$ without intervention of $[\![\bar{a}z \mid a(x).\omega]\!]$. Therefore $[\![0]\!]$ *must* $[\![\bar{a}z \mid a(x).\omega]\!]$ but 0 *must* $\bar{a}z \mid a(x).\omega$. As the encoding is must-preserving this is a contradiction. □

Proposition 5.5.3 *Let $[\![\cdot]\!] : A\pi \to \mathcal{P}$ with $\mathcal{P} \in \{PIA\pi, POA\pi, PA\pi\}$, satisfying must-preserving. Then $[\![\bar{a}z \mid \bar{a}'z']\!] \not\downarrow$.*

Proof.

As a means of contradiction, let us suppose that $[\![\bar{a}z \mid \bar{a}'z']\!] \uparrow$. By must-preservation $[\![\bar{b}y \mid b(x).\omega]\!] \not\xrightarrow{\omega}$ as otherwise $[\![b(x)]\!]\ must\ [\![\bar{b}y \mid b(x).\omega]\!]$ a contradiction, then $[\![\bar{a}z \mid \bar{a}'z']\!] \mid [\![\bar{b}y \mid b(x).\omega]\!]$ can diverge following an infinite computation of τ-actions from $[\![\bar{a}z \mid \bar{a}'z']\!]$ without intervention of $[\![\bar{b}y \mid b(x).\omega]\!]$. Therefore $[\![\bar{a}z \mid \bar{a}'z']\!]\ m\not{u}st\ [\![\bar{b}y \mid b(x).\omega]\!]$ but $\bar{a}z \mid \bar{a}'z'\ must\ \bar{b}y \mid b(x).\omega$. As the encoding is must-preserving this is a contradiction.

□

For the remainder of this section we use the notion of derivative process:

Definition 5.5.2 (Derivative processes) For any process P the set of its (τ) derivative processes is defined as $Der(P) := \{Q | P(\xrightarrow{\tau})^* Q\}$.

Lemma 5.5.2 *Let $[\![\cdot]\!] : A\pi \to \mathcal{P}$ with $\mathcal{P} \in \{PIA\pi, POA\pi, PA\pi\}$, satisfying must-preserving and fn($[\![0]\!]$) $= \emptyset$. There is at least one maximal computation from $[\![x(y).\omega]\!]$ such that $[\![x(y).\omega]\!] \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} P_3 \xrightarrow{\tau}$ [. . .], where there is no $P_i \xrightarrow{\omega}$ with $i \geq 0$.*

Proof.

Let us consider $[\![x(y).\omega]\!] \mid [\![0]\!]$. Since $fn([\![0]\!]) = \emptyset$ we know that there are no possible synchronisations between $[\![0]\!]$ and its derivatives with $[\![x(y).\omega]\!]$ and its derivatives. As $[\![0]\!] \not\downarrow$ (Proposition 5.5.2), then the maximal computations of τ-actions from $[\![0]\!]$ are finite. Therefore in any maximal computation of τ-actions from $[\![0]\!] \mid [\![x(y).\omega]\!]$ there is a finite number of τ-actions corresponding to those ones of a maximal (finite) computation of τ-actions from $[\![0]\!]$ and the rest of τ-actions corresponds to those ones of a maximal computation of τ-actions from $[\![x(y).\omega]\!]$. By must-preservation $[\![0]\!]\ m\not{u}st\ [\![x(y).\omega]\!]$, it means there is a maximal computation from $[\![0]\!] \mid [\![x(y).\omega]\!]$ where there is no process satisfying the predicate $\xrightarrow{\omega}$. Therefore there must be a least one maximal computation of τ-actions from $[\![x(y).\omega]\!]$ such that $[\![x(y).\omega]\!] \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} P_3 \xrightarrow{\tau}$ [. . .], where there is no $P_i \xrightarrow{\omega}$ with $i \geq 0$. □

The following result is needed in the proof of the main lemma of this section.

Lemma 5.5.3 *Let $[\![\cdot]\!] : A\pi \to \mathcal{P}$ with $\mathcal{P} \in \{PIA\pi, POA\pi, PA\pi\}$, satisfying:*

- *compositionality w.r.t input prefix.*
- *must-preservation.*
- $fn(\llbracket 0 \rrbracket) = 0$.

then $\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket$ or some of its derivatives can synchronise with $C_{x(y)}[0]$ or with some of $C_{x(y)}[0]$'s derivatives.

Proof.

As a means of contradiction, let us suppose that neither $\llbracket \bar{x}z \mid x'z' \rrbracket$ nor its derivatives can synchronise with $C_{x(y)}[0]$ or its derivatives. Considering this assumption we analyse maximal computations of τ-actions from $C_{x(y)}[\llbracket \omega \rrbracket]$ and $\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \mid C_{x(y)}[\llbracket \omega \rrbracket]$ we shall conclude that $\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket$ *must* $\llbracket x(y).\omega \rrbracket$ for any names x, y, z, x', z'. This is a contradiction since the encoding is must-preserving.

From Lemma 5.5.2 and compositionality wrt input prefix there is a maximal computation s of τ-actions from $C_{x(y)}[\llbracket \omega \rrbracket]$ does not satisfy the predicate $\xrightarrow{\omega}$. It means $\llbracket \omega \rrbracket$ is guarded in the whole computation. Let us consider two possible cases for s

- The computation s is finite: $C_{x(y)}[\llbracket \omega \rrbracket] \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots P_n \not\xrightarrow{\tau}$. Where $\llbracket \omega \rrbracket$ is guarded in C_i, for all $i \in \{1, \ldots, n\}$. Therefore it is possible to construct a maximal computation of τ-actions for $\llbracket \bar{x}z \mid x'z' \rrbracket \mid C_{x(y)}[\llbracket \omega \rrbracket]$ of the following form:

$$\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \mid C_{x(y)}[\llbracket \omega \rrbracket] \xrightarrow{\tau} Q_1 \mid C_{x(y)}[\llbracket \omega \rrbracket] \xrightarrow{\tau} \ldots$$
$$\ldots Q_m \mid C_{x(y)}[\llbracket \omega \rrbracket] \xrightarrow{\tau} Q_m \mid P_1 \xrightarrow{\tau} \ldots Q_m \mid P_n \not\xrightarrow{\tau}$$

  Where $\llbracket \bar{x}z \mid x'z' \rrbracket \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \ldots Q_m \not\xrightarrow{\tau}$ is a maximal computation of τ-actions of $\llbracket \bar{x}z \mid x'z' \rrbracket$ [1]. Notice that $Q_m \mid P_n \not\xrightarrow{\tau}$ as $Q_m \not\xrightarrow{\tau}$, $P_n \not\xrightarrow{\tau}$ and as $\llbracket \omega \rrbracket$ is guarded in P_n, P_n can not synchronise with Q_m (it is because of the assumption that $\llbracket \bar{x}z \mid x'z' \rrbracket$ and its derivatives cannot synchronise $\llbracket a(x).0 \rrbracket$ and its derivatives).

[1] Notice that every maximal computation of τ-actions of $\llbracket \bar{x}z \mid x'z' \rrbracket$ is finite as $\llbracket \bar{x}z \mid x'z' \rrbracket \not\downarrow$ from Theorem 5.5.3.

Therefore there is a maximal computation of τ-actions of $[\![\bar{x}z \mid \bar{x}'z']\!] \mid C_{x(y)}[[\![\omega]\!]]$ not satisfying the predicate $\xrightarrow{\omega}$ i.e. $[\![\bar{x}z \mid \bar{x}'z']\!]$ *m⁄ust* $[\![x(y).\omega]\!]$. By must-preservation this is a contradiction.

- The computation s is infinite: $[\![x(y).\omega]\!] \xrightarrow{\tau} R_1 \xrightarrow{\tau} R_2 \xrightarrow{\tau} \ldots$. Where $[\![\omega]\!]$ is guarded in R_i, for all $i \geq 1$. It is possible to construct a maximal infinite computation of τ-actions for $[\![\bar{x}z \mid [\![x'z']\!] \mid x(y).\omega]\!]$ of the following form:

$$[\![\bar{x}z \mid \bar{x}'z']\!] \mid C_{x(y)}[[\![\omega]\!]] \xrightarrow{\tau} [\![\bar{x}z \mid \bar{x}'z']\!] \mid R_1 \xrightarrow{\tau} [\![\bar{x}z \mid \bar{x}'z']\!] \mid R_2 \xrightarrow{\tau} \ldots$$

Where $[\![\bar{x}z \mid \bar{x}'z']\!] \mid R_i \xrightarrow{\tau} [\![\bar{x}z \mid \bar{x}'z']\!] \mid R_{i+1}$ for all $i \geq 1$. It means that there is a maximal infinite computation of τ-actions of $[\![\bar{x}z \mid \bar{x}'z']\!] \mid [\![x(y).\omega]\!]$ with $[\![\omega]\!]$ unguarded and therefore this does not satisfy the predicate $\xrightarrow{\omega}$. Then $[\![\bar{x}z \mid \bar{x}'z']\!]$ *m⁄ust* $C_{x(y)}[[\![\omega]\!]]$. By must-preservation it is a contradiction.

□

Now, we can prove that a must-preserving, compositional wrt input prefix, preserving infinite behaviour wrt ! encoding introduces divergent behaviours.

Lemma 5.5.4 Let $[\![\cdot]\!] : \mathrm{A}\pi \to \mathcal{P} \in \{\mathrm{PIA}\pi, \mathrm{POA}\pi, \mathrm{PA}\pi\}$ be an encoding satisfying:

- compositionality w.r.t. input prefix and replication,
- must-preservation,
- preservation of infinite behaviour wrt !,
- $[\![\omega]\!] \xrightarrow{\omega}$,
- $fn([\![0]\!]) = \emptyset$,

Then $\forall x, x', y, z, z' \in \mathcal{N}$, $C_!\{C_{x(y)}[0]\} \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$ or $C_{x(y)}[0] \mid C_!\{[\![\bar{x}z \mid \bar{x}'z']\!]\} \uparrow$.

Proof.

From Lemma 5.5.3 a synchronisation can happen between a derivative of $[\![\bar{x}z \mid \bar{x}'z']\!]$ and a derivative of $C_{x(y)}[0]$. There are four cases:

- Case 1 : $C_{x(y)}[0] \stackrel{\bar{x}''z''}{\Longrightarrow}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{x''z''}{\Longrightarrow}$: From compositionality wrt replication $C_!'[C_{x(y)}[0]] = [\![!C_{x(y)}[0]]\!]$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] = [\![!\bar{x}z \mid \bar{x}'z']\!]$. From Item 1 of preservation of infinite behaviour $C_!'[C_{x(y)}[0]] \stackrel{\bar{x}''z''}{\Longrightarrow}$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] \stackrel{x''z''}{\Longrightarrow}$.

- Case 2 : $C_{x(y)}[0] \stackrel{x''z''}{\Longrightarrow}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{\bar{x}''z''}{\Longrightarrow}$: From compositionality wrt replication $C_!'[C_{x(y)}[0]] = [\![!C_{x(y)}[0]]\!]$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] = [\![!\bar{x}z \mid \bar{x}'z']\!]$. From Item 1 of preservation of infinite behaviour $C_!'[C_{x(y)}[0]] \stackrel{x''z''}{\Longrightarrow}$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] \stackrel{\bar{x}''z''}{\Longrightarrow}$.

- Case 3 : $C_{x(y)}[0] \stackrel{\bar{x}''(z'')}{\Longrightarrow}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{x''z''}{\Longrightarrow}$: From compositionality wrt replication $C_!'[C_{x(y)}[0]] = [\![!C_{x(y)}[0]]\!]$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] = [\![!\bar{x}z \mid \bar{x}'z']\!]$. From Item 1 of preservation of infinite behaviour $C_!'[C_{x(y)}[0]] \stackrel{\bar{x}''(z'')}{\Longrightarrow}$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] \stackrel{x''z''}{\Longrightarrow}$.

- Case 4 : $C_{x(y)}[0] \stackrel{x''z''}{\Longrightarrow}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{\bar{x}''(z'')}{\Longrightarrow}$: From compositionality wrt replication $C_!'[C_{x(y)}[0]] = [\![!C_{x(y)}[0]]\!]$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] = [\![!\bar{x}z \mid \bar{x}'z']\!]$. From Item 1 of preservation of infinite behaviour $C_!'[C_{x(y)}[0]] \stackrel{x''z''}{\Longrightarrow}$ and $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] \stackrel{\bar{x}''(z'')}{\Longrightarrow}$.

Now we prove that in each of the cases above $C_!'[C_{x(y)}[0]] \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$ or $C_{x(y)}[0] \mid C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] \uparrow$.

- From Case 1 we know that $C_{x(y)}[0] \stackrel{\bar{x}''z''}{\Longrightarrow}$, $C_!'[[\![\bar{x}z \mid \bar{x}'z']\!]] \stackrel{x''z''}{\Longrightarrow}$, $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{x''z''}{\Longrightarrow}$, and $C_!'[C_{x(y)}[0]] \stackrel{\bar{x}''z''}{\Longrightarrow}$. Now we can consider each of the target calculus:

  - If $\mathcal{P} =$ PIAπ or $\mathcal{P} =$ PAπ: From $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{x''z''}{\Longrightarrow}$ we know that $[\![\bar{x}z \mid \bar{x}'z']\!] \Longrightarrow \equiv (\nu z_1)..(\nu z_m)(!x''z''.P' | Q')$ where $\forall i \in [1..m], x'' \neq z_i$. From $C_!'[C_{x(y)}[0]] \stackrel{\bar{x}''z''}{\Longrightarrow}$ and Item 2 of preservation of infinite behaviour wrt ! we have $C_!'[C_{x(y)}[0]] \stackrel{\bar{x}''z''}{\Longrightarrow} \stackrel{\bar{x}''z''}{\Longrightarrow} \ldots$. Therefore $C_!'[C_{x(y)}[0]] \mid [\![\bar{x}z \mid \bar{x}'z']\!] \uparrow$
  - If $\mathcal{P} =$ POAπ: From $C_{x(y)}[0] \stackrel{\bar{x}''z''}{\Longrightarrow}$ we know that $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_m)(!\bar{x}''z''|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. From

$C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{x''z''}{\Longrightarrow}$ and Item 2 of preservation of infinite behaviour wrt ! we have $C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{x''z''}{\Longrightarrow} \stackrel{x''z''}{\Longrightarrow} \stackrel{x''z''}{\Longrightarrow} \ldots$. Therefore $C_{x(y)}[0] \mid C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \uparrow$.

- From Case 2 we know that $C_{x(y)}[0] \stackrel{x''z''}{\Longrightarrow}$, $C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{\bar{x}''z''}{\Longrightarrow}$, $\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{\bar{x}''z''}{\Longrightarrow}$, and $C_![C_{x(y)}[0]] \stackrel{x''z''}{\Longrightarrow}$. Now we can consider each of the target calculus:

  - If $\mathcal{P} = \text{PIA}\pi$ or $\mathcal{P} = \text{PA}\pi$: From $C_{x(y)}[0] \stackrel{x''z''}{\Longrightarrow}$ we know that $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_m)(!x''z''.P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. From $C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{\bar{x}''z''}{\Longrightarrow}$ and Item 2 of preservation of infinite behaviour wrt ! we have $C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{\bar{x}''z''}{\Longrightarrow} \stackrel{\bar{x}''z''}{\Longrightarrow} \stackrel{\bar{x}''z''}{\Longrightarrow} \ldots$. Therefore $C_{x(y)}[0] \mid C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \uparrow$.
  - If $\mathcal{P} = \text{POA}\pi$: From $\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \stackrel{\bar{x}''z''}{\Longrightarrow}$ we know that $\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \Longrightarrow \equiv (\nu z_1)..(\nu z_m)(!\bar{x}''z''.P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. From $C_![C_{x(y)}[0]] \stackrel{x''z''}{\Longrightarrow}$ and Item 2 of preservation of infinite behaviour wrt ! we have $C_![C_{x(y)}[0]] \stackrel{x''z''}{\Longrightarrow} \stackrel{x''z''}{\Longrightarrow} \ldots$. Therefore $C_![C_{x(y)}[0]] | \llbracket \bar{x}z | \bar{x}'z' \rrbracket \uparrow$

- The treatment of Case 3 Case 4 are similar to that of Case 1 and Case 2 respectively.

Therefore we can conclude that $C_![C_{x(y)}[0]] \mid \llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \uparrow$ or $C_{x(y)}[0] \mid C_!\llbracket \bar{x}z \mid \bar{x}'z' \rrbracket \uparrow$.

□

We conclude this section with our second impossibility result:

Theorem 5.5.2 Let $\llbracket \cdot \rrbracket : \text{A}\pi \to \mathcal{P} \in \{\text{PIA}\pi, \text{POA}\pi, \text{PA}\pi\}$ be an encoding satisfying:

- compositionality w.r.t. input prefix and replication,
- preservation of infinite behaviour wrt !,
- $\llbracket \omega \rrbracket \stackrel{\omega}{\longrightarrow}$,

- $fn([\![0]\!]) = \emptyset$,

Then the encoding is not must-preserving

Proof.

Similar to Proof of Theorem 5.5.1 but using Lemma 5.5.4 instead of Lemma 5.5.1.

□

5.6 Specialized impossibility result for PAπ

In the previous section we gave a uniform impossibility result for the existence of encodings of Aπ into the (semi-)persistent calculi. In this section, we give a further impossibility result, under different hypotheses, for encodings from Aπ into PAπ.

Theorem 5.6.1 Let $[\![\cdot]\!]$ be an encoding from Aπ into PAπ that satisfies:

1. compositionality w.r.t. input prefix,
2. $[\![\omega]\!] \xrightarrow{\omega}$.

Then $[\![\cdot]\!]$ is not must-preserving.

Proof. As a means of contradiction, let us suppose that $[\![\cdot]\!]$ is must-preserving. Therefore, $[\![x(y).\omega]\!]$ *must* $[\![\bar{x}z]\!]$ for any $x, y, z \in \mathcal{N}$, by compositionality wrt input prefix $C_{x(y)}[[\![\omega]\!]]$ *must* $[\![\bar{x}z]\!]$. Let us consider two cases according to the behaviour of $C_{x(y)}[[\![\omega]\!]]$, as a result we shall conclude that $C_{x(y)}[0] \mid [\![\bar{x}z]\!] \uparrow$:

- $C_{x(y)}[[\![\omega]\!]] \xrightarrow{\tau}$: In this case, $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!x'(y').P|!\bar{x'}z'|Q)$. As the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) we infer that $C_{x(y)}[0]$ $\equiv (\nu z_1) \ldots (\nu z_n)(!x'(y').P'|!\bar{x'}z'|Q')$, as a result $C_{x(y)}[0] \mid [\![\bar{x}z]\!] \uparrow$.
- $C_{x(y)}[[\![\omega]\!]] \not\xrightarrow{\tau}$: As the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) and $\omega \notin C_{x(y)}[\cdot]$ (This is a consequence of $\omega \notin \mathcal{N}$) we conclude that ω is prefixed in $C_{x(y)}[[\![\omega]\!]]$. As for $[\![\bar{x}z]\!]$, if $[\![\bar{x}z]\!] \xrightarrow{\tau}$ then $[\![\bar{x}z]\!] \equiv$

$(\nu z_1)\ldots(\nu z_n)(!x''(y'').P''|!\bar{x''}z''|Q'')$. Therefore $[\![\bar{x}z]\!]\uparrow$ and thus $C_{x(y)}[0] \mid [\![\bar{x}z]\!]\uparrow$. Hence, we consider the case when $[\![\bar{x}z]\!] \stackrel{\tau}{\not\longrightarrow}$ in the remainder of this proof. Since $C_{x(y)}[[\![\omega]\!]] \stackrel{\tau}{\not\longrightarrow}$, ω is prefixed in $C_{x(y)}[[\![\omega]\!]]$ and $[\![\bar{x}z]\!] \stackrel{\tau}{\not\longrightarrow}$, there must be at least one interaction between $C_{x(y)}[[\![\omega]\!]]$ and $[\![\bar{x}z]\!]$ so that ω becomes observable. i.e., $C_{x(y)}[[\![\omega]\!]] \mid [\![\bar{x}z]\!] \stackrel{\tau}{\longrightarrow}$ where either $C_{x(y)}[[\![\omega]\!]] \stackrel{\bar{x}''z''}{\longrightarrow}$ and $[\![\bar{x}z]\!] \stackrel{x''z''}{\longrightarrow}$ or $C_{x(y)}[[\![\omega]\!]] \stackrel{x''z''}{\longrightarrow}$ and $[\![\bar{x}z]\!] \stackrel{\bar{x}''z''}{\longrightarrow}$, let us consider two cases, the other are analogous:

- $C_{x(y)}[[\![\omega]\!]] \stackrel{\bar{x}''z''}{\longrightarrow}$ and $[\![\bar{x}z]\!] \stackrel{x''z''}{\longrightarrow}$: then, $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!\bar{x}''z''.P|Q)$ where $\forall i \in [1..n], x'' \neq z_i$ and $[\![\bar{x}z]\!] \equiv (\nu z_1)..(\nu z_m)(!x''(y'').P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$. As the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) $C_{x(y)}[0] \equiv (\nu z_1)..(\nu z_n)(!\bar{x}''z''.P''|Q'')$ where $\forall i \in [1..n], x'' \neq z_i$. As a result, $C_{x(y)}[0] \mid [\![\bar{x}z]\!]\uparrow$.
- $C_{x(y)}[[\![\omega]\!]] \stackrel{x''z''}{\longrightarrow}$ and $[\![\bar{x}z \mid \bar{x}'z']\!] \stackrel{\bar{x}''z''}{\longrightarrow}$: then, $[\![\bar{x}z]\!] \equiv (\nu z_1)..(\nu z_m)(!\bar{x}''z''.P'|Q')$ where $\forall i \in [1..m], x'' \neq z_i$ and $C_{x(y)}[[\![\omega]\!]] \equiv (\nu z_1)..(\nu z_n)(!x''(y'').P|Q)$ where $\forall i \in [1..n], x'' \neq z_i$. As the hole in $C_{x(y)}[\cdot]$ is prefixed (Proposition 5.5.1) $C_{x(y)}[0] \equiv (\nu z_1)\ldots(\nu z_n)(!x''(y'').P''|Q'')$ where $\forall i \in [1..n], x'' \neq z_i$. Therefore, $C_{x(y)}[0] \mid [\![\bar{x}z]\!]\uparrow$.

As $C_{x(y)}[0] \mid [\![\bar{x}z]\!]\uparrow$ and by following the reasoning used in the proof of Theorem 5.5.1 we can conclude that $C_{x(y)}[[\![\omega]\!]] \mid [\![\bar{x}z]\!]$ has at least one infinite computation where ω is not observable. Hence, $C_{x(y)}[[\![\omega]\!]]$ *must* $[\![xz]\!]$, i.e., $[\![x(y).\omega]\!]$ *must* $[\![\bar{x}z]\!]$. $[\![\cdot]\!]$ is not must-preserving, this is a contradiction.

□

The above theorem resembles the impossibility result in [70] about the existence of an encoding from Aπ into PAπ wrt weak bisimulation (and output equivalence). However, the hypothesis of the result in [70] is different. Namely, it is restricted to encodings homomorphic wrt parallelism.

5.7 Decidability results for POAπ

We shall prove that there is no computable encoding preserving divergence or convergence from Aπ into POAπ. We do this by proving that unlike for Aπ, divergence and convergence are decidable for POAπ processes.

We need to prove that the set of reachable processes through a τ-action can be computed; $Succ(P) = \{P' \,\big|\, P \xrightarrow{\tau} P'\}$ is computable.

W.l.o.g we assume that all the bound and free names are distinct in every process we consider in this section. Notice that every process can be transformed into an equivalent process with distinct names by using α-conversion.

It is well-known that the relation $\xrightarrow{\alpha}$ is image-finite [83]. Therefore the set of successors of a process P, $Succ(P)$, is finite. Here we describe how to build this set.

5.7.1 Computing Successors

Lemma 5.7.1 *For any P, $Deriv_{\bar{x}z}(P) = \{P' \,\big|\, P \xrightarrow{\bar{x}z} P'\}$ is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0$: $Der(P) := \emptyset$.
- $P = \bar{x}y.Q$: $Der(P) := \{Q\}$ if $y = z$, otherwise $Der(P) = \emptyset$.
- $P = x(y).Q$: $Der(P) := \emptyset$.
- $P = Q \mid R$: $Der(P) := \{(Q' \mid R) \,\big|\, Q' \in Der(Q)\} \cup \{Q \mid R' \,\big|\, R' \in Der(R)\}$.
- $P = (\nu y)Q$: $Der(P) := \{(\nu y)Q' \,\big|\, Q' \in Der(Q)\}$ if $y \notin \{x, z\}$.
- $P =!Q$: $Der(P) := \{(Q' \mid !Q) \,\big|\, Q' \in Der(Q)\}$.

It can be proved that $Der(P) = Deriv_{\bar{x}z}(P)$ by induction on P.

□

Lemma 5.7.2 *For any P, $Deriv_{bound-output}(P) = \{(\alpha, P') \,\big|\, P \xrightarrow{\alpha} P' \; where \; \alpha \; is \; a \; bound-output\}$ is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0$: $Der(P) := \emptyset$.
- $P = \bar{x}y.Q$: $Der(P) := \emptyset$.
- $P = x(y).Q$: $Der(P) := \emptyset$.
- $P = Q \mid R$: $Der(P) := \{(\alpha, Q' \mid R) \,\big|\, (\alpha, Q') \in Der(Q)\} \cup \{(\alpha, Q \mid R') \,\big|\, (\alpha, R') \in Der(R)\}$.
- $P = (\nu y)Q$: $Der(P) := \{(\bar{x}(y), (\nu y)Q') \,\big|\, Q' \in Deriv_{\bar{x}y}(P)) \; where \; x \notin y\}$
- $P = !Q$: $Der(P) := \{(\alpha, Q' \mid !Q) \,\big|\, (\alpha, Q') \in Der(Q)\}$.

It can be proved that $Der(P) = Deriv_{bound-output}(P)$ by induction on P.

□

Lemma 5.7.3 *For any P, $Deriv_{xz}(P) = \{P' \,\big|\, P \xrightarrow{xz} P'\}$ is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0$: $Der(P) := \emptyset$.
- $P = \bar{x}y.Q$: $Der(P) := \emptyset$.
- $P = x(y).Q$: $Der(P) := \{Q\{z/y\}\}$.
- $P = Q \mid R$: $Der(P) := \{(Q' \mid R) \,\big|\, Q' \in Der(Q)\} \cup \{(Q \mid R') \,\big|\, R' \in Der(R)\}$.
- $P = (\nu y)Q$: $Der(P) := \{((\nu y)Q') \,\big|\, Q' \in Der(Q)\}$ if $y \notin \{x, z\}$.

- $P = !Q : Der(P) := \{(Q' \mid !Q) \Big| Q' \in Der(Q)\}$.

It can be proved that $Der(P) = Deriv_{xz}(P)$ by induction on P.

□

Now we show how to calculate $Succ(P) = \{P' \Big| P \xrightarrow{\tau} P'\}$.

Lemma 5.7.4 *For any P, $Succ(P) = \{P' \Big| P \xrightarrow{\tau} P'\}$ is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0$: $Der(P) := \emptyset$.
- $P = \bar{x}y.Q$: $Der(P) := \emptyset$.
- $P = x(y).Q : Der(P) := \emptyset$.
- $P = Q \mid R : Der(P) := \{(Q' \mid R) \Big| Q' \in Der(Q)\} \cup \{(Q \mid R') \Big| R' \in Der(R)\} \cup Der_n(P)$ where $Der_n(P)$ represents the set of the derivatives processes from P through τ-action resulting from synchronization between Q and R, $Der_n(P)$ is defined as follows: $Der_n(P) := \{(Q' \mid R') \Big| Q' \in Deriv_{\bar{x}z}(Q),\ R' \in Deriv_{xz}(R)\ for\ some\ x, z \in fn(Q)\} \cup \{(Q' \mid R') \Big| Q' \in Deriv_{xz}(Q),\ R' \in Deriv_{\bar{x}z}(R)\ for\ some\ x, z \in fn(R)\} \cup \{(\nu z)(Q' \mid R') \Big| (\bar{x}(z), Q') \in Deriv_{bound-output}(Q), R' \in Deriv_{xz}(R)\ for\ some\ x \in fn(Q), z \notin fn(R)\} \cup \{(\nu z)(Q' \mid R') \Big| (\bar{x}(z), R') \in Deriv_{bound-output}(R), Q' \in Deriv_{xz}(Q)\ for\ some\ x \in fn(R), z \notin fn(Q)\}$.
- $P = (\nu y)Q : Der(P) := \{(\nu y)Q' \Big| Q' \in Der(Q)\}$
- $P = !Q : Der(P) := \{(Q' \mid !Q) \Big| Q' \in Der(Q)\} \cup \{((Q' \mid Q'') \mid !Q) \Big| Q' \in Deriv_{\bar{x}z}(Q),\ Q'' \in Deriv_{xz}(Q)\ for\ some\ x, z \in fn(Q)\} \cup \{((\nu z)(Q' \mid Q'')|!Q) \Big| (\bar{x}(z), Q') \in Deriv_{bound-output}(Q), Q'' \in Deriv_{xz}(Q)\ for\ some\ x \in fn(Q), z \notin fn(Q)\}$.

The calculability of $Der(P)$ when $P = Q \mid R$ or $P = !Q$ relies on the calculability of $Deriv_{xz}(\_)$, $Deriv_{\bar{x}z}(\_)$ and $Deriv_{bound-output}(\_)$ which is shown in Lemmata 5.7.3,5.7.1,5.7.2 respectively.

It can be proved that $Der(P) = Succ(P)$ by induction on P.

□

By using the function $Succ$, we can now determine whether a process is convergent (divergent) or not.

5.7.2 Decidability of convergence and divergence

Nowe, we can show that convergence and divergence are decidable for POAπ. First we need to introduce the notion of *occurrence of linear input prefix*.

Definition 5.7.1 (Occurrences of linear inputs prefix) *Let $P \in$ POAπ. The maximal number of occurrences of linear inputs in P, $LinearInp(P)$, is given inductively as follows : $LinearInp(0) = 0$, $LinearInp(!\bar{x}) = 0$, $LinearInp(a(x).P) = 1 + LinearInp(P)$, $LinearInp(!P) = 0$, $LinearInp((\nu x)P) = LinearInp(P)$, $LinearInp(P \mid Q) = LinearInp(P) + LinearInp(Q)$.*

The following proposition says that only input actions that come from linear input prefixes can participate, by a synchronization, in a finite maximal sequence of τ-actions.

Proposition 5.7.1 *Let P, $P' \in$ POAπ such that $P \xrightarrow{\tau} P'$ and P' is convergent. Then P is convergent and each τ-move from P into P' is produced by a synchronisation between an output and an input action coming from a linear input prefix.*

Proof.

To prove the first part, as a means of contradiction let us suppose that P is non-convergent, i.e. there is no maximal finite computation from P. As any maximal computation from P' can be seen as the ending part of a maximal computation from P passing through P'. Each maximal computation from P' must be infinite. Therefore. P' is non-convergent, a contradiction.

To prove that each τ-move from P into (convergent) P' is produced by a synchronisation between an output and an input action coming from a linear input prefix. It is enough to see that P' is non-convergent in other case: the output actions are persistent at time in POAπ, i.e. once an output action can be performed, it can be executed at anytime later on, the input actions coming from a non-linear input prefix are persistent as well, it is due to the effect of the replication operator over the input prefix. Therefore once a synchronisation involving an input action from a non-linear input prefix can happen, it can be repeated at anytime later on. i.e. there would not be a maximal finite computation after performing this kind of synchronisation. i.e. If $P \xrightarrow{\tau} P'$ where the synchronisation (τ-move) results from the participation of an input action coming from a non-linear input prefix, then any maximal computation from P' would be infinite. P' would be non-convergent. □

As a corollary from Proposition 5.7.1, we have the following proposition:

Proposition 5.7.2 *Let P, $P' \in$ POAπ such that $P \xrightarrow{\tau} P'$ and P' is convergent. Then $LinearInp(P) \geq 1$.*

A crucial observation to prove the decidability of convergence and divergence in POAπ is that the number of occurrences of linear input prefix decreases as long as a finite computation is performed.

Proposition 5.7.3 *Let P and $P' \in$ POAπ such that $P \xrightarrow{\tau} P'$ and P' is convergent. Then $LinearInp(P') = LinearInp(P) - 1$.*

Proof. From Proposition 5.7.1, we know that any τ-move from P into P' corresponds to a synchronisation where the input action comes from a linear input prefix. The participation of this kind of input action implies that an occurrence of a linear input prefix is consumed from P. Notice that although the execution of this input action can substitute names in P, the linear or persistent nature of the rest of the process remains unchanged. As for the output action, the consumption of an output action does not alter the number of occurrences of linear input prefix, it is due to the asynchronous nature of the calculus. □

The following Lemma states an upper bound of the length of the maximal finite computations which depends on the number of occurrences of linear inputs prefix. Notice that the lower bound does not depend on this number, e.g. $a(x).0 \mid a(x).0 \mid \ldots \mid a(x).0 \stackrel{\tau}{\not\longrightarrow}$.

Lemma 5.7.5 *Let $P \in$ POAπ. For each maximal finite computation c from P, $length(c) \leq LinearInp(P)$.*

Proof.

Let us consider any maximal finite computation from P:

$$P \stackrel{\tau}{\longrightarrow} P_1 \stackrel{\tau}{\longrightarrow} P_2 \stackrel{\tau}{\longrightarrow} P_3 \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} P_n \stackrel{\tau}{\not\longrightarrow}$$

From Theorem 5.7.3, we know that $LinearInp(P_1) = LinearInp(P) - 1$, in general $LinearInp(P_i) = LinearInp(P) - i$. Consequently, $LinearInp(P_{LinearInp(P)}) = 0$. From Proposition 5.7.2 $LinearInp(P_{LinearInp(P)}) \stackrel{\tau}{\not\longrightarrow}$. Therefore $length(c) \leq LinearInp(P)$.

□

From the computable function $Succ$, we can define and calculate a function $Succ^i(P) = \{P'' \mid P' \stackrel{\tau}{\longrightarrow} P'' \ for \ some \ P' \in Succ^{i-1}(P)\}$ where $Succ^1(P) = Succ(P)$ and $Succ^0(P) = \{P\}$, in a similar way we can identify the stable processes derivable from P at i τ-actions by the function $SuccSt^i(P) = \{P'' \mid P' \stackrel{\tau}{\longrightarrow} P'' \ for \ some \ P' \in Succ^{i-1}(P) \ and \ Succ(P'') = \{\}\}$ where $SuccSt^0(P) = \{P \mid Succ(P) = \{\}\}$.

Now, it is easy to see from Lemma 5.7.5 and by using the function $Succ^i$ and $SuccSt^i$, which are computable from Lemma 5.7.4, that divergence and convergence are decidable.

Theorem 5.7.1 *Divergence is decidable in POAπ.*

Proof.

From Lemma 5.7.5, a POAπ process P is divergent if and only if there is at least one computation from P whose length is greater than

$LinearInp(P)$. It can be checked whether such a computation exists by verifying that $|Succ^{LinearInp(P)+1}(P)| \geq 0$. From Lemma 5.7.4 it is clear that $Succ^{LinearInp(P)+1}(P)$ can be straightforwardly calculated. □

Theorem 5.7.2 *Convergence is decidable in POAπ.*

Proof. From Theorem 5.7.5, a POAπ process P is convergent if and only if there is at least one maximal computation from P whose length is less or equal to $LinearInp(P)$, i.e. if there is at least one stable process derivable from P at most in $LinearInp(P)$ τ-moves. It can be checked whether such a stable process exists by verifying that $|SuccSt^0(P)| + |SuccSt^1(P)| + |SuccSt^2(P)| + \ldots + |SuccSt^{LinearInp(P)}(P)| \geq 1$. From Lemma 5.7.4 it is clear that $SuccSt^i(P)$ for any natural number i can be straightforwardly calculated.

□

As corollary from Theorem 5.7.1 and Theorem 5.7.2 and considering Remark 5.1.1 we obtain the following separation result:

Theorem 5.7.3 *There is no encoding preserving and reflecting divergence (convergence) from Aπ into POAπ.*

5.8 Summary and related work

In this chapter we studied the expressive power of several subcalculi of Aπ by considering Testing semantics. As main contribution we showed a general negative result for a large class of encodings from Aπ into any of its semi-persistent subcalculi w.r.t must semantics. This class of encodings have properties such as compositionality and preservation of infinite behaviour wrt !, $fn([\![0]\!]) = 0$ and $[\![\omega]\!] \xrightarrow{\omega}$, in our opinion all these properties are reasonable. In fact, the encodings proposed in [70] belong to this class. The expressiveness gap relies on the fact that the encoding necessarily introduces divergence: it arises when translating processes with linear components (e.g. $\bar{x}z$, xy) and persistent ones (e.g. $!a(x).w$)). The translation transforms some linear components into persistent ones, thus some

interactions will introduce divergence. A stronger separation result holds for encodings into POAπ and PAπ. We proved that there are no computable encodings from Aπ into POAπ or PAπ preserving and reflecting divergence or convergence. This was proved by showing that convergence and divergence are decidable for the target calculi.

Most of the related work was discussed in the introduction. In a different context, in [64] it is shown that the separate choice encoding of the π-calculus into the asynchronous π-calculus preserves weak bisimulation, while in [26] the authors prove that no must-preserving encoding of the (choiceless) synchronous π-calculus into the asynchronous one exists. Hence must semantics is a good candidate to study the expressiveness of persistence when divergence is taken into account. Nevertheless, differently from [26], where the separation result is mainly due to the non-atomicity of the sequences of steps which are necessary in the asynchronous π-calculus to mimic synchronous communication, our negative results rely on the fact that divergence is introduced when encoding Aπ-processes in a semi-persistent subcalculus. As previously mentioned the study of persistence in [70] is incomplete as ignores the crucial issue of divergence. In this chapter, we used the divergence-sensitive framework of testing semantics to give a more complete account of the expressiveness of persistence in asynchronous calculi. In particular, as discussed in the introduction of this chapter, this work supports informal expressiveness loss claims in persistent asynchronous languages [11, 34, 30].

The separation result between Aπ and its semi-persistent subcalculi showed in Section 5.5.1 and the separation result between Aπ and PAπ in Section 5.6 were published as [25].

The separation results between Aπ and its semi-persistent subcalculi showed in Section and 5.5.2 and the decidability of convergence and divergence for POAπ given in Section 5.7 have not been published.

Chapter 6

Conclusions

We shall conclude this dissertation with a discussion on general related work and an overall summary of its contents (more specialized related work and summaries can be found at the end of each previous chapter). We shall also describe possible directions for future research.

- *Chapter 3.* In Chapter 3 we have studied the expressive power of $\text{CCS}_!$ w.r.t faithful encodings. These encodings do not allow to move from a (weakly) terminating state into a (strongly) non-terminating state. It is well-known that non-faithful encodings are necessary to achieve Turing expressivenes in $\text{CCS}_!$ [22]. We have proved that it is not possible to provide faithful encodings for models of computability strictly less expressive than Turing Machines into $\text{CCS}_!$, e.g. Context-Free languages. A mechanism allowing to move from a (weakly) terminating state into a (strongly) non-terminating state is also used in encodings of other languages in order to show their expressive power [41, 16]. We could explore as a future work the expressive power of these languages w.r.t. faithful encodings. In the author's opinion, it would be interesting to see whether the line of research followed in Chapter 3 could be useful in the understanding of the expressive power of the formalisms in [41, 16].

- *Chapter 4.* In Chapter 4 we have studied the expressive power of the interplay between replication and restriction. As two of the main results we have

proved that Turing-expresiveness is lost in $CCS_!$ when the restricted declarations are not under the scope of replication, and this in turn is more expressive than restriction-free $CCS_!$. This study leaves open expressiveness questions about other forms of interplay between replication and restriction. For example, the behaviour that can be achieved in $CCS_!$ by imposing that the occurrences of restricted names declared previously under the scope of replication can not be under the scope of other replication. The author thinks that that fragment is not Turing-expressive in the sense of Busi, Gabbrielli and Zavattaro [21, 22]. It would also interesting to study the expressiveness w.r.t. the number of restricted names. e.g two restricted names provides more expressiveness than only one? As shown in Chapter 3, priorities provides a significant expressive power. An interesting direction would be to find non-trivial fragments with a limited form of priorities in which relevant properties such as reachability, convergence and divergence among others are decidable. We claim that there is a relation between priorities and inhibitor arcs in Petri Nets as both are able to preclude other actions from being executed. This behaviour is fundamental in achieving Turing-expressiveness. These similarities could be explored, for example, by using decidability results of a proper subclass of Petri Nets systems with inhibitor arcs, called Primitive Systems [20].

- *Chapter 5.* In Chapter 5 we have studied the expressive power of linearity in Aπ by using Testing semantics. As main result we have proved that full linearity, i.e., linearity in both inputs and outputs, is necessary in Aπ in terms of expressiveness. Our result relies on the fact that the interplay between linear terms and persistent terms in Aπ can not be simulated, under some reasonable conditions, when linearity is restricted to either inputs or outputs. This expressivenes gap arises as divergence is introduced. It would be interesting to explore if full linearity without presence of any means of infinite behaviour is more expressive, at some extent, than a limited form of linearity (either in inputs or outputs) together with persistence. We think that it is possible to define a problem along the lines of [69, 27, 91] which can be solved by using full linearity but not with persistence and a limited

linearity.

We also think that it would be interesting to use Testing semantics to study other elements such as polyadicity in the π-calculus. In fact, we believe that it is possible to show that there exists an expressiveness gap between the polyadic and the monadic (a) synchronous π-calculus under some reasonable conditions including must-testing preservation. Intuitively the reason is that an encoding from polyadic into monadic would require to encode the polyadic synchronisation by following a sequence of two or more steps in the monadic one resembling a protocol. The completion of this "protocol", unlike the polyadic synchronisation, could be precluded because of a divergent behaviour. In fact the well-known encoding from the polyadic π-calculus into the monadic π-calculus is not must-preserving.

Although in Chapter 4 we focused our expressiveness study on fragments of $\text{CCS}_!$ (i.e. the π-calculus without mobility) we claim that all the results presented in that chapter are valid for the full π-calculus as mobility does not seem to add expressiveness to $\text{CCS}_!^{-!\nu}$ and $\text{CCS}_!^{-\nu}$ that would invalidate the results there presented. Likewise we claim that, apart from the specialized result for POAπ, the results presented in Chapter 5 are valid for the full synchronous π-calculus as the results do not rely on asynchrony.

Bibliography

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[3] L. Aceto and A. Ingólfsdóttir. On the expressibility of priority. *Inf. Process. Lett.*, 109(1):83–85, 2008.

[4] J. Alves-Foss. An efficient secure authenticated group key exchange algorithm for large and dynamic groups. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.

[5] R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.

[6] R. Amadio and C. Meyssonnier. On decidability of the control reachability problem in the asynchronous $\pi-$calculus. *Nordic Journal of Computing*, 9(2), 2002.

[7] J. Aranda, C. D. Giusto, M. Nielsen, and F. D. Valencia. Ccs with replication in the chomsky hierarchy: The expressive power of divergence. In Z. Shao, editor, *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 383–398. Springer, 2007.

[8] J. Aranda, F. D. Valencia, and C. Versari. On the expressive power of restriction and priorities in ccs with replication. In L. de Alfaro, editor, *FOS-*

SACS, volume 5504 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2009.

[9] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *J.ACM.*, 40(3):653–682, 1993.

[10] J. C. M. Baeten and F. Corradini. Regular expressions in process algebra. In *LICS '05*, pages 12–19, Washington, DC, USA, 2005. IEEE Computer Society.

[11] E. Best, F. de Boer, and C. Palamidessi. Partial order and sos semantics for linear constraint programs. In D. Garlan and D. L. Métayer, editors, *COORDINATION*, volume 1282 of *Lecture Notes in Computer Science*, pages 256–273. Springer, 1997.

[12] B. Blanchet. Security protocols: from linear to classical logic by abstract interpretation. *Inf. Process. Lett.*, 95(5):473–479, 2005.

[13] M. Boreale and M. Buscemi. A framework for the analysis of security protocols. In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 483–498. Springer, 2002.

[14] E. Borger, E. Gradel, and Y. Gurevich. *The Classical Decision Problem*. Springer Verlag, 1994.

[15] G. Boudol. Asynchrony and the pi-calculus. Technical Report RR-1702, INRIA Sophia Antipolis, 1992.

[16] M. Bravetti and G. Zavattaro. On the expressive power of process interruption and compensation. *Mathematical. Structures in Comp. Sci.*, 19(3):565–599, 2009.

[17] E. Brinksma, A. Rensink, and W. Vogler. Fair testing. In I. Lee and S. A. Smolka, editors, *CONCUR*, volume 962 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 1995.

[18] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, July 1984.

[19] O. Burkart, D. Caucal, F. Moller, and B. Steffen. *Verification on infinite structures*, chapter 9, pages 545–623. Elsevier, North-Holland, 2001.

[20] N. Busi. Analysis issues in petri nets with inhibitor arcs. *Theor. Comput. Sci.*, 275(1-2):127–177, 2002.

[21] N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. recursive definitions in channel based calculi. In *ICALP'03*, volume 2719 of *LNCS*, pages 133–144. Springer-Verlag, 2003.

[22] N. Busi, M. Gabbrielli, and G. Zavattaro. Comparing recursion, replication, and iteration in process calculi. In *ICALP'04*, volume 3142 of *LNCS*, pages 307–319. Springer-Verlag, 2004.

[23] D. Cacciagrano. *On Synchronous and Asynchronous Communication: Some Expressiveness Results*. PhD thesis, University of L'Aquila, 2004.

[24] D. Cacciagrano and F. Corradini. On synchronous and asynchronous communication paradigms. In A. Restivo, S. R. D. Rocca, and L. Roversi, editors, *ICTCS*, volume 2202 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 2001.

[25] D. Cacciagrano, F. Corradini, J. Aranda, and F. D. Valencia. Linearity, persistence and testing semantics in the asynchronous pi-calculus. *Electr. Notes Theor. Comput. Sci.*, 194(2):59–84, 2008.

[26] D. Cacciagrano, F. Corradini, and C. Palamidessi. Separation of synchronous and asynchronous communication via testing. *Electr. Notes Theor. Comput. Sci.*, 154(3):95–108, 2006.

[27] M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nord. J. Comput.*, 10(2):70–98, 2003.

[28] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.

[29] S. Christensen, Y. Hirshfeld, and F. Moller:. Decidable subsets of ccs. *Comput. J*, 37(4):233–242, 1994.

[30] F. Crazzolara. *Language, Semantics, and Methods for Security Protocols*. PhD thesis, University of Aarhus, 2003.

[31] F. Crazzolara and G. Winskel. Events in security protocols. In *ACM Conference on Computer and Communications Security*, pages 96–105, 2001.

[32] C. Eccher and C. Priami. Design and implementation of a tool for translating sbml into the biochemical stochastic pi-calculus. *Bioinformatics*, 22(24):3075–3081, 2006.

[33] J. Esparza and M. Nielsen. Decidability issues for petri nets. Technical report, BRICS RS-94-8, 1994.

[34] F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Inf. Comput.*, 165(1):14–41, 2001.

[35] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *CSFW*, pages 160–173. IEEE Computer Society, 2001.

[36] C. Fournet and M. Abadi. Hiding names: Private authentication in the applied pi calculus. In M. Okada, B. C. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, editors, *ISSS*, volume 2609 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2002.

[37] S. J. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.

[38] P. Giambiagi, G. Schneider, and F. D. Valencia. On the expressiveness of infinite behavior and name scoping in process calculi. In I. Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*. Springer, 2004.

[39] J. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[40] M. Giunti, K. Honda, V. T. Vasconcelos, and N. Yoshida. Session-based type discipline for pi calculus with matching. Presented at PLACES 2009 — 2nd International Workshop in Programming Language Approaches to Concurrency and Communication-cEntric Software, 2009.

[41] C. D. Giusto, J. A. Pérez, and G. Zavattaro. On the expressiveness of forwarding in higher-order communication. In M. Leucker and C. Morgan, editors, *ICTAC*, volume 5684 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2009.

[42] U. Goltz. Ccs and petri nets. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *LNCS*, pages 334–357. Springer, 1990.

[43] D. Gorla. On the relative expressive power of asynchronous communication primitives. In L. Aceto and A. Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2006.

[44] D. Gorla. Synchrony vs asynchrony in communication primitives. *Electr. Notes Theor. Comput. Sci.*, 175(3):87–108, 2007.

[45] D. Gorla. Towards a unified approach to encodability and separation results for process calculi. In van Breugel and Chechik [88], pages 492–507.

[46] Y. Hirshfeld. Petri nets and the equivalence problem. In *Proceedings of CSL'93*, volume 832 of *LNCS*, pages 165–174. Springer Verlag, 1993.

[47] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *ECOOP*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[48] H. Hüttel. Undecidable equivalences for basic parallel processes. In *TACS'94*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 1994.

[49] H. Huttel and J. Srba. Recursion vs. replication in simple cryptographic protocols. In *SOFSEM'05*, volume 3381 of *LNCS*, pages 175–184. Springer-Verlag, 2005.

[50] C. B. Jones. A pi-calculus semantics for an object-based design notation. In E. Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1993.

[51] P. C. Kanellakis and S. A. Smolka. CCS expressions finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[52] N. Kobayashi and T. Suto. Undecidability of bpp equivalences revisited. Technical report, Tohoku University, 2007.

[53] C. Laneve and A. Vitale. Expressivity in the kappa family. *Electr. Notes Theor. Comput. Sci.*, 218:97–109, 2008.

[54] R. Lucchi and M. Mazzara. A pi-calculus based semantics for ws-bpel. *J. Log. Algebr. Program.*, 70(1):96–118, 2007.

[55] S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. In *EXPRESS'03*, 2003.

[56] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer, 1991.

[57] R. Meersman, Z. Tari, M.-S. Hacid, J. Mylopoulos, B. Pernici, Ö. Babaoglu, H.-A. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, editors. *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*, volume 3760 of *Lecture Notes in Computer Science*. Springer, 2005.

[58] R. Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984.

[59] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.

[60] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[61] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[62] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation,*, 100, 1992.

[63] M. Minsky. *Computation: finite and infinite machines*. Prentice Hall, 1967.

[64] U. Nestmann. What is a 'good' encoding of guarded choice? *Electr. Notes Theor. Comput. Sci.*, 7, 1997.

[65] R. D. Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

[66] M. Nielsen, C. Palamidessi, and F. Valencia. On the expressive power of concurrent constraint programming languages. In *PPDP 2002*, pages 156–167. ACM Press, Oct. 2002.

[67] C. Olarte. *Universal Temporal Concurrent Constraint Programming*. PhD thesis, l'Ecole Polytechnique, 2009.

[68] C. Olarte and F. D. Valencia. The expressivity of universal timed CCP: Undecidability of monadic FLTL and closure operators for security. In *Proc. of PPDP 08*. ACM, 2008.

[69] C. Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[70] C. Palamidessi, V. A. Saraswat, F. D. Valencia, and B. Victor. On the expressiveness of linearity vs persistence in the asychronous pi-calculus. In *LICS*, pages 59–68. IEEE Computer Society, 2006.

[71] C. Palamidessi and F. D. Valencia. Recursion vs replication in process calculi: Expressiveness. *Bulletin of the EATCS*, 87:105–125, 2005.

[72] J. Parrow. Trios in concert. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 621–637. MIT Press, 2000.

[73] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *CSFW*, pages 84–95. IEEE Computer Society, 1997.

[74] A. Phillips and L. Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In M. Calder and S. Gilmore, editors, *CMSB*, volume 4695 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2007.

[75] A. Phillips, L. Cardelli, and G. Castagna. A graphical representation for biological processes in the stochastic pi-calculus. 4230:123–152, 2006.

[76] I. Phillips. Ccs with priority guards. *J. Log. Algebr. Program.*, 75(1):139–165, 2008.

[77] B. C. Pierce. Concurrent objects in a process calculus. In T. Ito and A. Yonezawa, editors, *Theory and Practice of Parallel Programming*, volume 907 of *Lecture Notes in Computer Science*, pages 187–215. Springer, 1994.

[78] B. C. Pierce and D. N. Turner. Pict: a programming language based on the pi-calculus. In G. D. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction*, pages 455–494. The MIT Press, 2000.

[79] K. V. S. Prasad. Broadcast calculus interpreted in ccs upto bisimulation. *Electr. Notes Theor. Comput. Sci.*, 52(1), 2001.

[80] F. Puhlmann. Soundness verification of business processes specified in the pi-calculus. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 6–23. Springer, 2007.

[81] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.

[82] D. Sangiorgi. The name discipline of uniform receptiveness. *Theor. Comput. Sci.*, 221(1-2):457–493, 1999.

[83] D. Sangiorgi and D. Walker. *The $\pi-$calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[84] V. Saraswat. *Concurrent Constraint Programming*. The MIT Press, Cambridge, MA, 1993.

[85] V. Saraswat and P. Lincoln. Higher-order linear concurrent constraint programming. Technical report, Xerox PARC, 1992.

[86] H. Smith and P. Fingar. *Business Process Management: The Third Wave*. Meghan-Kiffer Press, 2003.

[87] D. Taubner. Finite representation of CCS and TCSP programs by automata and Petri nets. In *LNCS 369*. Springer Verlag, 1989.

[88] F. van Breugel and M. Chechik, editors. *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*. Springer, 2008.

[89] W. M. P. van der Aalst. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype, 2004.

[90] V. T. Vasconcelos. Fundamentals of session types. In M. Bernardo, L. Padovani, and G. Zavattaro, editors, *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 158–186. Springer, 2009.

[91] C. Versari, N. Busi, and R. Gorrieri. On the expressive power of global and local priority in process calculi. In *CONCUR*, volume 4703 of *LNCS*, pages 241–255. Springer, 2007.

[92] M. Vigliotti, I. Phillips, and C. Palamidessi. Separation results via leader election problems. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 172–194. Springer, 2005.

[93] G. Zavattaro and L. Cardelli. Termination problems in chemical kinetics. In van Breugel and Chechik [88], pages 477–491.

Index