Expressiveness in Mobile Process Calculi

Ernesto Posse

School of Computer Science, McGill University

Montreal, Quebec, Canada

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfilment of the requirements of the degree of

Master of Science in Computer Science

March, 2001

Abstract

The development of formal methods and theories for the construction and analysis of concurrent systems has been a subject of increased interest, particularly in the last decade. Amongst the most recognized and studied theories is the π-calculus and its family of languages. The π-calculus is a language for describing and reasoning about *mobile* systems, that is, systems in which the topology of the communications network is dynamic and not fixed a priori. Examples of such systems include mobile phone networks and the TCP/IP protocol that underlies the Internet. Another paradigm for concurrency is known as *Concurrent Constraint Programming* or CCP for short. This is a particularly attractive model because of its close ties with Logic and its declarative style. In this thesis a survey of the π-calculus family and the CCP family is presented, emphasizing the relations both within each family and between the two paradigms. In particular we explore the question of whether CCP supports mobility.

Rèsume

Le développement de méthodes formels et de théories pour la construction et l'analyse des systèmes concurrents connait une forte croissance d'intérêt pendant la dernière décennie. Parmi les theories les plus reconnues se trouve le π-calculus et la famille de langages de sa descendance. Le π-calculus est un langage pour la description et raisonnement sur les systèmes *mobiles*, ceux où la topologie du réseau de communication change de façon dynamique. Les réseaux de téléphones mobiles et le protocole TCP/IP qui sert de base pour l'Internet figurent parmi les exemples de tels systèmes. Un autre paradigme de la concurrence est la *Programmatión Concurrente par Restrictions* ou CCP. Ses relations profondes relatives à la Logique et son style declaratif confèrent à la CCP un attrait particulier. Cette thèse présente une étude des familles du π-calculus et de CCP en mettant à point les relations entre les deux. Particulièrment nous examinons la question de l'admissión de la mobilité en CCP.

Acknowledgments

I would like to thank my advisor Prakash Panangaden for his help and guidance through the development of this thesis. I would also like to thank the School of Computer Science for providing an enriching environment in which to work. Catuscia Palamidessi provided invaluable help and comments that gave us insight into the relations between π-calculi, CCS and CCP. Björn Victor also helped by clarifying the relation between the Fusion calculus and π-calculus. My friends here at McGill also deserve thanks, both for support and for our debates on subjects sometimes marginal and sometimes directly related to this work. Particularly I would like to thank Will Renner, Jacob Eliosoff, and Alexander (Sasha) Wait. Will also patently proofread this thesis, and suggested many improvements. Special thanks to my family, in particular my mother Pilar Posada, for her constant encouragement and support.

Contents

Chapter 1

Introduction

In the world of computation today we find many kinds of complex software and hardware systems that include some sort of concurrent processing. Distributed and parallel computing, multi-threaded languages and operating systems, are just examples of the general concept of concurrency. A concurrent system is a system that has at least two different parts that, to some extent, perform their activities independently, and "at the same time." The idea of simultaneity need not be that of two activities performed at the same "physical" time. It can be abstracted to refer to "logical" time. This is the case of languages and operating systems with multiple threads of control executing in a single physical processor. Since there is only one processor, only one instruction is executed at a time (ignoring the internal pipeline of the processor), but conceptually, all the threads are active and running.

There are a number of reasons why concurrency is desirable. Firstly, in the presence of several physical processors we obtain a speed advantage if we can distribute the tasks necessary for solving the problem in an appropriate fashion amongst the processors. Secondly, it gives us an advantage from the design point of view, by serving as a tool to model realistically the problem domain, since most systems in the real world are naturally concurrent. Thirdly, the use of concurrency leads to more modular systems, since the developer decomposes the system in parts that work independently of non-essential constraints on the sequence of events or activities.

The advantages of concurrency do not come for free. The development and analysis of concurrent systems have been long known to be very difficult tasks. In order to perform a task, the multiple parts that form a concurrent system must interact and communicate with each other, to exchange, analyze or synthesize the information involved in solving the problem. The difficulty arises from this interaction.

There are different strategies for communication between processes. These include *shared variables* and *message passing*. In the former, messages can be posted and read by all processes that have access to a common memory (sometimes called a *blackboard*). In the latter, communication is performed through *channels* that carry messages so there is a precise control of which agents send or receive information. In this view, agents perform the basic operations of sending and receiving a message through a given channel.

Under the message-passing model, a communicating concurrent system can be seen as a network of processes connected by the channels. In recent years there has been an increasing interest in *mobile* systems, that is, systems in which the topology of the communication network evolves dynamically. The concept of mobility has found applications mainly in telecommunications and in the Internet.

It is desirable to be able to talk about properties of concurrent systems such as correctness with respect to a specification, liveness, fairness, etc. However, the change of network structure at runtime considerably complicates this analysis since the patterns of communication are not fixed. Intuition about how a system behaves is bound to lead to wrong conclusions (e.g. saying that the system is deadlock-free when in fact it can deadlock). It is therefore of vital importance to use reliable tools for defining and reasoning about this kind of system. We need mathematical models of computation for this.

In Theoretical Computer Science we find some foundational models such as Turing Machines, and the λ-calculus. Although these models have been successful in the study of concepts such as computational complexity and functional abstraction and application, they are inappropriate for dealing with interacting and concurrent systems. Quoting Milner

> "In looking for basic notions for a model of concurrency it is therefore probably wrong to extrapolate from λ-calculus, except to follow its example in seeking something small and powerful. (Here is an analogy: Music is an art form, but it would be wrong to look for an aesthetic theory to cover all art forms by extrapolation from musical theory.)" [26]

These models are not suitable to deal with concurrent communicating systems because they are based on the assumption that the system receives some input produces some output and ends processing. On the other hand, interactive systems intend to model *reactive* behaviour, in which input is continuously received and output produced so processing does not necessarily terminate.

Since Turing Machines and the λ-calculus do not appropriately represent the issues of concurrent computation, several frameworks have been developed to deal with these. These frameworks usually receive the name of *process calculi* or *process algebras*. One of the most influential developments in this field was Hoare's CSP, or *Communicating Sequential Processes* ([15]). Closely related was Milner's CCS, or *Calculus of Communicating Systems* ([25]). These languages provided a precise framework for describing and reasoning about concurrency, however, neither of them deal with mobile computation. Although mobility was first tackled in the Actor model of Hewitt and Agha ([14], [2]), not until Engberg and Nielsen ([10]) was an algebraic formulation provided. This work was latter simplified by Milner, Parrow and Walker [30] with the π-calculus.

The π-calculus serves us as the base point for the study of expressiveness in mobile process calculi, since it has become the de facto standard against which process calculi are compared. It can be seen both as a language for describing and for reasoning about systems; as a programming language as well as a specification language, due to its declarative nature. Several variants have been defined since its introduction. Amongst the most well-known are the *asynchronous π-calculus*, the *πI-calculus*, and more recently, the *fusion calculus*.

A different model of programming concurrent systems is found in the *Concurrent Constraint Programming* paradigm ([39], [35], [43]). This approach is based on the shared memory model of communication. Under this model variables need not have fixed values and agents provide partial information in the form of constraints over variables, all of which integrate a shared *constraint store*. The basic operations that processes perform are posting a constraint in the store (*tell*) or inquiring the store to see if a constraint is valid (*ask*). This model is very attractive, because it is closely tied to logic ([22]).

The diversity of languages and calculi for concurrency immediately suggests the question of expressiveness. How can we determine if a language is more powerful or expressive than another? Strictly speaking, no model of computation can have more expressive power than Turing Machines or the λ-calculus in the sense that these model *every* possible computable function. However, Turing machines are only a model of sequential computation. The issue of two separate machines interacting, is not even addressed by Turing-completeness. A major point about the difference between sequential and concurrent computation is the ability to handle infinite input/output. Quoting Panangaden

> "In the world of sequential computation if one subprogram produces infinite output this output is 'useless'. The subprogram will run forever and nothing will happen with the rest of the program. If, however, two subprograms run *concurrently* then one could be producing infinite output which the other examines *while it is being produced*. As simple example suppose that a Turing Machine produces an infinite sequence of primes in increasing order. Another Turing Machine can examine this list and check for the occurrences of a specific number. From the traditional viewpoint of sequential computation this is a divergent program! Clearly from a parallel viewpoint the program is entirely nontrivial and does not deserve to be lumped together with a program that just loops without ever producing output." [37]

We are interested in two different but related concepts of expressiveness:

- What does it mean to say that a programming language or calculus is more expressive or powerful than another?
- What does it mean to say that an agent, system or machine is more powerful that another?

Before discussing these, let us clarify the terminology. A *system*, is a collection of *agents*. The behaviour of an agent is a *process*. In the setting of concurrent computation, a system has at least two agents. A *programming language or calculus* is a collection of *terms* or *programs* with some semantics specifying what the terms mean. Systems and agents are represented by terms in a calculus. It is often the case that systems and agents are represented by the same syntactical category in the language, in order to allow for the definition of hierarchical systems, i.e. systems made out of subsystems. Hence we use the words "system", "agent", "process" and "term" interchangeably.

In the context of concurrent systems, the expressiveness of agents depends on how they communicate. The idea that interaction with external agents, or with the environment, should be taken into account to differentiate between the agents led Milner and Park ([25]) to the concept of "bisimulation" as a *behavioural equivalence* between agents. Roughly speaking, this concept is based on the following intuitions. We consider an agent A to be at least as powerful as an agent B if A can simulate B. We can say that A is equally powerful, (or behaviourally equivalent) to B if A and B cannot be distinguished by an external observer, and thus could be used interchangeably in any system.

What about the expressiveness of a programming language with respect to another? Determining which of two programming languages is more expressive, in the context of concurrency and interaction, depends on the relative power of the agents represented by terms in each language. To be able to compare the power of agents in different calculi, we need to understand the behaviour of such agents. Unless we can answer the question of whether two agents are equivalent, we cannot really say that we understand what is the behaviour of an agent, and what is its relative power. Thus the notion of behavioural equivalence of agents plays a very important role in comparing the expressiveness of calculi.

How do we relate the expressive power of languages with the power of their agents? Consider two languages $\mathcal{L}_1$ and $\mathcal{L}_2$. We expect $\mathcal{L}_1$ to be at least as expressive than $\mathcal{L}_2$ if each term in $\mathcal{L}_2$ can be simulated by a term in $\mathcal{L}_1$. In order to be able to make such comparison there needs to be a translation from $\mathcal{L}_2$ to $\mathcal{L}_1$. But it is not enough to say that there must be *a* translation. The establishment of the relative expressive power of the calculi depends on the (syntactic and semantic) properties that are preserved and/or reflected by the translation. One such properties is the relative power of agents within a language. We expect that the relative power between agents be respected by the translation. This is, when two agents are equally powerful in a language, their corresponding translations will have the same relation. This is the intuition behind the concept of "full-abstraction" ([48]), which serves as a natural criterion to establish the equivalent expressive power of two languages.

Full abstraction is not the only criterion to establish the relative expressiveness of calculi. Sometimes we might be interested in particular aspects or features of these calculi, such as the ability to handle certain tasks, or to represent certain kind of objects. In this sense we would consider a language to be at least as powerful as another if there is a translation from the latter to the former that preserves and/or reflects some essential property of the language being translated. Equivalently, we would consider that there is a gap in expressiveness between two languages if there is no translation that respects a particular property of interest. It might be the case that there are translations from one language to another, but we would still be inclined to say that there is gap when some essential aspect is not respected by any translation. In the context of concurrency and interaction, the properties that we are particularly interested in, are the capabilities of a system to simulate mobility.

It is in light of these concepts that we intend to compare the relative power of the languages within the π-calculus and CCP families. The object of this thesis to provide a map of the relations between different process calculi, and an attempt to establish some yet unknown relations, in particular, we are interested in exploring the possibility or impossibility of simulating mobility in CCP.

1.1 Contributions

The more general contributions of this thesis include:

- A uniform presentation of calculi in the two different paradigms for concurrency considered, namely mobile processes and concurrent constraints. This includes a uniform treatment of the semantics, by presenting both labelled and unlabelled transition systems for each calculi, and proving the equivalence of such systems by standard techniques. Also, the definition of sensible notions of behavioural equivalence in terms of the standard notion of *bisimulation* ([25]).
- A "map" of the known relations between the calculi discussed.

The specific contributions are:

- The establishment of a gap in expressiveness between concurrent languages with support for mobility (π-calculi) and Concurrent Constraint Programming languages. Specifically the impossibility of encoding internal mobility in CCP languages under some reasonable assumptions.
- The proof of a correspondence between labelled and unlabelled transition semantics for CCP.

Chapter 2

Background

This section provides the technical background on the basic concepts of calculi, operational semantics, bisimulation, translations and expressiveness. We assume the reader is familiar with basic set-theoretic concepts, inductive definitions, and the induction principle.

2.1 Calculi syntax and contexts

To design, build, analyze, and reason about systems we use *languages* or *calculi*. A *language* or *calculus* is a set of *terms* or *expressions* together with some associated meaning. The expressions or terms of a language represent components, agents or processes of a system. The *syntax* of a language describes the means for combining the basic terms into more complex terms to represent complex structure or behaviour of a system. The *semantics* describes what is the meaning of each term in the language, e.g. what entity or behaviour in a system is being modeled by a particular expression.

The syntax of a language is often given in an inductive way, defining first what the basic elements are, and then what the more complex elements are in terms of simpler expressions. It is common to use Backus-Naur Form (BNF) to describe the syntax of a language. For instance, assume a language $\mathcal{L}$, whose expressions are strings from the alphabet $\Sigma \stackrel{def}{=} \{a, b, \triangleleft, \otimes\} \cup \mathcal{V}$ with $\mathcal{V} \stackrel{def}{=} \{x, x_1, x_2, \cdots\}$. Suppose that the set of terms in $\mathcal{L}$ is defined inductively as follows (ignoring parenthesis):

$$\mathcal{L} \stackrel{def}{=} \{a, b\} \cup \{x \triangleleft p : x \in \mathcal{V}, p \in \mathcal{L}\} \cup \{p_1 \otimes p_2 : p_1, p_2 \in \mathcal{L}\}$$

This language will include terms such as b, $x \triangleleft a$, $(x \triangleleft b) \otimes a$, etc. The BNF for this language is given by:

$$P \quad ::= \quad a \quad | \quad b \quad | \quad x \triangleleft P \quad | \quad P_1 \otimes P_2$$

It will often be necessary to talk about the *context* of an agent. For a particular language, we define a context $\mathcal{C}$ as a term with a "hole" or place-holder $[\cdot]$. If P is a term in the language, $\mathcal{C}[P]$ is the term that results from replacing the $[\cdot]$ in $\mathcal{C}$ by P. These contexts can also be defined inductively by a BNF. For instance the set of $\mathcal{L}$-contexts for the language $\mathcal{L}$ described above, is given by:

$$\mathcal{C} \quad ::= \quad [\cdot] \quad | \quad x \triangleleft \mathcal{C} \quad | \quad \mathcal{C} \otimes P \quad | \quad P \otimes \mathcal{C}$$

For example, some contexts are $x \triangleleft [\cdot]$, $(x \triangleleft [\cdot]) \otimes a$, etc.[1]

The set of "minimal" contexts $\{x \triangleleft [\cdot],\ [\cdot] \otimes P,\ P \otimes [\cdot]\}$ is called the set of *elementary* contexts.

2.2 Operational semantics

When describing the semantics of a programming language or model of computation, it is common to do it by means of defining how the agents, processes, or programs of the language evolve. This approach is known as "operational semantics". We can think of the operational semantics as a "state transition system", in which we specify for some abstract machine how is it that its state, or configuration, evolves over time in order to execute programs of the language. This is in contrast with the so called "denotational semantics" in which the meaning of each term in the language is given by some element in an abstract domain.

While denotational semantics is more abstract as it is independent of any particular computational view, in the sense that the denotation of a program does not depend on how it is executed by a machine, operational semantics is very useful for that same reason: it gives us a concrete handle on the process of computation itself allowing us to reason about how the agents compute and interact.

In order to give an operational semantics, we have to define what is a "state" in the abstract machine, and how it evolves, i.e. given that the system is in a particular state, which state, or states, will be the next. Suppose that we are dealing with a language $\mathcal{L}$. We can think of our abstract machine as a "term rewriting system" in which computation is performed by means of replacing (or rewriting) the program to be executed with another, preferably simpler, program in $\mathcal{L}$. Thus, we can say that the states in the abstract machine are the terms themselves. Now, given two agents[2] E_1 and E_2 in $\mathcal{L}$, we want to formally express the notion that "E_1 evolves in a single computational step to become E_2". We do this by defining a "reduction" relation $\rightarrow_{\mathcal{L}} \subseteq \mathcal{L} \times \mathcal{L}$, so the notion of single computational step is captured by asserting $(E_1, E_2) \in \rightarrow_{\mathcal{L}}$. Normally we write $E_1 \rightarrow E_2$ for $(E_1, E_2) \in \rightarrow$, and we rescind from the subscript when it is clear from the context the language to which we are referring.

Even though defining the operational semantics in such way is very useful, it is somewhat restrictive. Sometimes we might be interested in defining how the agent

[1] Note that these are *single* hole contexts. In this thesis we do not deal with *multiple* hole contexts.

[2] We use "agents", "terms", "programs" and "processes" indistinctively.

evolves with respect to some specific context, for instance some environment that contains variables. If the set of such environments is Env, we would change the signature of our reduction relation to something like $\rightarrow \subseteq \mathcal{L} \times Env \times \mathcal{L} \times Env$ and write $(E_1, \sigma_1) \rightarrow (E_2, \sigma_2)$ for $(E_1, \sigma_1, E_2, \sigma_2) \in \rightarrow$, to mean "under the environment σ_1, the agent E_1 evolves into E_2 with environment σ_2".

This idea of enriching the reductions, can be thought of as adorning the reduction relation with some label(s); for instance some people write $E_1 \xrightarrow{(\sigma_1,\sigma_2)} E_2$ instead of $(E_1, \sigma_1) \rightarrow (E_2, \sigma_2)$. In general, we can choose some labels from a set $\mathcal{A}$ and say that $\rightarrow \subseteq \mathcal{L} \times \mathcal{A} \times \mathcal{L}$. These labels need not be a pair representing the state before, and the state after the transition. A frequent use is to consider $\mathcal{A}$ to be some set of *actions* that the agents can perform. This kind of relation is normally called "transition", or "commitment". Again, we write $E_1 \xrightarrow{\alpha} E_2$ for $(E_1, \alpha, E_2) \in \rightarrow$ to mean that E_1 performs the action α evolving into E_2.

In our abstract machine view of semantics, we could consider the labels as conditions on the current configuration, that need to be met for the machine to move to the next allowed state.

A formalization of the semantics using reduction or transition relations was given in [25] (although it is a much older concept).

Definition 2.1. *A structure $(\mathcal{L}, \rightarrow)$ where $\rightarrow \subseteq \mathcal{L} \times \mathcal{L}$ is called an* ***unlabelled transition system****, or UTS for short. A structure $(\mathcal{L}, \mathcal{A}, \rightarrow)$ where $\rightarrow \subseteq \mathcal{L} \times \mathcal{A} \times \mathcal{L}$ is called a* ***labelled transition system*** *or LTS for short.*

We specify the operational semantics of the language by defining which pairs of expressions belong to $\rightarrow$, in the case of UTS, or triples of expression-action-expression, in the case of LTS. It is customary to define this relation inductively by *inference rules*. An inference rule of the form

$$\frac{A_1 \quad A_2 \quad \ldots \quad A_n}{B}\ \text{RULE-NAME} \qquad \text{if <side conditions>}$$

should be read as "if A_1 and A_2 and ... and A_n then B". An inference rule without premises is an axiom. The rule can be applied only if the side conditions are met. For the definition of an operational semantics the A_i's and the B are assertions of the form $M \rightarrow M'$ (or $M \xrightarrow{\alpha} M'$ for an LTS) where M and M' are the different syntactic forms of the language.

When defining semantics this way, it is also customary to provide another relation between expressions called "structural congruence", written $\equiv \subseteq \mathcal{L} \times \mathcal{L}$. This is an equivalence relation that is supposed to group together (in equivalence classes) terms that we don't want to consider semantically different. It is important to note that this is a "static" equivalence, this is, it equates terms based solely on their syntax, and not on their dynamic behaviour. In section 2.3 we will talk about "dynamic" equivalence relations, when we approach the concept of bisimilarity. This approach of defining semantics in terms of a reduction system built upon a structural congruence, according to [31], was proposed in [27], and inspired on the "Chemical Abstract Machine" of Berry and Boudol ([5]).

We normally include structural congruence in the signature of the language, be it labelled or unlabelled, and provide an inference rule for reductions or transitions that allows us to change freely between structurally congruent terms. Such a rule generally takes the following form[3]:

$$\frac{M \xrightarrow{\alpha} M'}{N \xrightarrow{\alpha} N'}\text{CONGR} \quad \text{if } M \equiv N \text{ and } M' \equiv N'$$

The notions of reduction and transition represent a single computational step. We extend this notion to a sequence of steps, or *computation*. We use the symbol $\Rightarrow$ to stand for $\rightarrow^*$ denoting the transitive and reflexive closure of $\rightarrow$. Thus, the assertion $P \Rightarrow Q$, representing a computation from P to Q, means that there are terms R_1, R_2, ..., R_n such that $P \rightarrow R_1 \rightarrow R_2 \rightarrow ... \rightarrow R_n \rightarrow Q$, or that $P \equiv Q$. In the setting of labelled transition systems, a computation is a sequence of transitions $P \overset{\tilde{\alpha}}{\Rightarrow} Q$, where $\tilde{\alpha} = \alpha_1, \alpha_2, \cdots, \alpha_n$, is a sequence of actions, meaning that there are R_1, R_2, ..., R_n such that $P \xrightarrow{\alpha_1} R_1 \xrightarrow{\alpha_2} R_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{n-1}} R_n \xrightarrow{\alpha_n} Q$, or $P \equiv Q$. We often refer to $\rightarrow$ and $\xrightarrow{\alpha}$ as "strong" reductions or transitions, and to $\Rightarrow$ and $\overset{\alpha}{\Rightarrow}$ as "weak" reductions or transitions.

What kind of semantics is better? This is not easy to answer. LTS semantics gives us finer grained information, and allows us to reason locally, whereas UTS gives us a more general view of the semantics, and tends to be more intuitive. Another aspect in favor of LTS semantics is that given a term, we can represent its possible evolution paths as a graph in which nodes are terms, and edges are transitions, with their labels representing actions. An analogous graph for a UTS would not be as informative because edges would not have labels. It would tell us "where" the system can go, but not how.

An important justification for defining formally the semantics of a language is that we can use mathematical tools available to reason about and prove properties of programs. One useful tool is the principle of induction. In this context the use of this principle appears in two forms: "structural" induction, and induction on the derivations (also known as induction on the inference, or induction on the height of the proof tree). In the first case, we make induction based on the structure of the programs (according to the syntax). Hence it is generally used for proving static properties. For dynamic properties we generally use induction on derivations. In these, we analyze the possible cases for last inference, assuming the induction hypothesis for the premises of each rule, and for axioms.

Given the two forms of semantics for a particular language, we expect them to agree, i.e., at the appropriate level of abstraction, agents that have some behaviour according to UTS, must have the same behaviour according to LTS and vice-versa.

[3] This is the LTS form of the rule. For an UTS version, we simply do not write the α labels.

This is a property that must be proven, and generally involves induction on derivations.

2.3 Bisimulation

The notion of equivalence is essential to the semantics of a language. Unless we have a way of distinguishing terms or agents, we cannot claim that we have a well defined notion of "meaning" of an agent, or its power. We need to be able to determine whether two agents are equivalent in some sense.

We now address this issue of whether two agents should be considered "dynamically" or "behaviourally" equivalent. As pointed out in the introduction, this is a notion that, in the case of concurrent systems, should depend on the capabilities of interaction of an agent with its environment. So if we want to consider two processes as equivalent we have to rely on the idea that they "match" each other's actions. This idea of matching is formalized by the notion of *simulation*[4]. We will consider an LTS $(\mathcal{L}, \mathcal{A}, \equiv, \rightarrow)$.

Definition 2.2 (Simulation and similarity). *A binary relation $\mathcal{S} \subseteq \mathcal{L} \times \mathcal{L}$ is called a* ***simulation*** *iff for any terms $P, Q \in \mathcal{L}$, $P\mathcal{S}Q$ implies that if for any term $P' \in \mathcal{L}$ and any action $\alpha \in \mathcal{A}$ such that $P \stackrel{\alpha}{\rightarrow} P'$, then there is a term $Q' \in \mathcal{L}$ such that $Q \stackrel{\alpha}{\rightarrow} Q'$ and $P'\mathcal{S}Q'$.*

We say that Q ***simulates*** *P, or that P and Q are* ***similar****, written $P \dot{\prec} Q$, iff there is a simulation $\mathcal{S}$ such that $P\mathcal{S}Q$.*

The intuitive idea behind this concept is that Q simulates P in the sense that every action that P makes is matched by the same action in Q, reaching a similar state, i.e. a state that itself continues to simulate its counterpart.

We point out a few properties (Milner, [25]):

Proposition 2.3.

(i) *$\dot{\prec}$ is reflexive and transitive.*

(ii) *$\dot{\prec}$ is itself a simulation.*

(iii) *$\dot{\prec}$ is the largest simulation.*

Proof. (i) Similarity is reflexive because there is a simulation $\mathcal{S}$ such that for any $P \in \mathcal{L}$, $P\mathcal{S}P$. Namely that simulation is $\mathcal{S} = \{(P, P) | P \in \mathcal{L}\}$. It is easy to show this set is a simulation, because P can always match its own moves.
For transitivity, assume that $P \dot{\prec} Q$ and $Q \dot{\prec} R$. Then there are simulations $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $P\mathcal{S}_1Q$ and $Q\mathcal{S}_2R$. Define $\mathcal{S} \stackrel{def}{=} \mathcal{S}_1\mathcal{S}_2$, i.e. the composition of the given

[4] David Park and Robin Milner [25] are credited as the inventors of the notions of simulation and bisimulation.

simulations: $\mathcal{S} = \{(a,b) \mid \exists c.\ a\mathcal{S}_1 c \ \&\ c\mathcal{S}_2 b\}$. Clearly it is the case that $P\mathcal{S}R$. It now suffices to show that $\mathcal{S}$ is indeed a simulation. Take any $(a,b) \in \mathcal{S}$, and suppose that $a \stackrel{\alpha}{\to} a'$. Then there is a c such that $a\mathcal{S}_1 c$ and $c\mathcal{S}_2 b$. Knowing the first simulation we obtain that $c \stackrel{\alpha}{\to} c'$ and $a'\mathcal{S}_1 c'$. Then, by the second simulation, $b \stackrel{\alpha}{\to} b'$ and $c'\mathcal{S}_2 b'$. This implies that $(a',b') \in \mathcal{S}$, hence $\mathcal{S}$ is a simulation, so $P \dot{\prec} R$.

(ii) Assume that $P \dot{\prec} Q$. By definition of similarity, there is a simulation $\mathcal{S}$ where $P\mathcal{S}Q$. Hence whenever $P \stackrel{\alpha}{\to} P'$ we know that $Q \stackrel{\alpha}{\to} Q'$ with $P'\mathcal{S}Q'$. Since P' and Q' are related by a simulation, they are similar: $P' \dot{\prec} Q'$, thus satisfying the conditions of a simulation.

(iii) To see that similarity is maximal, it is enough to prove that

$$\dot{\prec} \ = \ \bigcup \{ \ \mathcal{S} \mid \mathcal{S} \text{ is a simulation } \}$$

To see this, note that $(a,b) \in \dot{\prec}$ if and only iff $(a,b) \in \mathcal{S}$ for some simulation $\mathcal{S}$; this means that $(a,b) \in \cup\{\mathcal{S} \mid \mathcal{S} \text{ is a simulation } \}$. This set is maximal because it contains all simulations.

$\square$

The notions of simulation and similarity are one-directional (although not necessarily antisymmetric). If we are interested in considering two agents as equivalent, they need to match each other's actions. An alternative is to consider "two-way simulation", i.e. the agents are equivalent if one simulates the other and vice-versa. This introduces some problems, as pointed out by the following example. Consider two machines P and Q as depicted in figure 2.1. It is easy to see that Q simulates P. The simulation is $\mathcal{S}_1 = \{(P_1,Q_1),(P_2,Q_2),(P_3,Q_3),(P_4,Q_4)\}$. The converse is also true: P simulates Q, and the simulation is

$$\mathcal{S}_2 = \{(Q_1,P_1),(Q_2,P_2),(Q_3,P_3),(Q_4,P_4),(Q_5,P_2),(Q_6,P_3),(Q_7,P_2),(Q_8,P_4)\}$$

The problem however is that two-way simulation is not an appropriate notion of equivalence because we would expect equivalent machines to be able to replace one another in any context, i.e. they must interact with the environment in the same way. But the example shows that this is not the case for all states of the machines. Consider for instance state Q_5. In P, state P_2 simulates Q_5, but in reality, no state in P really corresponds to Q_5. If the machine Q is initially presented with an action a and moves to state Q_5 and then it receives an action c, it cannot respond to it. Meanwhile, if P is presented with the same sequence of actions ac, it successfully ends in state P_4. Hence the two machines do not react with the same behaviour to external stimuli.

Given that two-way simulation does not seem to capture completely the notion of equivalence that we are after, we adopt a different concept: bisimilarity.

Definition 2.4 (Bisimulation and bisimilarity (Milner, [25])). *A simulation $\mathcal{S}$ is called a **bisimulation** iff $\mathcal{S}^{-1}$ is also a simulation. This is, $P\mathcal{S}Q$ implies that*

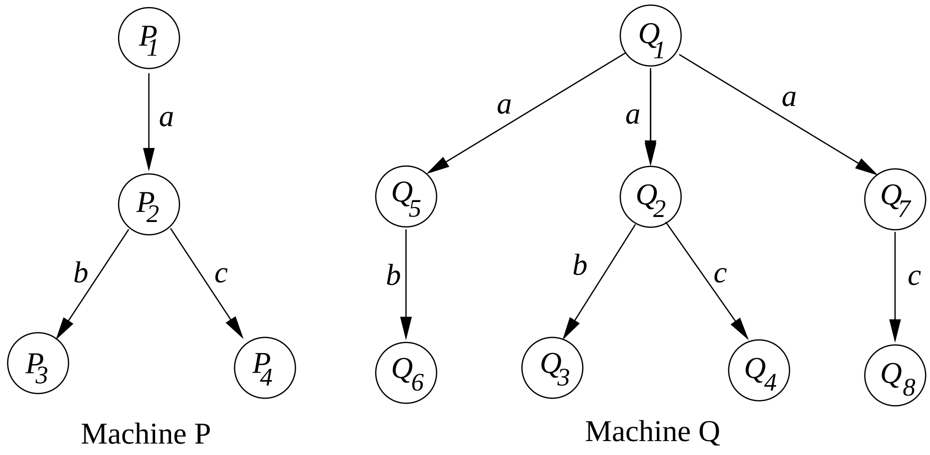

Figure 2.1: Are these machines equivalent?

(i) Whenever $P \stackrel{\alpha}{\to} P'$, *then* $Q \stackrel{\alpha}{\to} Q'$ *and* $P'\mathcal{S}Q'$

(ii) Whenever $Q \stackrel{\alpha}{\to} Q'$, *then* $P \stackrel{\alpha}{\to} P'$ *and* $P'\mathcal{S}Q'$

We say that P *and* Q *are* ***bisimilar****, written* $P\dot{\sim}Q$ *iff there is a bisimulation* $\mathcal{S}$ *such that* $P\mathcal{S}Q$.

The fact that this concept is useful to distinguish between agents, is given by establishing that $\dot{\sim}$ is an equivalence relation. The following has been proved by Milner ([25]).

Proposition 2.5.

(i) $\dot{\sim}$ *is an equivalence relation.*

(ii) $\dot{\sim}$ *is itself a bisimulation.*

(iii) $\dot{\sim}$ *is the largest bisimulation.*

Proof. (i) First we establish that $\dot{\sim}$ is a simulation. The argument is as in item (ii) in 2.3. Then, since $\dot{\sim}$ is a simulation, it is reflexive and transitive. Symmetry is easily established: given P and Q such that $P\dot{\sim}Q$ we know that there is a simulation $\mathcal{S}$ such that $\mathcal{S}^{-1}$ is also a simulation and $P\mathcal{S}Q$. This implies that $Q\mathcal{S}^{-1}P$, so there is a simulation $\mathcal{R} \stackrel{def}{=} \mathcal{S}^{-1}$ such that $Q\mathcal{R}P$ and its inverse $\mathcal{R}^{-1} = \mathcal{S}$ is also a simulation. Hence $Q\dot{\sim}P$.

(ii) Analogous to proposition 2.3.

(iii) Analogous to proposition 2.3.

□

Returning to the example of two-way simulation, we could have saidthat $P \not\dot{\sim} Q$, and hence should not be considered equivalent.

Recall from the previous section that it is common to present the operational semantics of a language together with a structural congruence $\equiv$ between agents, and a CONGR rule that allow us to replace structurally congruent agents and preserve transitions. It is easy to see that with such a rule, structural congruence is a bisimulation, because if $P \equiv Q$ and $P \xrightarrow{\alpha} P'$, and noting that $\equiv$ is reflexive so that $P' \equiv P'$, then by using a CONGR rule, $Q \xrightarrow{\alpha} P'$. In other words, $\equiv \subseteq \dot{\sim}$.

Congruence relations

If we consider two agents equivalent, we expect them to be mutually replaceable. If one of them exists within an environment, we should be able to take it out, and "plug-in" the other so that the overall system or environment behaves in the same way. In other words, any observer or external client should not be able to distinguish between the two agents. The interaction between agent and observer must remain the same.

This idea is formalized by the algebraic notion of *congruence*[5]. A congruence is an equivalence relation that is preserved by the operations in some domain, or in our context, an equivalence relation that is preserved by the operators of the language.

Definition 2.6. *Let $\mathcal{L}$ be a calculus, and $\cong \subseteq \mathcal{L} \times \mathcal{L}$ an equivalence relation between $\mathcal{L}$ terms. We say that $\cong$ is a* ***congruence*** *iff for any $P, Q \in \mathcal{L}$, whenever $P \cong Q$ then for all $\mathcal{L}$-contexts $\mathcal{C}$, $\mathcal{C}[P] \cong \mathcal{C}[Q]$.*

More succinctly we could have say that $\cong$ is closed under arbitrary contexts.

The notion of bisimilarity attempts to capture an appropriate notion of behavioural equivalence. However, we can only say that this goal is achieved if bisimilarity is a congruence in the language of interest. As we will see, this depends on the language under consideration. Sometimes it will be necessary to refine the notion of bisimilarity to accommodate or reflect the idiosyncrasies of the language, in order to obtain a congruence. Often it turns out that even after adapting the notion of bisimilarity to the language it still is not a congruence. When this is the case, we can "force" the equivalence to be a congruence by defining an equivalence relation on top of it. We say that the new equivalence is *induced* from the old one.

Definition 2.7 (Induced congruence). *Let $\mathcal{L}$ be a calculus, and $\dot{\sim} \subseteq \mathcal{L} \times \mathcal{L}$ an equivalence relation between $\mathcal{L}$ terms. We define a binary relation $\sim \subseteq \mathcal{L} \times \mathcal{L}$ as the smallest equivalence relation such that for any $P, Q \in \mathcal{L}$, $P \sim Q$ if and only if for all $\mathcal{L}$-contexts $\mathcal{C}$, it holds that $\mathcal{C}[P] \dot{\sim} \mathcal{C}[Q]$. We say that $\dot{\sim}$* ***induces the congruence*** *$\sim$.*

[5]The structural congruence relation mentioned before as a special kind of equivalence between agents, is defined for each particular calculus in terms of the more general concept of congruence provided here.

More succinctly we could have said that the congruence induced by an equivalence is the closure of the equivalence under all contexts.

Even if obtaining congruence relations is one of the most important objectives of semantics, sometimes we might find interesting and meaningful equivalence relations that, while not being congruences, still preserve some, or most of the operators of the language.

A great deal of research in concurrency and interaction has had these issues as a central theme.

2.4 Translations and Expressiveness

In this section we examine in more detail the concept of expressiveness. In the introduction we mentioned several approaches to this concept, but we decided to concentrate on the understanding of expressive power in terms of the capabilities of interaction between agents. In section 2.3 we introduced the notions of simulation and bisimulation as a means of describing the idea that one agent can interact with the environment (at least) in the same way as another. We are assuming that agents are being described within the same language. Note that here we are using the term "language", or "calculus" to refer to the set of all agents, not the set of strings of labels produced by a specific automaton. Our emphasis is on comparing process calculi rather than individual agents. To do so, we need to talk about translations between different calculi.

2.4.1 Full abstraction

Fix, for the moment, two languages $\mathcal{L}_1$ and $\mathcal{L}_2$. A *translation*, *embedding*, or *encoding* from $\mathcal{L}_1$ to $\mathcal{L}_2$ is simply a function $t : \mathcal{L}_1 \to \mathcal{L}_2$. We say that $\mathcal{L}_2$ is at least as expressive as $\mathcal{L}_1$ if there is a translation $t : \mathcal{L}_1 \to \mathcal{L}_2$ that satisfies certain criteria. This definition of course leaves a wide margin of freedom to determine what is a good translation (if it exists) depending on what the criteria are. As we will see, the results in expressiveness are based on the choice of criteria for translations. There are a few widely accepted concepts to describe or prove the "correctness" of a translation. These concepts, introduced below, are very generic, but their application varies from one language to another, and one translation to the next.

Usually the criteria for establishing the correctness of a translation is given in terms of the notions of equivalence, simulation or transitions for the languages. We expect the translation to "behave well" with respect to such relations, i.e. transitions in one language correspond to transitions in the other, and equivalent terms are translated into equivalent terms. This idea is generalized by the following properties.

Definition 2.8. *Let $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ be a translation , and consider two binary relations $\mathcal{R}_1 \subseteq \mathcal{L}_1 \times \mathcal{L}_1$ and $\mathcal{R}_2 \subseteq \mathcal{L}_2 \times \mathcal{L}_2$, we say that:*

- $[\![\cdot]\!]$ *is **complete** w.r.t* $\mathcal{R}_1, \mathcal{R}_2$ *iff for any* $E, F \in \mathcal{L}_1$, $E\mathcal{R}_1F$ *implies* $[\![E]\!]\mathcal{R}_2[\![F]\!]$.
- $[\![\cdot]\!]$ *is **adequate** w.r.t* $\mathcal{R}_1, \mathcal{R}_2$ *iff for any* $E, F \in \mathcal{L}_1$, $[\![E]\!]\mathcal{R}_2[\![F]\!]$ *implies* $E\mathcal{R}_1F$.
- $[\![\cdot]\!]$ *is **fully-abstract** w.r.t* $\mathcal{R}_1, \mathcal{R}_2$ *iff it is complete and adequate.*

We intend to convey the idea that $\mathcal{R}_1$ and $\mathcal{R}_2$ are related notions, such as transitions, and (bi)similarity. Obviously such notions are language dependent, hence the mention of two separate relations $\mathcal{R}_1$ and $\mathcal{R}_2$; but in general we are interested in comparing transitions in one language with transitions in the other, or similarity in the source with similarity in the target.

Sometimes it is useful to compare expressiveness in terms of a property that is not necessarily a binary relation. We could formalize this idea of respecting a property as follows.

Definition 2.9. *Given a translation* $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$, *and two properties (predicates)* $\theta_1 : \mathcal{L}_1 \to \{\boldsymbol{T}, \boldsymbol{F}\}$ *and* $\theta_2 : \mathcal{L}_2 \to \{\boldsymbol{T}, \boldsymbol{F}\}$, *we say that:*

- $[\![\cdot]\!]$ ***preserves*** θ_1, θ_2 *iff for any* $E \in \mathcal{L}_1$, $\theta_1(E)$ *implies* $\theta_2([\![E]\!])$.
- $[\![\cdot]\!]$ ***reflects*** θ_1, θ_2 *iff for any* $E \in \mathcal{L}_1$, $\theta_2([\![E]\!])$ *implies* $\theta_1(E)$.
- $[\![\cdot]\!]$ *is* θ_1, θ_2***-fully-abstract*** *iff it preserves and reflects* θ_1, θ_2.

Of course, the signature of the θ predicates could be more complex, depending on the property that we are interested in comparing for the languages.

Notice that we can see completeness as a preservation property, i.e. t is complete w.r.t. $\mathcal{R}_1, \mathcal{R}_2$ iff t preserves $\mathcal{R}_1, \mathcal{R}_2$. In the same way, adequacy is a reflection property.

2.4.2 Respecting the semantics

Let us examine the notions of completeness, adequacy, and full abstraction with respect to transitions, simulation, and equivalence. Completeness w.r.t. transitions, means that transitions are preserved, i.e. actions in the source language of the translation are matched by actions in the target language. Assume that $\mathcal{A}_1$ represents the set of actions in $\mathcal{L}_1$, and $\mathcal{A}_2$ is the set of actions in $\mathcal{L}_2$. Let $\rightarrow_1 \subseteq \mathcal{L}_1 \times \mathcal{A}_1 \times \mathcal{L}_1$ and $\rightarrow_2 \subseteq \mathcal{L}_2 \times \mathcal{A}_2 \times \mathcal{L}_2$ be the transition relations defining the operational semantics of each language. Furthermore, assume that there is a way to map actions in one language into actions of the other, i.e. a map $a : \mathcal{A}_1 \to \mathcal{A}_2$. Then, the notion of completeness as preservation of transitions is formally stated as follows:

$$\text{For any } P, P' \in \mathcal{L}_1,\ \alpha \in \mathcal{A}_1,\ P \xrightarrow{\alpha}_1 P' \ \text{ implies } \ t(P) \xrightarrow{a(\alpha)}_2 t(P')$$

This property means that the translation faithfully captures the dynamics of the source language. Notice the similarity of this concept with that of a standard simulation, as in definition 2.2. This is not a coincidence. Completeness, or preservation of transitions, can be seen as a generalization of the notion of simulation to a "cross"-language domain.

The symmetric notion of adequacy, representing a *soundness* criterion can also be expressed in a similar fashion.

$$\begin{array}{l}\text{For any } P \in \mathcal{L}_1, Q \in \mathcal{L}_2\ \beta \in \mathcal{A}_2,\\ \qquad \text{if } t(P) \xrightarrow{\beta}_2 Q \text{ then there are } P' \in \mathcal{L}_1, \alpha \in \mathcal{A}_1 \text{ such that } P \xrightarrow{\alpha}_1 P' \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{where } Q = t(P') \text{ and } \beta = a(\alpha)\end{array}$$

Adequacy as reflection of transitions, can be understood as stating that the behaviour of translated terms is meaningful in the source language

Another instance of these concepts corresponds to considering completeness and adequacy as preservation and reflection (respectively) of similarity and behavioural equivalence, instead of just transitions.

When we say that a translation is complete w.r.t similarity, we are saying that if in language $\mathcal{L}_1$, agent P simulates Q then in $\mathcal{L}_2$, $t(P)$ simulates $t(Q)$. Recall that the notion of simulation, allows us to replace one agent with another (although not necessarily the other way around). If P simulates Q, we can use P in place of Q in any environment. If we replace "simulates" by "is behaviourally equivalent", then both processes are interchangeable, and the environment cannot tell them apart. The preservation of this property by a translation, means that the the ability to replace is also preserved. If a translation is not complete w.r.t. equivalence, it means that there are two agents in the source language that behave in the same way with respect to their environment, but their translations react differently to the environment. Such a translation would not be very good.

A symmetric analysis can be made of adequacy. A translation that is not adequate w.r.t. behavioural equivalence, is one where there are behaviourally different terms in the source language mapped to terms that exhibit the same behaviour in the target language. This implies that the interaction with the environment has not been preserved, and hence, some expressiveness has been lost. For this, lack of adequacy is much worse than failure of completeness.

If a translation is not adequate, or complete, then it is not capturing correctly the semantics of the source language. If no fully abstract translation is possible, it means that the target language is unable to capture some essential feature(s) of the source language, and thus is less expressive.

From respecting transitions to respecting equivalence

We have seen two important instances of completeness and adequacy from the point of view of transitions and (bi)simulations. But (bi)similarity is defined in terms of transitions. How are all these notions related then? The following establishes one such relation, and is the base for proving that a given translation is fully-abstract.

Theorem 2.10. *If a translation preserves and reflects transitions then it is fully abstract with respect to bisimilarity.*

Proof. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages, and $[\![\cdot]\!]$ a translation from $\mathcal{L}_1$ to $\mathcal{L}_2$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the set of actions in each language respectively, and $a : \mathcal{A}_1 \to \mathcal{A}_2$ a map between such actions. We use the notation $\dot{\sim}_1$, and $\dot{\sim}_2$ for bisimilarity in $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Assume that $[\![\cdot]\!]$ preserves and reflects transitions.

(i) Completeness: We want to prove that for any P, and Q, $P \ \dot{\sim}_1 \ Q$ implies $[\![P]\!] \ \dot{\sim}_2 \ [\![Q]\!]$. We can do this in the standard way, by showing a bisimulation in $\mathcal{L}_2$ that contains $([\![P]\!], [\![Q]\!])$ given that P and Q are bisimilar. Specifically we want to show that $\mathcal{S} \stackrel{def}{=} \{([\![P]\!], [\![Q]\!]) : P \dot{\sim}_1 Q\}$ is a bisimulation. Take any $([\![P]\!], [\![Q]\!]) \in \mathcal{S}$. Suppose that $[\![P]\!] \stackrel{\beta}{\to}_2 R$. Then, since $[\![\cdot]\!]$ reflects transitions, $R = [\![P']\!]$ for some $P' \in \mathcal{L}_1$, $\beta = a(\alpha)$ for some $\alpha \in \mathcal{A}_1$ and $P \stackrel{\alpha}{\to} P'$. Now, since $P \ \dot{\sim}_1 \ Q$, we have that $Q \stackrel{\alpha}{\to} Q'$ with $P' \ \dot{\sim}_1 \ Q'$. So, by preservation of transitions, $[\![Q]\!] \stackrel{a(\alpha)}{\to} [\![Q']\!]$. Hence $([\![P']\!], [\![Q']\!]) \in \mathcal{S}$, because $P' \ \dot{\sim}_1 \ Q'$.
(ii) Adequacy: This is the exact dual of the one above.

$\square$

To our knowledge the converse has not been established, and it could be useful to prove negative expressiveness results.

The theorem and proof above are rather general, and might not be directly applicable to the languages being compared, since it depends on the notions of transitions and (bi)simulation. These may involve additional conditions that must be checked, e.g. late/early/weak bisimilarity. For example, when dealing with "weak" (bi)similarities, we might have several choices for the notion of "preserving" transitions; does it mean that one transition is matched by many, or that a sequence of transitions is matched by a sequence? All these details depend on the languages, however the proof above serves as the sketch for proofs of full-abstraction.

Encodings of a language in a sublanguage

When exploring the expressiveness of some calculus we face the question of which of its operators are strictly necessary and which are not. In such case, we define a sub-calculus or sublanguage of the original, as a (proper) subset of the full calculus. Then, to asses the power of the sub-calculus with respect to the original, we can use a notion of equivalence, similarity or bisimilarity of the full language, to compare terms of the sublanguage with terms of the full language. This allows us to prove full-abstraction with respect to the notion of equivalence used.

Lemma 2.11. *Consider $\mathcal{L}_1$ to be some language, and $\mathcal{L}_2 \subseteq \mathcal{L}_1$ a sublanguage. Let $\sim \subseteq \mathcal{L}_1 \times \mathcal{L}_1$ be an equivalence relation, and $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ a translation. If for all $P \in \mathcal{L}_1$, $P \sim [\![P]\!]$ then $[\![\cdot]\!]$ is fully-abstract w.r.t. $\sim$.*

Proof. We first prove completeness. Let $P_1, P_2 \in \mathcal{L}_1$ such that $P_1 \sim P_2$. By hypothesis $P_1 \sim [\![P_1]\!]$, and $P_2 \sim [\![P_2]\!]$. Since $\sim$ is an equivalence, by transitivity we have $[\![P_1]\!] \sim [\![P_2]\!]$. Adequacy is obtained with a dual argument. $\square$

2.4.3 About the types of translations

Besides full-abstraction, we might be interested also in other properties of a translation. For instance, we would like the translation of a term to be expressed only in terms of the translation of its sub-terms. Such a translation is called *compositional*. This is defined formally in terms of *contexts*.

Definition 2.12. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages, and $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ a translation between them. We say that $[\![\cdot]\!]$ is* ***compositional*** *if for any term $P \in \mathcal{L}_1$ and any $\mathcal{L}_1$-context $\mathcal{C}_1$, there is an $\mathcal{L}_2$-context $\mathcal{C}_2$ such that $[\![\mathcal{C}_1[P]]\!] = \mathcal{C}_2[[\![P]\!]]$.*

Often, we want a specific relation between the contexts of a compositional translation to emphasise the ties between the languages. For instance, when considering two concurrent languages, we might prefer a translation that is truly distributed in the sense that processes which are parallel in the source language, are translated into processes that are parallel in the target, and no mediator is involved. In other words we would like the translation to preserve the *parallel composition* operator from one language to the other. In general we might want this property for different operators, i.e. the context $\mathcal{C}_1$ should be mapped to $\mathcal{C}_2$ in a "uniform" way. The following definition formalizes this for binary operators.

Definition 2.13. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two such languages, and $\otimes_1$ and $\otimes_2$ be two binary operators in each language. Then we say that a translation $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ is said to be* ***uniform w.r.t.*** $\otimes_1$ ***and*** $\otimes_2$ *if for all $P, Q \in \mathcal{L}_1$, $[\![P \otimes_1 Q]\!] = [\![P]\!] \otimes_2 [\![Q]\!]$.*

More succinctly we could have said that $[\![\cdot]\!]$ is an homomorphism between $(\mathcal{L}_1, \otimes_1)$ and $(\mathcal{L}_2, \otimes_2)$.

This property however, might be too strong. It might rule out many meaningful translations. In particular, a translation might require some kind of mediator agent between two agents in a parallel composition, i.e. the translation might be defined as $[\![P \otimes_1 Q]\!] \stackrel{def}{=} [\![P]\!] \otimes_2 M \otimes_2 [\![Q]\!]$, where M is the mediator introduced by the translation.

Not all operators in a language are binary. We could conceive this notion of uniformity also for unary operators such as *restriction*. Usually languages provide some means of hiding names from the environment so that agents have *private* or *local* variables, not (directly) accessible to the external world. A common notation for such an operator, in the context of π-calculi, is $\nu x.P$, where ν is the restriction operator, x is the name to hide, and P is the body of the agent. In a translation that is uniform with respect to restriction, hiding is preserved: $[\![\nu_1 x.P]\!] = \nu_2 x.[\![P]\!]$, where ν_1 and ν_2 are the restriction operators for the source and target languages. As with the parallel composition example, this kind of translation is also restrictive, because names might require different handling in the languages. Such translations often introduce "handler" processes, for instance as in $[\![\nu_1 x.P]\!] \stackrel{def}{=} \nu_2 x.(H(x) \otimes_2 [\![P]\!])$, where H is the handler for x.

Chapter 3

The π-calculus

The π-calculus was proposed by Milner, Parrow and Walker [30] in order to express explicitly the notion of mobility absent from Milner's previous work in his Calculus of Communicating Systems or CCS for short ([25]).

The π-calculus contains three basic entities: names, actions, and processes or agents. A system described by an expression in the π-calculus is a network of processes which communicate through channels. Names represent channels, and processes are composed by actions. Names also represent variables and constants. There is no need, in the theory for considering these separately. Since the object of study of these foundational models is the interaction between parts of a system, the only actions considered are communicating actions, specifically, actions for *sending* and *receiving* messages through the channels.

Definition 3.1. *Assume an infinite set $\mathcal{N}_\pi$ of names $u, v, w, x, y, z, \ldots$. We denote sequences of names by $\tilde{x}$ or $\vec{x}$. The set of π-calculus terms for processes, ranged over by $P, Q, R, \ldots$ and the set of actions ranged over by α are defined by the syntax shown in table 3.1. $\mathcal{P}_\pi$ denotes the set of process terms (P) and $\mathcal{A}_\pi$ denotes the set of actions (α).*

Informal description of the semantics

Informally terms should interpreted as follows: 0 is the process that does nothing. An agent of the form $\alpha.P$ is a process that performs the action α and then behaves like P. The action α is often called the *prefix* of $\alpha.P$. The prefix $\nu x.$ is called the *restriction operator*. In an agent of the form $\nu x.P$, the name x is *local* or *private* so there is no possible interaction of this agent with the external world through this channel. A process of the form $P \mid Q$ represents the parallel composition of the processes P and Q, that is, the two processes execute concurrently. The agent $P + Q$, called *summation*, represents non-deterministic choice, so it behaves like either P or Q. Intuitively, in a summation the first agent that performs an action continues and the other doesn't. The match and mismatch operators correspond to a limited form of

P	$::=$	0	Nil
	$\mid$	$\alpha.Q$	Prefix
	$\mid$	$\nu\tilde{x}.Q$	Restriction
	$\mid$	$Q \mid R$	Parallel Composition
	$\mid$	$Q + R$	Summation
	$\mid$	$[x = y]Q$	Match
	$\mid$	$[x \neq y]Q$	Mismatch
	$\mid$	$A(\tilde{x})$	Procedural call

α	$::=$	τ	Silent action
	$\mid$	$u(\tilde{x})$	Input
	$\mid$	$\overline{u}\langle\tilde{x}\rangle$	Output

Table 3.1: The syntax of π-calculus

conditional that represents testing for name equality and inequality respectively, i.e. the process $[x = y]Q$ blocks until the condition is satisfied (similarly for $[x \neq y]Q$. Finally, the term $A(\tilde{y})$ corresponds to the intuitive notion of "calling" an agent that has been defined by an equation of the form

$$A(x_1, x_2, ..., x_n) \stackrel{def}{=} P$$

this is, replacing the occurrence of $A(y_1, y_2, ..., y_n)$ by the body of A's definition P, making the appropriate substitution of parameters, i.e. the x's in P replaced by the y's.

The three actions are: 1) *input* or *receive*, represented by $u(x)$ in which u is the name of the channel, and x is a name to be received; 2) *output* or *send* is represented by $\overline{u}\langle x\rangle$ in which u is the name of the channel and x is the name sent; and 3) the *silent* action τ which represents an internal communication. For input and output actions, u is called the *subject* of the action and x is the *object*.

Notice that there is no distinction between names, constants, and variables. It has been shown that this distinction is not necessary, and it would only complicate the theory behind the calculus. Notice also that there are no data types or means to construct data structures, and that the calculus is *first order*, that is, the only thing that can be passed around are names, but it is not possible to send a process through a channel. Nonetheless, the π-calculus is expressive enough to allow the definition of arbitrary data structures. It is also possible to define a type system for it, and higher-order variants have been studied. Process calculi in which it is only possible to communicate names are called *name-passing calculi* and the higher order variants are called *agent-passing calculi*.

Communication occurs in a *synchronous* manner. This means that both input and output are blocking operations. When an agent has as prefix an input action, it is blocked until some other agent is sending something through the given channel. In the synchronous model of communication the sending action is also blocking so an agent with an output prefix is also blocked until there is an agent ready to receive through the channel in question. This is usually known as *rendezvous* and is analogous to communication through telephone. In asynchronous communication, on the contrary, the output operation is non-blocking and is akin of communication by mail. In chapter 4 the asynchronous variant of the π-calculus is discussed.

The input prefix and the restriction operator are binding. In an agent of the form $u(x).P$ or $\nu x.P$ we say that the *scope* of x is P, and every occurrence of x in P is said to be *bound*. A variable that is not bound is *free*. If a name occurs free in an agent, as the subject of some action, we can say that this name is a port to the external world, but if it is bound, no interaction with the external world can occur through this name.

With this description, let us take a closer look at the role of names. As mentioned above, names stand for channels, as well as variables and constants. One can view names as "ports" through which agents communicate. When considering an agent of the form $x(z).P$, we say that the process is *located* at x. In that sense x is the "address" of the agent, and thus, the notion of name, corresponds informally to the traditional notion of pointer. However, this notion is slightly different, since we can have several processes located at the same name. In such case, any third party can interact, non-deterministically, with any of the agents sharing the same port. Under this view of names, the match and mismatch operators correspond to testing for pointer equality and inequality respectively.

Some abbreviations and terminology

Another important aspect is that the calculus described here is *polyadic* which means that it is possible to send/receive several names in a single action. This is opposed to *monadic* communication in which it is possible to send only one name at a time. In the full π-calculus that we are describing now, this is not an important distinction, since it is possible to encode the polyadic calculus in the monadic fragment. To obtain polyadicity with only the monadic fragment of the calculus the only important thing is to setup a private channel, so that other processes that might be listening through the original channel don't get a chance to interfere with the transmission of the sequence of values. This is shown below.

$$
\begin{aligned}
\overline{u}\langle x_1 x_2 \cdots x_n \rangle &\stackrel{def}{=} \nu p.\overline{u}\langle p \rangle.\overline{p}\langle x_1 \rangle.\overline{p}\langle x_2 \rangle.\cdots.\overline{p}\langle x_n \rangle \\
u(x_1 x_2 \cdots x_n) &\stackrel{def}{=} u(p).p(x_1).p(x_2).\cdots.p(x_n)
\end{aligned}
$$

Original syntax	Abbreviation
$u()$	u
$\overline{u}\langle\rangle$	$\overline{u}$
$\nu x.\overline{u}\langle x\rangle$	$\overline{u}(x)$
$\alpha_1.\alpha_2.\cdots\alpha_n.0$	$\alpha_1.\alpha_2.\cdots\alpha_n$
$\nu x_1.\nu x_2.\cdots\nu x_n.P$	$\nu x_1 x_2\cdots x_n.P$
$P_1 \mid P_2 \mid \cdots \mid P_n$	$\Pi_{i=1}^n P_i$
$P_1 + P_2 + \cdots + P_n$	$\Sigma_{i=1}^n P_i$

Table 3.2: Some syntactic abbreviations

In the rest of this chapter we restrict ourselves to the monadic π-calculus. This does not change the semantics, since, as we have seen, we can simulate the emission and reception of multiple names with monadic input and output.

There are other common abbreviations summarized in table 3.2.

We say that a summation is *guarded* if all the P_i's are of the form $\alpha_i.Q_i$. We say that a summation is *input-guarded* (respectively *output-guarded*) if all the actions α_i are input actions (output actions respectively). We say that a summation is *mixed* if it includes both input and output guards. We call the π-calculus that allows only input-guarded choice the π^{inp}-calculus. Similarly for the π-calculus that allows only output-guarded choice the π^{out}-calculus, and π^{mix} for the calculus allowing mixed choice. A fourth variant is called the π^{sep}-calculus, which allows both input and output guarded summations, but not mixed choice.

An important aspect of the language is the use of procedural call expressions. When we admit recursive definitions we obtain the full computational power of Turing machines. An important derived construct is the *replication* operator, defined as follows: $!\,P \stackrel{def}{=} P \mid\, !\,P$.

The notation $Q\{x/y\}$ represents the agent Q with all free occurrences of y substituted by x. So given a definition $A(x_1, x_2, ..., x_n) \stackrel{def}{=} P$ in which the names x_i appear free in the agent P, a term $A(y_1, y_2, ..., y_n)$ is equivalent to the term $P\{y_1/x_1, y_2/x_2, ..., y_n/x_n\}$.

Communication and mobility

Computation proceeds through communication. We express computation with a *reduction* relation or a *transition* relation. We write the fact that an agent P becomes or evolves into an agent Q as $P \rightarrow Q$. So the most important element in the reduction of terms in the π-calculus is the communication interaction:

$$\overline{u}\langle x\rangle.P \mid u(y).Q \quad \rightarrow \quad P \mid Q\{x/y\}$$

The statement above represents precisely our intended meaning for communication: the left-hand agent sends a name x through the channel u while the right-hand agent receives through the same channel u the name x, so all free occurrences of y in Q are replaced by x.

How does the π-calculus model mobility? Names stand for constants and variables, but they stand also for channels. Since there is no formal distinction between these, one can freely pass channel names in a communication. We can see this through an example. Imagine that we have an agent of the form

$$\nu x.(P \mid \overline{u}\langle x\rangle.Q) \mid u(y).R$$

Furthermore, assume that x appears free in both P and Q and is a channel connecting them. Assume also that y occurs free in R. For instance we can have

$$\begin{aligned} P &\stackrel{def}{=} \overline{x}\langle k\rangle.\overline{x}\langle l\rangle \\ Q &\stackrel{def}{=} x(a).Q' \\ R &\stackrel{def}{=} y(b).R' \end{aligned}$$

So a first reduction sends x through u, making the channel x accessible to R:

$$\nu x.(P \mid Q \mid R\{x/y\})$$

Which we can rewrite, according to the definitions, as:

$$\nu x.(\overline{x}\langle k\rangle.\overline{x}\langle l\rangle \quad \mid \quad x(a).Q' \quad \mid \quad x(b).R'\{x/y\})$$

Notice that the first action of R has been changed to $x(b)$. Now both the second and third agents in the composition are "listening" through x. Hence the channel x "moved", and it is now connecting not only P and Q but also R, i.e. the topology of the network evolved.

An interesting phenomenon found in the π-calculus known as *scope extrusion* is possible thanks to mobility. The idea is that an agent can pass a private name to another agent, but even though it is private, the receiving agent gains access to it, thus the scope of the private name is extended beyond that of its sender:

$$u(y).Q \mid \nu x.(\overline{u}\langle x\rangle.P) \;\rightarrow\; \nu x.(Q\{x/y\} \mid P)$$

Although at first sight one can think that π-calculus is a model of message-passing communication, a closer look will show that it can be said to embed both message passing and shared variables to an extent. In an agent of the form $\nu x.(P \mid Q)$ the name x is effectively shared by both P and Q. This statement will be strengthened in future chapters when we show how the π-calculus is able to simulate some paradigms with shared memory.

3.1 Semantics

We now define formally the behaviour of processes in terms of a *congruence* relation and a *reduction* relation, but first some preliminary definitions.

3.1.1 Substitution of names

The π-calculus is centered on the notion of names. They represent channels of communication between agents. They can be viewed as constants, and variables. For this reason one fundamental operation is that of name substitution. It's role will become apparent when we present the UTS and LTS for the language.

Definition 3.2. *The set of* ***bound names*** *and the set of* ***free names of an action*** *α, denoted $bn(\alpha)$ and $fn(\alpha)$ respectively, are defined as follows:*

$$
\begin{array}{ll}
bn(\tau) \stackrel{def}{=} \emptyset & fn(\tau) \stackrel{def}{=} \emptyset \\
bn(\overline{u}\langle x\rangle) \stackrel{def}{=} \emptyset & fn(\overline{u}\langle x\rangle) \stackrel{def}{=} \{u, x\} \\
bn(u(x)) \stackrel{def}{=} \{x\} & fn(u(x)) \stackrel{def}{=} \{u\}
\end{array}
$$

We extend this notion for π-terms in a straightforward way:

Definition 3.3. *The set of* ***bound names*** *and the set of* ***free names of a π-term*** *P, denoted $bn(P)$ and $fn(P)$ respectively, are defined as follows:*

$$
\begin{array}{llll}
bn(0) & \stackrel{def}{=} \emptyset & fn(0) & \stackrel{def}{=} \emptyset \\
bn(\nu x.P) & \stackrel{def}{=} \{x\} \cup bn(P) & fn(\nu x.P) & \stackrel{def}{=} fn(P) \setminus \{x\} \\
bn(\alpha.P) & \stackrel{def}{=} bn(\alpha) \cup bn(P) & fn(\alpha.P) & \stackrel{def}{=} (fn(\alpha) \cup fn(P)) \setminus bn(\alpha) \\
bn(P \mid Q) & \stackrel{def}{=} bn(P) \cup bn(Q) & fn(P \mid Q) & \stackrel{def}{=} fn(P) \cup fn(Q) \\
bn(P\ +\ Q) & \stackrel{def}{=} bn(P) \cup bn(Q) & fn(P\ +\ Q) & \stackrel{def}{=} fn(P) \cup fn(Q) \\
bn([x = y]P) & \stackrel{def}{=} bn(P) & fn([x = y]P) & \stackrel{def}{=} fn(P) \cup \{x, y\} \\
bn([x \neq y]P) & \stackrel{def}{=} bn(P) & fn([x \neq y]P) & \stackrel{def}{=} fn(P) \cup \{x, y\}
\end{array}
$$

The set of all names in a term P is denoted by $n(P)$.

Notice that $n(P) = fn(P) \cup bn(P)$ and $n(\alpha) = fn(\alpha) \cup bn(\alpha)$.

As in the λ-calculus we have to be careful about substitution of names, because a free name might become bound in a substitution, resulting in a term that is not equivalent to the correct one. The definition below takes care of this, defining substitution in a fashion that avoids capture. This is done in the standard way.

Definition 3.4. *A **substitution** is a function $\sigma : \mathcal{N}_\pi \to \mathcal{N}_\pi$. We write the substitution $\sigma = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ as $\{y_1/x_1, y_2/x_2, ..., y_n/x_n\}$. The term $P\sigma$ stands for the agent P with all free occurrences of x_i replaced by $\sigma(x_i)$, changing bound names in case some name is captured. Substitution over terms is defined as shown in table 3.3.*

We assume that the general substitution $P\{y_1/x_1, y_2/x_2, ..., y_n/x_n\}$ can be read as $P\{y_1/x_1\}\{y_2/x_2\}...\{y_n/x_n\}$. The process of renaming bound names of a term is called α-conversion, and we say $P \equiv_\alpha Q$ to mean that Q can be obtained from P by α-conversion. Formally, if $x' \notin n(P)$ and $x \notin bn(P)$.

$$\nu x.P \equiv_\alpha \nu x'.P\{x'/x\}$$
$$u(x).P \equiv_\alpha u(x').P\{x'/x\}$$

3.1.2 Process congruence

Since we are interested in considering processes as equivalent if they have the same behaviour with respect to the environment, we need to define a notion of π-contexts for representing the environment in which the agent lives.

Definition 3.5. *The set of **process contexts**, ranged over by $\mathcal{C}, \mathcal{C}_0, \mathcal{C}_1, \cdots$ is defined by the following syntax:*

$$\begin{aligned}\mathcal{C} \quad ::= \quad & [\cdot] \quad | \quad \alpha.\mathcal{C} \quad | \quad [x = y]\mathcal{C} \quad | \quad [x \neq y]\mathcal{C} \quad | \quad \nu x.\mathcal{C} \quad | \quad P \,|\, \mathcal{C} \quad | \quad \mathcal{C} \,|\, P \\ | \quad & P + \mathcal{C} \quad | \quad \mathcal{C} + P\end{aligned}$$

*where x, α, and P are names, actions and processes according to definition 3.1. If Q is a term, $\mathcal{C}[Q]$ is the term that results from replacing the placeholder $[\cdot]$ in $\mathcal{C}$ by Q. The contexts $\alpha.[\cdot], [x = y][\cdot], [x \neq y][\cdot], \nu x.[\cdot], P \mid [\cdot], [\cdot] \mid P, P + [\cdot], and [\cdot] + P$ are called **elementary contexts**.*

Our first notion of equivalence, albeit a static one, is that of *process congruence*. It represents a substitution property, this is, equivalent agents under process congruence can be freely substituted for each other in any process context without changing the behaviour of the entire system.

Definition 3.6 (π process congruence ([29])). *A **process congruence** $\cong \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is an equivalence relation among agents such that for all $P, P' \in \mathcal{P}_\pi$ if $P \cong P'$ then:*

(i) For any action α, $\alpha.P \cong \alpha.P'$.

(ii) For any names x, y,

- $\nu x.P \cong \nu x.P'$

$$
\begin{aligned}
0\{y/x\} &\stackrel{def}{=} 0 \\
(P \mid Q)\{y/x\} &\stackrel{def}{=} P\{y/x\} \mid Q\{y/x\} \\
(P \ + \ Q)\{y/x\} &\stackrel{def}{=} P\{y/x\} \ + \ Q\{y/x\} \\
(\nu x.P)\{y/x\} &\stackrel{def}{=} \nu x.P \\
(\nu y.P)\{z/x\} &\stackrel{def}{=} \nu y.P\{z/x\} \\
&\qquad\qquad \text{if } (x \neq y \text{ and } y \neq z) \text{ or } x \notin fn(P) \\
(\nu y.P)\{y/x\} &\stackrel{def}{=} \nu y'.P\{y'/y\}\{y/x\} \\
&\qquad\qquad \text{if } x \neq y \text{ and } x \in fn(P) \\
&\qquad\qquad \text{where } y' \notin fn(P),\ y' \neq x \text{ and } y' \neq y \\
(\overline{u}\langle x\rangle.P)\{y/u\} &\stackrel{def}{=} \overline{y}\langle x\rangle.P\{y/u\} \\
(\overline{u}\langle x\rangle.P)\{y/x\} &\stackrel{def}{=} \overline{u}\langle y\rangle.P\{y/x\} \\
(u(x).P)\{y/u\} &\stackrel{def}{=} y(x).P\{y/u\} \\
(u(x).P)\{y/x\} &\stackrel{def}{=} u(x).P \\
(u(y).P)\{z/x\} &\stackrel{def}{=} u(y).P\{z/x\} \\
&\qquad\qquad \text{if } (x \neq y \text{ and } y \neq z) \text{ or } x \notin fn(P) \\
(u(y).P)\{z/x\} &\stackrel{def}{=} u(y').P\{y'/y\}\{z/x\} \\
&\qquad\qquad \text{if } x \neq y \text{ and } x \in fn(P) \\
&\qquad\qquad \text{where } y' \notin fn(P),\ y' \neq x \text{ and } y' \neq y \\
([x = y]P)\{z/x\} &\stackrel{def}{=} [z = y]P\{z/x\} \\
([x = y]P)\{z/y\} &\stackrel{def}{=} [x = z]P\{z/y\} \\
([x \neq y]P)\{z/x\} &\stackrel{def}{=} [z \neq y]P\{z/x\} \\
([x \neq y]P)\{z/y\} &\stackrel{def}{=} [x \neq z]P\{z/y\} \\
([x = y]P)\{z/u\} &\stackrel{def}{=} [x = y]P\{z/u\} \\
&\qquad\qquad \text{if } u \neq x \text{ and } u \neq y \\
([x \neq y]P)\{z/u\} &\stackrel{def}{=} [x \neq y]P\{z/u\} \\
&\qquad\qquad \text{if } u \neq x \text{ and } u \neq y
\end{aligned}
$$

Table 3.3: Substitution of free names in a term

- $[x = y]P \cong [x = y]P'$
- $[x \neq y]P \cong [x \neq y]P'$

(iii) For any agent Q,

- $P \mid Q \cong P' \mid Q$
- $Q \mid P \cong Q \mid P'$
- $P + Q \cong P' + Q$
- $Q + P \cong Q + P'$

The following proposition simply formalizes the intuition behind process congruence as satisfying the substitution property in contexts.

Proposition 3.7 (([29])). *An equivalence relation $\cong$ is a process congruence if and only if for any pair of terms $P, Q \in \mathcal{P}_\pi$ and for any context $\mathcal{C}$, $P \cong Q$ implies $\mathcal{C}[P] \cong \mathcal{C}[Q]$.*

Proof. ($\Leftarrow$) Assume that for any P, Q, and any context $\mathcal{C}$, if $P \cong Q$ then $\mathcal{C}[P] \cong \mathcal{C}[Q]$. Fix A, B such that $A \cong B$. Then by our assumption, for any context $\mathcal{C}[A] \cong \mathcal{C}[B]$. In particular this holds for the elementary contexts: $\alpha.A \cong \alpha.B$, $[x = y]A \cong [x = y]B$, $[x \neq y]A \cong [x \neq y]B$, $\nu x.A \cong \nu x.B$, $P \mid A \cong P \mid B$, etc. This means that $\cong$ is indeed a process congruence.

($\Rightarrow$) By induction on the structure of the context. Assume that $\cong$ is a process congruence, and fix A, B such that $A \cong B$.

Case 1: $\mathcal{C} = [\cdot]$. Hence $\mathcal{C}[A] = A \cong B = \mathcal{C}[B]$.

Case 2: $\mathcal{C} = \alpha.\mathcal{C}'$. Then $\mathcal{C}[A] = \alpha.\mathcal{C}'[A]$ and $\mathcal{C}[B] = \alpha.\mathcal{C}'[B]$. By induction hypothesis, $\mathcal{C}'[A] \cong \mathcal{C}'[B]$. Hence $\alpha.\mathcal{C}'[A] \cong \alpha.\mathcal{C}'[B]$ because $\cong$ is a process congruence. Therefore $\mathcal{C}[A] \cong \mathcal{C}[B]$.

Case 3: $\mathcal{C} = P \mid \mathcal{C}'$. Then $\mathcal{C}[A] = P \mid \mathcal{C}'[A]$ and $\mathcal{C}[B] = P \mid \mathcal{C}'[B]$. By induction hypothesis, $\mathcal{C}'[A] \cong \mathcal{C}'[B]$. Again, since $\cong$ is assumed to be a process congruence, $P \mid \mathcal{C}'[A] \cong P \mid \mathcal{C}'[B]$, which means that $\mathcal{C}[A] \cong \mathcal{C}[B]$.

The rest of the cases are similar.

□

3.1.3 Structural congruence

Built upon the notion of process congruence is the structural congruence for π-terms. This is also a static notion of equivalence, and its intention is to extend the previous concept to capture other "primitive" aspects of what we expect in a reasonable equivalence of agents. This is also found in [30], [26], and [29].

Definition 3.8 (π structural congruence). *The structural congruence* $\equiv \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ *is the process congruence that satisfies the following axioms:*

(i) $P \equiv Q$ *if* $P \equiv_\alpha Q$

(ii) $(\mathcal{P}_\pi, \ |\ , \ 0)$ *is an Abelian (commutative) monoid:*

- $P \mid 0 \equiv P$
- $P \mid Q \equiv Q \mid P$
- $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$

(iii)
- $P + Q \equiv Q + P$
- $(P + Q) + R \equiv P + (Q + R)$

(iv) $\nu x.0 \equiv 0$

(v) $\nu x.\nu y.P \equiv \nu y.\nu x.P$

(vi) $P \mid \nu x.Q \equiv \nu x.(P \mid Q)$ *if* $x \notin fn(P)$

(vii) $[x = y]\nu z.P \equiv \nu z.[x = y]P$ *if* z *is a name other than* x *and* y*.*

(viii) $[x \neq y]\nu z.P \equiv \nu z.[x \neq y]P$ *if* z *is a name other than* x *and* y*.*

The (vi) axiom is called "scope extrusion", or "Frobenius reciprocity", and is essential to the modelling of mobility.

The equivalence classes induced by this definition, have "representative" terms. This is the role of "standard forms", also known as "prenex forms".

Definition 3.9. *A term of the form* $\nu\vec{x}.(P_1 \mid P_2 \mid \cdots \mid P_n)$ *where each* P_i *is a summation is said to be in* ***standard form****.*

Proposition 3.10. *Every term is structurally congruent to a standard form.*

Proof. By applying scope extrusion, and alpha-conversion when necessary, we can extract the restriction operators that are not within a summation to the outermost level of the expression. □

3.1.4 Unlabelled Transition Semantics: Reduction

The UTS $(\mathcal{P}_\pi, \equiv, \rightarrow)$ is given as follows. The set $\mathcal{P}_\pi$ of terms was given in definition 3.1, and $\equiv$ is the structural congruence.

Definition 3.11. *The reduction relation* $\rightarrow \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ *is the smallest relation over processes satisfying the rules in table 3.4.*

$$\frac{}{\tau.P \to P}\ \text{TAU}$$

$$\frac{}{(u(x).P\ +\ R) \mid (\overline{u}\langle y\rangle.Q\ +\ S) \to P\{y/x\} \mid Q}\ \text{COMM}$$

$$\frac{P \to P'}{P \mid Q \to P' \mid Q}\ \text{PAR} \qquad \frac{P \to P'}{P\ +\ Q \to P'}\ \text{SUM}$$

$$\frac{P \to P'}{\nu x.P \to \nu x.P'}\ \text{RESTR} \qquad \frac{P\{\tilde{y}/\tilde{x}\} \to P'}{A(\tilde{y}) \to P'}\ \text{ID} \quad \text{if } A(\tilde{x}) \stackrel{def}{=} P$$

$$\frac{P \to P'}{[x = x]P \to P'}\ \text{MTCH} \qquad \frac{P \to P'}{[x \neq y]P \to P'}\ \text{MISMTCH if } x \neq y$$

$$\frac{P \to Q}{P' \to Q'}\ \text{CONGR} \quad \text{if } P \equiv P' \text{ and } Q \equiv Q'$$

Table 3.4: π-calculus reduction rules

The COMM axiom specifies how agents communicate. The RESTR rule defines the behaviour of the restriction operator. The PAR rule says that if one agent can perform an action, then the parallel composition of the agent with others can perform the action. Similarly, the SUM rule expresses the non-deterministic choice of course of action which follows an agent with the sum operator. The final rule states that agents that are structurally congruent have the same behaviour.

If the reader is concerned about the apparent lack of symmetric rules for PAR and SUM, notice that these are recovered easily by the use of CONGR as follows:

$$\frac{\dfrac{Q \to Q'}{Q \mid P \to Q' \mid P}\ \text{PAR} \qquad \dfrac{}{Q \mid P \equiv P \mid Q} \qquad \dfrac{}{Q' \mid P \equiv P \mid Q'}}{P \mid Q \to P \mid Q'}\ \text{CONGR}$$

For the SUM rule we can reconstruct the symmetric one in an analogous way. From now on, we won't show the use of CONGR explicitly, since it would only add clutter to the proofs.

3.1.5 Labelled Transition Semantics

Now we provide an LTS $(\mathcal{P}_\pi, \mathcal{A}_\pi, \equiv, \stackrel{\alpha}{\to})$ for the language. We keep the same notion of congruence from definition 3.8. The reference is [30].

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\,\text{PREF}_t \qquad \frac{P \xrightarrow{u(x)} P' \quad Q \xrightarrow{\overline{u}\langle y\rangle} Q'}{P \mid Q \xrightarrow{\tau} P'\{y/x\} \mid Q'}\,\text{COMM}_t$$

$$\frac{P \xrightarrow{\alpha} P'}{P\ +\ Q \xrightarrow{\alpha} P'}\,\text{SUM}_t \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\,\text{PAR}_t \text{ if } bn(\alpha) \cap fn(Q) = \emptyset$$

$$\frac{P \xrightarrow{\alpha} P'}{\nu x.P \xrightarrow{\alpha} \nu x.P'}\,\text{REST}_t \text{ if } x \notin n(\alpha) \qquad \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}\,\text{ID}_t \text{ if } A(\tilde{x}) \overset{def}{=} P$$

$$\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}\,\text{MTCH} \qquad \frac{P \xrightarrow{\alpha} P'}{[x \neq y]P \xrightarrow{\alpha} P'}\,\text{MISMTCH if } x \neq y$$

$$\frac{P \xrightarrow{\alpha} Q}{P' \xrightarrow{\alpha} Q'}\,\text{CONG}_t \quad \text{if } P \equiv P' \text{ and } Q \equiv Q'$$

Table 3.5: π-calculus transition rules

Definition 3.12. *The transition relation $\xrightarrow{\alpha} \subseteq \mathcal{P}_\pi \times \mathcal{A}_\pi \times \mathcal{P}_\pi$ over agents in $\mathcal{P}_\pi$ and actions in $\mathcal{A}_\pi$ is the least relation satisfying the rules in table 3.5*

With these rules, the role of the τ action becomes apparent. As said before, it represents a silent action. By this we mean that it is the action that corresponds to internal communication. As shown in the rule COMM_t, the effect of interaction is the τ action.

This definition represents the evaluation scheme known as *late instantiation* since the substitution is performed at the time where an internal communication takes place. An alternative definition is *early instantiation* in which variables are instantiated when the input transition is inferred. The scheme results by replacing PREF_t with TAU_t, OUT_t and INP_t, and COMM_t with COMM_t^e, defined as follows:

$$\frac{}{\tau.P \xrightarrow{\tau} P}\,\text{TAU}_t$$

$$\frac{}{\overline{u}\langle x\rangle.P \xrightarrow{\overline{u}\langle x\rangle} P}\,\text{OUT}_t \qquad \frac{}{u(x).P \xrightarrow{u(y)} P\{y/x\}}\,\text{INP}_t \text{ if } y \notin fn(\nu x.P)$$

$$\frac{P \xrightarrow{u(x)} P' \quad Q \xrightarrow{\overline{u}\langle x\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\,\text{COMM}_t^e$$

Note that the input rule has the side condition "$y \notin fn(\nu x.P)$", which means that y is a fresh name, so it won't produce any conflicts as to safely replace x. This definition is not very symmetric, but we can recover the symmetry by an alternative presentation which can be achieved just by changing COMM_t to COMM_t^e as above, and keeping PREF. To see how this is so, notice that both TAU_t and OUT_t are instances of PREF, and we can recover INP_t by means of alpha conversion as follows:

$$\dfrac{\dfrac{}{u(y).P\{y/x\} \xrightarrow{u(y)} P\{y/x\}}\ \mathrm{PREF} \qquad \dfrac{}{u(x).P \equiv u(y).P\{y/x\}}\ \alpha - \mathrm{conv\ if}\ y \notin fn(\nu x.P)}{u(x).P \xrightarrow{u(y)} P\{y/x\}}\ \mathrm{CONGR}$$

From an informal perspective, the late-instantiation scheme seems more natural, since it matches our intuition that substitution must be performed at the time of communication rather that at the time of applying the INP_t rule (or PREF_t in the simplified presentation). This early-instantiation scheme is akin to considering an isolated agent as capable of somehow guessing the "right" name to substitute in advance of the actual communication. However, from a formal perspective, both presentations allow us to prove the same reductions, in the sense of the UTS.

The following property is very useful, and can be proved by induction on the derivation of $P \xrightarrow{\alpha} P'$:

Lemma 3.13. *If $P \xrightarrow{\alpha} P'$, then for any substitution σ, $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.*

3.1.6 Equivalence of semantics

An important question is whether these two different presentations of the semantics (LTS and UTS) are actually equivalent or not. The intuition is that they should be, but this is a fact that has to be proven formally. However the concept of an action performed by an individual agent is not present in the UTS. This implies that the agreement must occur at some higher level of abstraction. As mentioned before, the silent action represents internal activity of an agent, and thus the agent is viewed as a whole. This is the same approach of the UTS, so the agreement is at this level: reductions correspond to silent actions and vice-versa. Here we provide the proof for the late instantiation scheme [1]. These have been established by Milner [29].

Lemma 3.14. *If $P \rightarrow P'$ then $P \xrightarrow{\tau} P'$.*

Proof. This is proved by induction on the inference of $P \rightarrow P'$. This is, for the last step of the derivation of $P \rightarrow P'$ we construct a proof (derivation tree) of $P \xrightarrow{\tau} P'$.

Case 1: The last inference is an instance of the TAU axiom, i.e. it was $\tau.P \rightarrow P$. This is matched in the LTS by the TAU_t axiom.

[1] The proof for early instantiation is very close to the given one.

Case 2: The last inference is an instance of the COMM axiom. $P \equiv (u(x).P_1 + Q_1) \mid (\overline{u}\langle y\rangle.P_2 + Q_2)$ and $P' \equiv P_1\{y/x\} \mid P_2$. This is matched in the LTS by constructing the following proof:

$$\dfrac{\dfrac{\dfrac{}{u(x).P_1 \xrightarrow{u(x)} P_1}\,\mathrm{PREF}_t}{u(x).P_1 + Q_1 \xrightarrow{u(x)} P_1}\,\mathrm{SUM}_t \qquad \dfrac{\dfrac{}{\overline{u}\langle y\rangle.P_2 \xrightarrow{\overline{u}\langle y\rangle} P_2}\,\mathrm{PREF}_t}{\overline{u}\langle y\rangle.P_2 + Q_2 \xrightarrow{\overline{u}\langle y\rangle} P_2}\,\mathrm{SUM}_t}{(u(x).P_1 + Q_1) \mid (\overline{u}\langle y\rangle.P_2 + Q_2) \xrightarrow{\tau} P_1\{y/x\} \mid P_2}\,\mathrm{COMM}_t$$

Case 3: The last inference was an application of the PAR rule, so $P \equiv Q \mid R$ and $P' \equiv Q' \mid R$, with $Q \rightarrow Q'$ by a shorter inference. Hence, by induction hypothesis we have that $Q \xrightarrow{\tau} Q'$. Now we easily construct the matching LTS proof:

$$\dfrac{Q \xrightarrow{\tau} Q'}{Q \mid R \xrightarrow{\tau} Q' \mid R}\,\mathrm{PAR}_t$$

The remaining cases, for SUM, RESTR, ID, and CONGR mimic the last case. □

To prove the converse we need an additional technical lemma. This represents the idea that when a process performs some action α, it is because the agent contained a subexpression of the form $\alpha.Q$.

Lemma 3.15. *If $P \xrightarrow{\alpha} P'$ where $\alpha \neq \tau$ then there are $\vec{x}, Q, Q'$, and R such that $P \equiv \nu\vec{x}.((\alpha.Q + Q') \mid R)$ and $P' \equiv \nu\vec{x}.(Q \mid R)$ where $n(\alpha) \cap \vec{x} = \emptyset$.*

Proof. By induction on the derivation of $P \xrightarrow{\alpha} P'$. The only important thing to notice is that we are assuming that the action is not τ, hence it is not an internal communication, so we can ignore the case of the rule COMM_t. □

Theorem 3.16. *$P \rightarrow P'$ if and only if $P \xrightarrow{\tau} P'$.*

Proof. The left-to-right direction has been established by lemma 3.14. For the right to left direction we do it by induction on the inference of $P \xrightarrow{\tau} P'$.

Case 1: The last inference is an instance of TAU_t. Hence $P \equiv \tau.P' \rightarrow P'$.

Case 2: The last inference is an instance of COMM_t. Hence $P \equiv Q \mid R$, and $P' \equiv Q' \mid R'\{x/y\}$ with $Q \xrightarrow{\overline{u}\langle x\rangle} Q'$ and $R \xrightarrow{u(y)} R'$. Then by lemma 3.15 we have:

$$\begin{array}{lcl} Q \equiv \nu\vec{w}_1.((\overline{u}\langle x\rangle.Q_1 + Q_2) \mid Q_3) & \text{and} & Q' \equiv \nu\vec{w}_1.(Q_1 \mid Q_3) \\ R \equiv \nu\vec{w}_2.((u(y).R_1 + R_2) \mid R_3) & \text{and} & R' \equiv \nu\vec{w}_2.(R_1 \mid R_3) \end{array}$$

where $\{u, x, y\} \cap \vec{w}_1, \vec{w}_2 = \emptyset$. Hence we can build the proof for $P \rightarrow P'$ by using CONGR.

The rest of the cases are analogous to the proof of lemma 3.14 $\square$

This result points out that reductions correspond to τ transitions. This is, the semantics are equivalent at the level of abstraction of internal communication. The UTS does not model the (potential) behaviour of agents individually. It rather represents the view of the entire system. The LTS on the other hand gives us an explicit handle of the capabilities of the agents, viewed isolated from the rest of the system.

3.2 Bisimilarity for π-processes

3.2.1 Strong bisimilarity: ground, late, early and open variants

At first it looks like the definitions of simulation, similarity, bisimulation and bisimilarity given in section 2.3 could be directly used for our notion of behavioral equivalence for π processes. However we have to adapt this notion to the new setting, in particular we need to take care of bound objects and input actions. Consider the terms $A \stackrel{def}{=} u(x).D$ and $B \stackrel{def}{=} \nu b.u(y).D$. Certainly we would like to consider these two as equivalent when they are both receiving a name p different from b, because they would react in the same way with an action $\alpha = u(p)$. Under the traditional notion of bisimilarity we would not have this.

Definition 3.17 (Strong ground bisimilarity). *A binary relation $\mathcal{S} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is called a **ground-simulation** iff for any terms $P, Q \in \mathcal{P}_\pi$, $P\mathcal{S}Q$ implies that*

- *Whenever $P \stackrel{\alpha}{\to} P'$ and $bn(\alpha) \cap fn(Q) = \emptyset$, there is a Q' such that $Q \stackrel{\alpha}{\to} Q'$ and $P'\mathcal{S}Q'$.*

*If $\mathcal{S}^{-1}$ is also a ground-simulation then $\mathcal{S}$ is called a **ground-bisimulation**.*

*We say that Q **simulates** P, or that P and Q are **ground similar**, written $P \dot{\prec}^g Q$, iff there is a ground-simulation $\mathcal{S}$ such that $P\mathcal{S}Q$.*

*We say that P and Q are **ground bisimilar**, written $P \dot{\sim}^g Q$, iff there is a ground-bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$.*

This definition, however does not capture the intended notion of equivalence. Ground bisimilarity, turns out not to be a process congruence. The problem is that it does not preserve parallel composition. To see this, consider the agents $A \stackrel{def}{=} u(x).[x = a]\overline{a}\langle a\rangle$, and $B \stackrel{def}{=} u(x).0$. It is easy to see that A and B are bisimilar, since both of them can perform an action $u(x)$ and the resulting terms $[x = a]\overline{a}\langle a\rangle$ and 0 are bisimilar, since $x \neq a$ (i.e. the two names are different). However, when we put $\overline{u}\langle a\rangle$ in parallel with A and B, we obtain different results: $\overline{u}\langle a\rangle \mid A \stackrel{\tau}{\to} [a = a]\overline{a}\langle a\rangle \stackrel{\overline{a}\langle a\rangle}{\to} 0$, but $\overline{u}\langle a\rangle \mid B \stackrel{\tau}{\to} 0 \not\to$, so the second transition cannot be matched. Hence $\overline{u}\langle a\rangle \mid A \not\dot{\sim} \overline{u}\langle a\rangle \mid B$.

The problem seems to be rooted at the kind of instantiations or substitutions are possible, when an input action is performed. This suggests the following redefinition.

Definition 3.18 (Strong late bisimilarity). *A relation $\mathcal{S} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is called a* ***strong late simulation*** *iff $P\mathcal{S}Q$ implies:*

(i) Whenever $P \xrightarrow{\alpha} P'$ where α is not an input action, then for some Q', $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.

(ii) Whenever $P \xrightarrow{u(x)} P'$ and $x \notin n(P) \cup n(Q)$ then there is a Q' such that $Q \xrightarrow{u(x)} Q'$ and for all names b, $P'\{b/x\}\ \mathcal{S}\ Q'\{b/x\}$.

If $\mathcal{S}^{-1}$ is also a strong late simulation, then $\mathcal{S}$ is called a ***strong late bisimulation****. We say that $P \dot{\prec}^l Q$ if there is a strong late simulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\prec}^l$ is called* ***strong late similarity****. We say that $P \dot{\sim}^l Q$ if there is a strong late bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\sim}^l$ is called* ***strong late bisimilarity****.*

The basic properties of strong late (bi)simulation are summarized in the following proposition ([30]).

Proposition 3.19.

(i) $\equiv \subseteq \dot{\sim}^l$

(ii) $\dot{\sim}^l$ is an equivalence relation.

(iii) $\dot{\prec}^l$ is a strong late simulation.

(iv) $\dot{\prec}^l$ is the largest strong late simulation.

(v) $\dot{\sim}^l$ is a strong late bisimulation.

(vi) $\dot{\sim}^l$ is the largest strong late bisimulation.

Proof. The first item just follows from the definition of the CONGR rule. The rest of the proof follows the same lines as in propositions 2.3 and 2.5. □

Another variant of this concept is known as *early*-(bi)simulation, which results from replacing the second condition in the definition, with:

(ii)' Whenever $P \xrightarrow{u(x)} P'$ and $x \notin n(P) \cup n(Q)$ then for all names b, there is a Q' such that $Q \xrightarrow{u(x)} Q'$ and $P'\{b/x\}\ \mathcal{S}\ Q'\{b/x\}$.

This seems like a harmless modification, but it results in a weaker equivalence in the sense that more terms are grouped in the same equivalence classes. As Milner, Parrow and Walker explain in [30], it is because in late bisimilarity we require that the matching transition works for all possible instantiations of the object received by the input action, whereas in early bisimilarity for each possible instantiation there must be a matching transition, but for different instantiations the matching transition can be different. This way, we can think of the early variant as considering that the input action happens before the transition, while in the late variant it happens after the transition. Hence the immediate consequence is that all processes that are late-bisimilar are also early-bisimilar, and there are processes that are early-bisimilar but not late-bisimilar ([30]).

Corollary 3.20. $\dot{\sim}^l \subset \dot{\sim}^e$

Proof. First we prove that $\dot{\sim}^l \subseteq \dot{\sim}^e$. By definition, if P and Q are late bisimilar, then each transition that P makes is matched by Q, resulting in states P' and Q' respectively, such that for each possible instantiation, P' and Q' are themselves late bisimilar, and vice-versa. This implies that for any instantiation, Q can match P, and vice-versa. But this is precisely the definition of early bisimilarity.

Now we show that the two bisimilarities disagree. Consider the following terms: $P \stackrel{def}{=} u(x).0 + u(x).R$ and $Q \stackrel{def}{=} P + u(x).[x = y]R$. We have that $P \dot{\sim}^e Q$, because for each possible instantiation, both P and Q match each other transitions. It is clear that Q can always match P's moves. To see how P matches Q's moves, we consider each possible instantiation b of x. If $b = y$ and if $Q \xrightarrow{u(x)} [x = y]R$ then $P \xrightarrow{u(x)} R$ and $([x = y]R)\{y/x\} \dot{\sim}^e R\{y/x\}$. If b is any other name, then $P \xrightarrow{u(x)} 0$ and $([x = y]R)\{b/x\} \dot{\sim}^e 0 \equiv 0\{b/x\}$.

On the other hand, $P \dot{\not\sim}^l Q$ because P cannot match the transition $Q \xrightarrow{u(x)} [x = y]R$. ☐

The properties described in proposition 3.19 also hold for the early variant.

A fundamental question remains: are any of these notions the "right" notion of behavioural equivalence that we are after? In other words are they process congruences? The answer is no. These notions, like ground bisimilarity, also fail to preserve all operators, but now the input prefix is the problematic operator. To see this consider the agents $P \stackrel{def}{=} \overline{x} \mid y$ and $Q \stackrel{def}{=} \overline{x}.y + y.\overline{x}$. We have that $P \dot{\sim} Q$ because there is a bisimulation that contains (P, Q), namely $\mathcal{S} = \{(P, Q), (\overline{x}, \overline{x}), (y, y), (0, 0)\}$. This captures the idea that concurrent processes can be reordered in time as we please. However there is a context in which they don't behave in the same way: $\mathcal{C}[\cdot] \stackrel{def}{=} \overline{u}\langle x\rangle \mid u(y).[\cdot]$. In this context we have that

$$\mathcal{C}[P] = \overline{u}\langle x\rangle \mid u(y).P \xrightarrow{\tau} P\{x/y\} = \overline{x} \mid x \xrightarrow{\tau} 0$$

while

$$\mathcal{C}[Q] = \overline{u}\langle x\rangle \mid u(y).Q \xrightarrow{\tau} Q\{x/y\} = \overline{x}.x + x.\overline{x}$$

The result of $\mathcal{C}[Q]$ does not have a τ action to match the second one of $\mathcal{C}[P]$, thus $\mathcal{C}[P] \dot{\not\sim} \mathcal{C}[Q]$. The problem, as for ground bisimilarity, is that there is a substitution

$\sigma = \{x/y\}$ such that $P\sigma \dot{\not\sim} Q\sigma$, even if it was the case that $P \dot{\sim} Q$. Semantically, substitution is linked to input, and the following lemma pinpoints the problem.

Lemma 3.21. *For any processes P and Q, and any names x, y, and u,*

- $P\{x/y\} \dot{\sim}^l Q\{x/y\}$ *if and only if* $u(y).P \dot{\sim}^l u(y).Q$*, and also*
- $P\{x/y\} \dot{\sim}^e Q\{x/y\}$ *if and only if* $u(y).P \dot{\sim}^e u(y).Q$

Proof. Follows directly from the definitions of bisimilarity and the LTS. We prove only the left-to-right direction of the late case, to show how these proofs are done in general. The rest is similar. It suffices to find a bisimulation that contains all pairs $(u(y).P, u(y).Q)$ given that for all names x, $P\{x/y\} \dot{\sim}^l Q\{x/y\}$. Let $\mathcal{S} \stackrel{def}{=} \{(u(y).P, u(y).Q) | \forall x, P\{x/y\} \dot{\sim}^l Q\{x/y\}\} \cup \dot{\sim}^l$. We show that $\mathcal{S}$ is such a bisimulation. Assume that $(u(y).P, u(y).Q) \in \mathcal{S}$. The agent $u(y).P$ can make a move $u(y).P \xrightarrow{u(x)} P\{x/y\}$ for any x. The other agent can match this move: $u(y).Q \xrightarrow{u(x)} Q\{x/y\}$, and by our hypothesis $P\{x/y\} \dot{\sim}^l Q\{x/y\}$. Thus $\mathcal{S}$ is a simulation and by a dual argument we determine that it is a bisimulation. □

From this lemma it follows that since substitution does not preserve bisimilarity as the counterexample showed, then it does not preserve input prefixes and hence it is not a process congruence.

In order to cope with this problem we have to refine the notion of simulation. One approach, which can be applied to many different notions of bisimilarity, is to "force" it to be a congruence. This is, we can refine bisimilarity simply by requiring that it behaves well under substitutions. In this sense we say that bisimilarity *induces* a corresponding congruence.

Definition 3.22. *A relation $\mathcal{S} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is* ***closed under substitution*** *if for every substitution σ, $P\mathcal{S}Q$ implies $P\sigma\mathcal{S}Q\sigma$.*

In the following definition, x stands for g (ground), l (late), e (early), or any other notion of bisimilarity that we define.

Definition 3.23 (Bisimilarity congruence). *Let $\dot{\sim}^x$ be a bisimilarity relation, such as ground, late, early, etc. We say that P and Q are (strongly) x-congruent, written $P \sim^x Q$ if $\dot{\sim}^x$ is closed under substitution.*

We now check formally that such a relation is indeed a process congruence, thus justifying the name. This is done very much along the lines of lemma 3.21.

Lemma 3.24. *$\sim^x$ is a process congruence, where x stands for l (late) or e (early).*

Proof. We prove that $\sim^l$ is preserved by all operators. Assume $P_1 \sim^l P_2$.

Input prefix: By definition, $\sim^l$ is preserved by arbitrary substitutions, so for any x, y, $P_1\{y/x\} \sim^l P_2\{y/x\}$, and by lemma 3.21, $u(x).P_1 \sim^l u(x).P_2$ as required.

Output prefix and tau: Let α be either an output action, or a τ action. Define $\mathcal{S} \stackrel{def}{=} \{(\alpha.P_1, \alpha.P_2) : P_1 \sim^l P_2\}$. We show that $\mathcal{S}$ is a (ground) bisimulation. Let $(\alpha.P_1)\mathcal{S}(\alpha.P_2)$, and $\alpha.P_1 \xrightarrow{\alpha} P_1$. This move is matched by $\alpha.P_2 \xrightarrow{\alpha} P_2$, and since $P_1 \sim^l P_2$ then $\mathcal{S}$ is a simulation. By a dual argument we have that it is a bisimulation. Hence $\alpha.P_1 \sim^l \alpha.P_2$.

Parallel composition: Let $\mathcal{S} \stackrel{def}{=} \{(P_1 \mid R, P_2 \mid R) : P_1 \sim^l P_2\}$. We show that $\mathcal{S}$ is a (ground) bisimulation. Take $(P_1 \mid R)\mathcal{S}(P_2 \mid R)$. Suppose that $P \mid R \xrightarrow{\alpha} M$. There are three possible cases depending on where the action originated:

1. It originated from R, i.e. $M \equiv P_1' \mid R'$ with $R \xrightarrow{\alpha} R'$ and $P_1' \equiv P_1$. Then by PAR_t, $P_2 \mid R \xrightarrow{\alpha} P_2 \mid R'$, and since $P_1 \sim^l P_2$ we have that $(P_1 \mid R')\mathcal{S}(P_2 \mid R')$ as required.
2. It originated from P_1, i.e. $M \equiv P_1' \mid R'$ with $P_1 \xrightarrow{\alpha} P_1'$ and $R' \equiv R$. Then, since $P_1 \sim^l P_2$, $P_2 \xrightarrow{\alpha} P_2'$ with $P_1' \sim^l P_2'$. So by PAR_t, $P_2 \mid R \xrightarrow{\alpha} P_2' \mid R$. Hence $(P_1' \mid R)\mathcal{S}(P_2' \mid R)$ as required.
3. It originated from an interaction between P_1 and R, so $\alpha = \tau$, i.e. $M \equiv P_1'\{y/x\} \mid R'$ with $P_1 \xrightarrow{u(x)} P_1'$ and $R \xrightarrow{\overline{u}\langle y\rangle} R'$. Since $P_1 \sim^l P_2$, we have that $P_2 \xrightarrow{u(x)} P_2'$ with $P_1' \sim^l P_2'$. By applying COMM_t, we obtain that $P_2 \mid R \xrightarrow{\tau} P_2'\{y/x\} \mid R'$, and since $\sim^l$ is closed under substitution, $P_1'\{y/x\} \sim^l P_2'\{y/x\}$. Therefore $M\mathcal{S}P_2'\{y/x\} \mid R'$ as required.

The rest of the cases are similar. □

The good thing about this definition of bisimilarity-congruence is that we obtain a process congruence out of a bisimilarity relation, by forcing it. This, however, has its down-side in that proving equivalence between terms becomes more difficult in the sense that such proofs require a heavier case analysis.

Notice that a bisimilarity congruence is a finer relation (more discriminating) than the corresponding "naked" bisimilarity, i.e. the bisimilarity relation not closed under substitution. If two processes are bisimilar-congruent, then they certainly are bisimilar, but, as the previous examples showed, the converse is not true.

Lemma 3.25. *$\sim^x \subset \dot{\sim}^x$, where x stands for l (late) or e (early).*

An alternative to taking the closure under substitution of bisimilarity to obtain a process congruence, has been proposed by Sangiorgi in [40], called *open*-bisimilarity.

Definition 3.26 (Open bisimilarity). *A relation $\mathcal{S} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is called a **strong open simulation** iff $P\mathcal{S}Q$ implies, for every substitution σ:*

- *Whenever $P\sigma \xrightarrow{\alpha} P'$ then for some Q', $Q\sigma \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.*

If $\mathcal{S}^{-1}$ is also a strong open simulation, then $\mathcal{S}$ is called a ***strong open bisimulation****. We say that $P \dot{\prec}^o Q$ if there is a strong open simulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\prec}^o$ is called* ***strong open similarity****. We say that $P \dot{\sim}^o Q$ if there is a strong open bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\sim}^o$ is called* ***strong open bisimilarity****.*

Proposition 3.27 (Sangiorgi [40]).

(i) $\dot{\sim}^o$ *is an equivalence relation.*

(ii) $\dot{\sim}^o$ *is the largest open bisimulation.*

(iii) $\dot{\sim}^o = \sim^o$*, i.e.* $\dot{\sim}^o$ *is a process congruence.*

Proof. The first two items are proved in a very similar fashion to the corresponding proofs for ground bisimilarity. The interesting one is the third. It is enough to prove that $\dot{\sim}^o$ is closed under substitution, so by lemma 3.24 we obtain that it is a congruence. We can see that it is closed under substitution from its definition. We can show this explicitly as follows. Suppose that $P \dot{\sim}^o Q$. Consider any substitution σ, and the set $\mathcal{S} \stackrel{def}{=} \{(P\sigma, Q\sigma) : P \dot{\sim}^o Q\}$. We show that this is an open bisimulation. Let $(P\sigma, Q\sigma) \in \mathcal{S}$ and $P\sigma \xrightarrow{\alpha} P'$. Then, since $P \dot{\sim}^o Q$, we have that $Q\sigma \xrightarrow{\alpha} Q'$ and $P' \dot{\sim}^o Q'$. Hence, $(P'\sigma, Q'\sigma) \in \mathcal{S}$. Thus it is an open bisimulation, so $P\sigma \dot{\sim}^o Q\sigma$ as required. □

The previous theorem justifies writing open bisimilarity using the notation $\sim^o$, without the dot, to emphasise that it is a congruence.

Open bisimilarity, is a finer behavioural equivalence than late and early congruences.

Lemma 3.28 (Sangiorgi [40]). $\sim^o \subset \sim^l \subset \sim^e$

3.2.2 Weak bisimilarity

One of the problems of the definitions above is that they are too strong. The requirement that we must match all τ actions in the same way as we match input and output actions is too stringent. Since the goal is to equate agents that interact in the same way with the environment, we should be able to ignore their internal actions. In this framework, the silent action τ represents internal activity that does not affect directly the environment. In other words, if an agent A makes one action α, an agent B can be considered as matching that behavior if it performs several τ actions before and/or after actually doing the action α, since the environment only observes the α actions and not the silent actions. Thus we must relax our notion of behavioral equivalence. This is done by introducing a "weak" version of simulation ([25], [30], [29]).

First some notation: we write $P \stackrel{\alpha}{\Rightarrow} Q$ for $P(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* Q$, i.e. either $P \xrightarrow{\alpha} Q$ directly, or there are $P_1, P_2, \cdots, P_n$ such that $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_i \xrightarrow{\alpha} P_{i+1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_n \xrightarrow{\tau} Q$.

Definition 3.29. *A relation $\mathcal{S} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is called a* ***weak late simulation*** *iff $P\mathcal{S}Q$ implies:*

(i) Whenever $P \xrightarrow{\alpha} P'$ where α is not an input action, then for some Q', $Q \xRightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.

(ii) Whenever $P \xrightarrow{u(x)} P'$ then there is a Q' such that $Q \xRightarrow{u(x)} Q'$ and for all names b, $P'\{b/x\}\ \mathcal{S}\ Q'\{b/x\}$.

If $\mathcal{S}^{-1}$ is also a weak late simulation, then $\mathcal{S}$ is called a ***weak late bisimulation****. The largest weak late simulation, written $\dot{\precsim}^l$ is called* ***weak late similarity****, and the largest weak late bisimulation written $\dot{\approx}^l$ is called* ***weak late bisimilarity****.*

As done previously, we can define the corresponding "weak early" and "weak open" variants by replacing the strong transitions with weak transitions.

Notice that $P \xRightarrow{\tau} Q$ is equivalent to $P(\xrightarrow{\tau})^*Q$ which includes the possibility of no transition at all, i.e. $Q \equiv P$. This means that τ actions can be matched by "not moving". We now summarize the properties of this equivalence.

Proposition 3.30.

(i) A strong late simulation is also a weak late simulation.

(ii) For any agents P, Q, if $P\dot{\prec}^l Q$ then $P\dot{\precsim}^l Q$.

(iii) For any agents P, Q, if $P\dot{\sim}^l Q$ then $P\dot{\approx}^l Q$.

(iv) $\dot{\approx}^l$ is an equivalence relation.

(v) $\dot{\precsim}^l$ is a weak late simulation.

(vi) $\dot{\precsim}^l$ is the largest weak late simulation.

(vii) $\dot{\approx}^l$ is a weak late bisimulation.

(viii) $\dot{\approx}^l$ is the largest weak late bisimulation.

(ix) For any process P, $P\dot{\approx}^l \tau.P$.

We omit the proofs as they follow directly from the definitions and are analogous to the corresponding for strong bisimilarity. The corresponding properties also hold for the weak early variants.

3.2.3 Barbed bisimilarity

As pointed out earlier, the LTS gives us a handle on the potential interactions of an agent isolated from the rest of the system, and as such, we were allowed to define the notions of bisimulation and bisimilarity. It is however possible to define such notions in the context of a UTS, provided that we can define how can an agent "match" another's behaviour without making explicit use of action labels. The answer to this is that we could consider the processes equivalent provided that they agree on the "observations" that the environment can make from them. We therefore need to establish clearly what are these observations of interest. Several ideas have been proposed ([26], [7]). The definition provided here, captures the idea that a process is observable at the ports in which it is ready to interact with the external world.

Definition 3.31. *We say that an agent P is **observable** at a name x, written $P \downarrow x$ iff there is an unguarded and unrestricted action α in P with x as subject. In other words, the predicate $\downarrow \subseteq \mathcal{P}_\pi \times \mathcal{N}_\pi$ is defined inductively as follows:*

(i) $\overline{x}\langle y \rangle.P \downarrow x$

(ii) $x(y).P \downarrow x$

(iii) $\nu y.P \downarrow x$ *if* $x \neq y$ *and* $P \downarrow x$

(iv) $(P \mid Q) \downarrow x$ *if* $P \downarrow x$ *or* $Q \downarrow x$

(v) $(P + Q) \downarrow x$ *if* $P \downarrow x$ *or* $Q \downarrow x$

*We denote $\Downarrow$ for $\rightarrow^*\downarrow$, i.e. $P \Downarrow x$ if there is a P' such that $P \rightarrow^* P'$ and $P' \downarrow x$. We say that P **immediately converges**, written $P\downarrow$ if there is an x such that $P \downarrow x$. Analogously, we say that P **converges**, written $P \Downarrow$ if there is an x such that $P \Downarrow x$.*

Based on this notion of observability as capability of immediate interaction, we can conceive a notion of bisimilarity in which agents with the same observables are considered equivalent. This definition was developed by Sangiorgi and Milner ([31], [26]).

Definition 3.32. *A relation $\mathcal{S} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi$ is called a **strong barbed simulation** iff $P\mathcal{S}Q$ implies:*

(i) *Whenever $P \rightarrow P'$ then for some Q', $Q \rightarrow Q'$ and $P'\mathcal{S}Q'$.*

(ii) *For any name x, whenever $P \downarrow x$ then $Q \downarrow x$.*

*If $\mathcal{S}^{-1}$ is also a strong barbed simulation, then $\mathcal{S}$ is called a **strong barbed bisimulation**. We say that $P \dot{\prec}^b Q$ if there is a strong barbed simulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\prec}^b$ is called **strong barbed similarity**. We say that $P \dot{\sim}^b Q$ if there is a strong barbed bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\sim}^b$ is called **strong barbed bisimilarity**.*

Note that even though $P \mid Q$ and $P + Q$ observe the same variables, they are not barbed bisimilar, because they might not necessarily match each other's reductions. Consider for instance $R_1 \stackrel{def}{=} a(x).P \mid \overline{a}\langle y\rangle.Q$ and $R_2 \stackrel{def}{=} a(x).P + \overline{a}\langle y\rangle.Q$. Certainly we have $R_1 \downarrow a$ and $R_2 \downarrow a$, but we have that $R_1 \rightarrow P\{y/x\} \mid Q$, while R_2 cannot match this move unless it is in a special context.

In [30] and [31] the following result was proven, showing that the process congruence induced by this concept ($\sim^b$) agrees with the process congruence induced by strong early congruence ($\sim^e$).

Theorem 3.33 (Milner, Sangiorgi [31]). $\sim^b = \sim^e$

There is also a weak variant of barbed simulation. We replace the conditions with the following:

(i) Whenever $P \rightarrow P'$ then for some Q', $Q \rightarrow^* Q'$ and $P'\mathcal{S}Q'$.

(ii) For any name x, whenever $P \downarrow x$ then there is a Q' such that $Q \rightarrow^* Q'$ and $Q' \downarrow x$. (Abbreviated $Q \rightarrow^* \downarrow x$)

Where $\rightarrow^*$ is as usual the transitive closure of $\rightarrow$. For the weak variants of similarity and bisimilarity we use the notation $\dot{\precsim}^b$ and $\dot{\approx}^b$.

3.3 Expressiveness: Encoding the λ-calculus in π

As our first concrete study of expressiveness in process calculi, we present here the encoding, given by Milner ([26], [27], [29]), of the *lazy λ-calculus* into the π-calculus. For this, we briefly recall the notions of the λ-calculus.[2]

The λ-calculus is the core of functional programming languages. It views computation in terms of functions and the main operation is function application (evaluating a function providing it with some arguments as to produce some output). This is in contrast with the π calculus, which focuses on *processes* and the core of computation is interaction. In the lambda calculus, every computable function is expressible. This implies that if a programming language is to have full computational power (from this functional point of view) it must be able to simulate the λ-calculus.

3.3.1 The lazy λ-calculus

We assume a set of names $\mathcal{N}_\lambda$, with x, y, z, ... ranging over this set. The set of lambda-terms $\mathcal{P}_\lambda$, ranged over by M, N, M', N_1, N_2, etc., is inductively defined by:

$$M \quad ::= \quad x \quad \mid \quad \lambda x.M_0 \quad \mid \quad M_1 M_2$$

[2]Here we present a rather succinct description of the λ-calculus, in particular, we omit the definition of substitution for lambda-terms. The reader is referred to the vast literature on λ-calculus for this, see for instance [4], [38].

The semantics for the (lazy)λ-calculus is given by an UTS as follows:

$$\frac{}{\lambda x.M \equiv_\alpha \lambda y.M\{y/x\}}\ \alpha - \text{CONV} \qquad \frac{}{(\lambda x.M)N \to M\{N/x\}}\ \beta - \text{RED}$$

$$\frac{M \to M'}{MN \to M'N}\ \nu - \text{APP} \qquad \frac{M \equiv_\alpha M' \quad M \to N \quad N \equiv_\alpha N'}{M' \to N'}\ \text{CONGR}$$

The behavioural equivalence for lambda terms, denoted $=_\lambda$ is the smallest equivalence relation between terms, that includes reduction, i.e. if one term reduces to another, then they are behaviourally equivalent; in particular $(\lambda x.M)N =_\lambda M\{N/x\}$.

3.3.2 The translation

The essence of this translation is to express the higher-order substitution operation found in the λ-calculus (i.e. substitution of arbitrary terms for names) in terms of the simpler first-order substitution of the π-calculus. This is done in a fashion that resembles the implementation in real programming languages of functional application and procedure calls. That is, application is not performed by physically replacing the name by the actual argument, but by executing the body of the function in an environment in which the formal argument is bound to the actual argument. This binding is created at the time of the function call.

Formally, the translation is given by a function that associates each lambda-term M together with a name u to a π-agent that expects its arguments through the port u.

Definition 3.34 (λ to π). *The translation* $[\![\cdot]\!] : \mathcal{P}_\lambda \to (\mathcal{N}_\pi \to \mathcal{P}_\pi)$ *is given by the following equations:*

$$\begin{aligned}
[\![x]\!]u &\stackrel{def}{=} \overline{x}\langle u\rangle \\
[\![\lambda x.M]\!]u &\stackrel{def}{=} u(xv).[\![M]\!]v && \textit{with } v \textit{ fresh} \\
[\![MN]\!]u &\stackrel{def}{=} \nu v.([\![M]\!]v \mid \nu x.(\overline{v}\langle xu\rangle.\mathsf{cell}(x,N))) && \textit{with } v, x \textit{ fresh}
\end{aligned}$$

where $\mathsf{cell}(x,N) \stackrel{def}{=} !\, x(w).[\![N]\!]w$.

As can be seen, the argument of an application is "stored" in a "cell". This cell is a (replicated) process with port x. The x could be understood as the "address" of the cell. So when a name is accessed it simply sends a message to the cell, informing it where the rest of the arguments (if any) are located. A lambda abstraction, is represented by a process waiting for its argument (and any further arguments) in port u. When the application is performed, we create a process for computing the body of the function called, we create a binding (cell) for its argument, and we establish a link between the body and the cell, i.e. we inform the body what is the address or port where the argument is.

3.3.3 Correctness of the translation

Now we establish that the π-calculus has the full expressive power of the λ-calculus. Recalling that notion, we expect that the translation preserves and reflects some relation between terms, preferably a behavioural equivalence relation. Hence what we want to prove if that for any lambda terms M and N, if $M =_\lambda N$ then $[\![M]\!] \approx_\pi [\![N]\!]$, i.e. $[\![\cdot]\!]$ is complete w.r.t. $=_\lambda, \approx_\pi$. We use weak bisimilarity rather than strong, because one beta reduction is simulated by several interactions in the π-translation. In particular, we want to show that beta-reduction, the computationally meaningful rule on the λ-calculus semantics, is faithfully respected by the translation.

Theorem 3.35 (Correctness of λ to π translation - Milner [27]).

$$[\![(\lambda x.M)N]\!] \approx_\pi [\![M\{N/x\}]\!]$$

Proof. First, we have that

$$\begin{aligned}
[\![(\lambda x.M)N]\!] &\equiv \nu v.(v(xw).[\![M]\!]w \mid \nu z.(\overline{v}\langle zu\rangle.\mathsf{cell}(z,N))) && \text{by definition of } [\![\cdot]\!] \\
&\xrightarrow{\tau} \nu z.(([\![M]\!]\{z/x\})u \mid \mathsf{cell}(z,N)) && \text{by COMM and CONGR}
\end{aligned}$$

Thus $[\![(\lambda x.M)N]\!] \approx_\pi \nu z.(([\![M]\!]\{z/x\})u \mid \mathsf{cell}(z,N))$. Then we only need to show that $\nu z.(([\![M]\!]\{z/x\})u \mid \mathsf{cell}(z,N)) \approx_\pi [\![M\{N/x\}]\!]$. We do this by induction on the structure of M. Notice that the arbitrary term substitution has been replaced by a simpler name substitution of the π-calculus. In the rest of this proof, we assume that $[\![M]\!]\{z/x\} \equiv [\![M]\!]$, to make it more readable.

Case 1 $M \equiv_\alpha z$. Then

$$\begin{aligned}
\nu z.([\![M]\!]u \mid \mathsf{cell}(z,N)) &\equiv \nu z.(\overline{z}\langle u\rangle \mid \mathsf{cell}(z,N)) \\
&\approx_\pi [\![N]\!]u \mid \mathsf{cell}(z,N) \\
&\approx_\pi [\![N]\!]u \\
&\equiv [\![M\{N/z\}]\!]u
\end{aligned}$$

Case 2 $M \equiv_\alpha y \neq z$. Hence we have

$$\nu z.([\![M]\!]u \mid \mathsf{cell}(z,N)) \approx_\pi \overline{y}\langle u\rangle \approx_\pi [\![M\{N/z\}]\!]$$

Case 3 $M \equiv_\alpha \lambda y.M_0$, so $[\![M]\!]u \equiv u(yv).[\![M_0]\!]v$. Hence

$$\begin{aligned}
\nu z.([\![M]\!]u \mid \mathsf{cell}(z,N)) &\approx_\pi u(yv).\nu z.([\![M_0]\!]v \mid \mathsf{cell}(z,N)) \\
&\approx_\pi u(yv).[\![M_0\{N/z\}]\!]v && \text{by induction} \\
&\equiv [\![\lambda y.(M_0\{N/z\})]\!]u \\
&\equiv [\![M\{N/z\}]\!]u
\end{aligned}$$

Case 4 $M \equiv_\alpha M_1 M_2$. This case requires some properties of the replication operator that we have not studied here, so we will provide only an outline, and the reader is referred to [29] for a more detailed account. We have that $[\![M]\!]u \equiv \nu v.([\![M_1]\!]v \mid \nu y.\overline{v}\langle yu\rangle.\mathsf{cell}(y,M_2))$. One of the properties not proved here, but intuitively correct is that the cell $!\ z(y).[\![N]\!]y$ can be distributed over parallel composition (making it available to both sub-terms) as follows:

$$\begin{aligned}\nu z.([\![M]\!]u \mid \mathsf{cell}(z,N)) \approx_\pi \\ &\nu v.(\nu z.([\![M_1]\!]v \mid \mathsf{cell}(z,N)) \mid \\ &\qquad \nu y.(\overline{v}\langle yu\rangle \mid \nu z.(\mathsf{cell}(y,M_2) \mid \mathsf{cell}(z,N))))\end{aligned}$$

We can simplify this by using the following:

$$\begin{aligned}\nu z.(\mathsf{cell}(y,M_2) \mid \mathsf{cell}(z,N)) &\approx_\pi\ !\ y(w).\nu z.([\![M_2]\!]w \mid \mathsf{cell}(z,N)) \\ &\equiv \mathsf{cell}(y,[\![M_2\{N/z\}]\!]) \quad \text{by induction hypothesis and definition of cell}\end{aligned}$$

So putting all this together, we have

$$\begin{aligned}&\nu z.([\![M]\!]u \mid \mathsf{cell}(z,N)) \\ &\approx_\pi \nu v.([\![M_1\{N/z\}]\!]v \mid \nu y.(\overline{v}\langle yu\rangle \mid \mathsf{cell}(y,[\![M_2\{N/z\}]\!]))) \\ &\equiv [\![(M_1 M_2)\{N/z\}]\!]u\end{aligned}$$

□

This theorem states that the translation is complete, because transitions and equivalence are preserved by it. This is enough to assert that the π-calculus is able to simulate any λ-term, and thus, has its full expressive power. This expressiveness represents not only the power to compute as functions, but also that despite being restricted to first-order processes, i.e. messages are just names without any structure, the π-calculus is able to express (a form of) higher-orderness[3]. This suggests an intrinsic expressive power of parallel composition and interaction.

3.4 Summary

In this chapter we described the basic π-calculus, as a language specially geared towards mobility, describing its semantics both in terms of a UTS and two forms of LTS. We showed how these presentations are equivalent at the level of internal communication. We also introduced, as notions required for the UTS and LTS of the π-calculus,

[3] Sangiorgi studied in [41] a higher-order variant of π-calculus, in which messages can be processes, not just names. He proved that allowing such higher-order processes does not increase expressive power, since they can be encoded in the first-order π-calculus.

the ideas of process congruence and structural congruence. Process congruence formalizes the "ideal" equivalence, in the sense that agents that are process-congruent are indistinguishable from the point of view of the environment in which they might live. Then we presented the concept of bisimilarity in the context of the π-calculus, and showed why this introduces several problems, making it hard to reach the goal of being a process congruence. This translates in the formulation of several different notions of bisimilarity.

Research in this field has focused on looking for behavioral process congruences in the sense of definition 3.6. All the definitions of bisimilarity shown here induce a corresponding definition of a behavioral congruence. This is summarized in the following table:

Bisimilarity relation	Induced congruence	Description
$\dot{\sim}^g$	$\sim^g$	Strong ground
$\dot{\sim}^l$	$\sim^l$	Strong late
$\dot{\sim}^e$	$\sim^e$	Strong early
$\dot{\sim}^o$	$\sim^o$	Strong open
$\dot{\sim}^b$	$\sim^b$	Strong barbed
$\dot{\approx}^g$	$\approx^g$	Weak ground
$\dot{\approx}^l$	$\approx^l$	Weak late
$\dot{\approx}^e$	$\approx^e$	Weak early
$\dot{\approx}^o$	$\approx^o$	Weak open
$\dot{\approx}^b$	$\approx^b$	Weak barbed

An alternative to the introduction of new notions of bisimilarity with hopes that they are process congruences is to modify the original calculus. We will describe the main variants of the π-calculus that have been proposed in this respect in the following chapters.

We also showed the expressive power of the π-calculus by giving a complete embedding of the (lazy)lambda calculus into π.

Chapter 4

Asynchronous communication: the π_a-calculus

Much in the same way as the λ-calculus is a canonical calculus for functional computation, the π-calculus aims to be a canonical calculus for concurrency. However, noting the complexities that arise when trying to define a suitable notion of behavioural equivalence it appears that the π-calculus theory does not provide such a canonical basis. CCS, the predecessor of the π-calculus, had a much simpler theory in which strong bisimilarity is a process congruence, but it lacked the ability to describe mobile systems. In order to obtain this holy grail of a behavioral equivalence that is also a congruence for mobile process calculi, two paths have been followed: 1) Refine the notion of (bi)simulation, and 2) Change the language. In section 3.2 some alternative notions of (bi)simulation were presented. The second approach has been followed by simplifying somehow the π-calculus. Following this approach, Honda and Tokoro ([17]), and independently Boudol ([7]) introduced the so-called "asynchronous" π-calculus.

One of the aspects of the π-calculus, as presented in chapter 2, is that communication occurs in a *synchronous* fashion: both the sender of a message and the receiver block until the interaction can occur. Sometimes, however, it might be desirable for the sender to continue processing without waiting for a receiver to take the message. This is *asynchronous* communication.

One way of achieving asynchronous communication, in a synchronous setting, is to use *buffers*. The idea is that the sender of a message does not put the message into a channel directly connected to the receiver, but passes it to a buffer process which holds the message and sends it to its destination when the receiver is able to interact. Since the buffer is modeled by a process in parallel with both the sender and the receiver, the sender can continue processing without waiting for acknowledgement from the receiver, just from the buffer. However the calculus will still allow synchronous communication.

A different approach is to simplify the calculus. The so-called "asynchronous" π calculus, or π_a for short, models asynchronous communication by disallowing a continuation after an output action, i.e. there are no terms of the form $\overline{u}\langle x\rangle.P$ in π_a. Output actions can only occur in parallel with other processes. If we want to model the action of sending a message asynchronously and executing a process P, we simply write the term

$$\overline{u}\langle x\rangle \mid P$$

In such a term, the output action is non-blocking because it can interact with an agent listening through u independently of P, while P can execute without waiting for the interaction to occur.

The set $\mathcal{P}_{\pi_a}$ of π_a terms is defined as the subset of $\mathcal{P}_\pi$ terms where output is always followed by 0. In its most basic form, what we call the *core* π_a, the syntax does not include the summation operator, nor the match/mismatch operators. Alternatively, we forbid explicitly continuations to output actions by defining $\mathcal{P}_{\pi_a}$ as follows:

Definition 4.1 (Core π_a terms). *Let $\mathcal{N}_{\pi_a}$ be an infinite set of names, ranged over by $u, v, w, x, y, \ldots$. As usual, $\tilde{x}$ stands for a sequence of names $x_1 x_2 \ldots x_n$. The set $\mathcal{P}_{\pi_a}$ of π_a terms is defined inductively as:*

$$P \quad ::= \quad 0 \quad | \quad \overline{u}\langle\tilde{x}\rangle \quad | \quad u(\tilde{y}).Q \quad | \quad \nu\tilde{x}.Q \quad | \quad Q \mid R \quad | \quad A(\tilde{x})$$

These constructs have the same interpretation as those of π. Notice that by including procedural definitions and calls ($A(\vec{x})$) and allowing recursion, we can define the replication operator as we did in π: $!\,P \stackrel{def}{=} P \mid\, !\,P$.

The core π_a-calculus is a simplification of the π-calculus in the sense that the π_a calculus is a sub-calculus of π, and thus, everything that can be done in π_a, can be done in π. The question that arises immediately is whether π_a has the same expressive power of π. Honda, Tokoro, and Boudol ([7]), proved that asynchronous interaction is enough to simulate synchronous communication. We present this result in section 4.2.1. Nonetheless, Palamidessi ([36]) proved that the core π_a, deprived of non-deterministic choice, an essential ingredient for concurrency, is not as powerful as the full π-calculus, result which we present in section 4.2.2.

Even with its limitations, π_a is quite an expressive calculus. As an example of its power, let us consider Honda and Yoshida's notions of *forwarder* and *equator* ([18]). A forwarder from a channel a to a channel b, written $a \twoheadrightarrow b$, is a process that acts like a one-cell buffer:

$$a \twoheadrightarrow b \stackrel{def}{=} !\, a(x).\overline{b}\langle x\rangle$$

When a and b are linked through a forwarder process, any message sent through a can be caught by any process listening through b.

An equator between a and b, written $a \twoheadleftarrow\!\!\twoheadrightarrow b$, is a process that makes a and b equivalent in the sense that processes communicating through any of them can interact between them. This can be defined in terms of two forwarder processes as follows:

$$a \twoheadleftarrow\!\!\twoheadrightarrow b \stackrel{def}{=} a \twoheadrightarrow b \mid b \twoheadrightarrow a$$

Since the listening capabilities of two channels are identified by an equator, it achieves the same effects as substitution with respect to the possible patterns of interaction of an agent. It turns out that the following algebraic law holds:

$$P\{a/b\} \approx_{\pi_a} \nu b.(a \twoheadleftarrow\!\!\twoheadrightarrow b \mid P)$$

where $\approx_{\pi_a}$ is (weak)bisimilarity congruence of π_a processes. Notice that this law is only valid in the asynchronous setting, because the forwarder from a to b implies an asynchronous communication between an agent sending a message through a, and another message listening through b.

Another example of its inherent expressiveness is the support of π_a for functional programming. The basic interaction in π_a is given by the reduction

$$u(x).P \mid \overline{u}\langle y\rangle \mid R \rightarrow P\{y/x\} \mid R$$

We can think of the action $\overline{u}\langle y\rangle$ as "calling" a procedure u with actual argument y, where the procedure is given by the receptor $u(x).P$, whose formal argument is x. Although in such an interaction, the procedure is consumed by the reaction, we can simply use the replication operator to make the procedure available for all other processes as in a client-server model.

$$!\, u(x).P \mid \overline{u}\langle y\rangle \mid R \rightarrow !\, u(x).P \mid P\{y/x\} \mid R$$

This idea of conceiving π_a interaction as functional application only covers "first-order" functions because only names can be transmitted through channels. However, we can actually simulate higher-order functions with an encoding of the lambda calculus. This encoding turns out to be almost identical to the one for the full π-calculus, as shown in section 3.3. The only difference is in the translation of functional application, where we replace the blocking output action with the non-blocking version:

$$[\![MN]\!]u \stackrel{def}{=} \nu v.([\![M]\!]v \mid \nu x.(\overline{v}\langle xu\rangle \mid \mathbf{cell}(x, N)))\text{with } v, x \text{ fresh}$$

4.1 Semantics

We now present the formal semantics for π_a, by providing, as we did for the π-calculus, unlabelled and labelled transition systems. We assume the same notion of structural congruence from definition 3.8.

4.1.1 Reductions

The UTS for the core π_a, is a tuple $(\mathcal{P}_{\pi_a}, \equiv, \rightarrow_{\pi_a})$, where $\rightarrow_{\pi_a}$, the reduction relation is the least relation satisfying the rules for π-reduction as in table 3.4, with the exception of the COMM rule, which is replaced by[1]

$$\frac{}{u(x).P \mid \overline{u}\langle y\rangle \to P\{y/x\}}\ \text{COMM}^a$$

In the case of π_a with summation the COMM rule is stated as:

$$\frac{}{(u(x).P\ +\ Q) \mid \overline{u}\langle y\rangle \to P\{y/x\}}\ \text{COMM}^a$$

4.1.2 Transitions

The LTS is defined similarly, as a tuple $(\mathcal{P}_{\pi_a}, \mathcal{A}_{\pi_a}, \equiv, \xrightarrow{\alpha}_{\pi_a})$, where $\mathcal{A}_{\pi_a} \stackrel{def}{=} \mathcal{A}_\pi$, and $\xrightarrow{\alpha}_{\pi_a}$, the transition relation is the least relation satisfying the rules for π-transition as in table 3.5, with the exception of the PREF_t rule, which is replaced, for the late semantics, by

$$\frac{}{\overline{u}\langle x\rangle \xrightarrow{\overline{u}\langle x\rangle} 0}\ \text{OUT}_t^a \quad \text{and} \quad \frac{}{u(x).P \xrightarrow{u(x)} P}\ \text{INP}_t^a$$

The change for the early semantics is analogous to the one for π-calculus.

4.1.3 Bisimilarity

Since π_a is a sub-calculus of π, the notion of open bisimilarity is also a congruence, as well as the early and late congruences. However, in this asynchronous setting, the input actions of an agent cannot be directly observed by an external agent. In order to emphasise this aspect of asynchronous interaction, an alternative to the previous bisimilarities was proposed by Honda and Tokoro ([16]). Here we present one possible characterization for this relation. This was proposed by Amadio, Castellani and Sangiorgi in [3], along with three other characterizations which they proved equivalent to the original proposed by Honda and Tokoro.

We first define a relation called an "$o\tau$"-bisimulation, that represents the insensitivity to input actions, meaning that only output (o) actions and silent actions (τ) are matched by $o\tau$-bisimilar processes.

Definition 4.2 ([3]). *A binary relation $\mathcal{S} \subseteq \mathcal{P}_{\pi_a} \times \mathcal{P}_{\pi_a}$ is called an $o\tau$**-simulation** iff for any terms $P, Q \in \mathcal{P}_{\pi_a}$, $P\mathcal{S}Q$ implies that if $P \xrightarrow{\alpha} P'$ where α is not an input action and $bn(\alpha) \cap fn(Q) = \emptyset$, then $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.*

*If $\mathcal{S}^{-1}$ is also an $o\tau$-simulation, then we call $\mathcal{S}$ an $o\tau$**-bisimulation**.*

We denote $\dot{\sim}^{o\tau}$ the largest $o\tau$-bisimulation, and $\dot{\approx}^{o\tau}$ its weak variant.

Now, to define the appropriate notion of equivalence, we close this relation under composition with arbitrary output actions.

[1] As usual, we will omit the π_a subscript while it is clear which calculus we are discussing.

Definition 4.3 (Asynchronous bisimilarity ([3])). *A relation $\mathcal{S} \subseteq \mathcal{P}_{\pi_a} \times \mathcal{P}_{\pi_a}$ is an* ***asynchronous bisimulation*** *if it is an $o\tau$-bisimulation and for any $P, Q \in \mathcal{P}_{\pi_a}$, $P\mathcal{S}Q$ implies that for any output action $\overline{u}\langle x\rangle$, $(\overline{u}\langle x\rangle \mid P)\mathcal{S}(\overline{u}\langle x\rangle \mid Q)$.*

We denote $\dot{\sim}_{\pi_a}$ the largest asynchronous bisimulation, and $\sim_{\pi_a}$ the induced asynchronous congruence. Also, the corresponding definition of weak asynchronous bisimulation is obtained, as usual, by replacing strong transitions by weak transitions, and we denote $\dot{\approx}_{\pi_a}$ the largest weak asynchronous bisimulation, and $\approx_{\pi_a}$ the induced weak asynchronous congruence.

This is actually an appropriate notion of behavioural equivalence, since it preserves all of π_a's operators.

Proposition 4.4 ([3]).

(i) $\dot{\sim}_{\pi_a}$, and $\dot{\approx}_{\pi_a}$ are equivalence relations.

(ii) $\dot{\sim}_{\pi_a} \; = \sim_{\pi_a}$, and $\dot{\approx}_{\pi_a} \; = \approx_{\pi_a}$; this is, $\dot{\sim}_{\pi_a}$ and $\dot{\approx}_{\pi_a}$ are process congruences.

Proof. Item (i) is easy to check. We will show only transitivity. Suppose that $P\dot{\sim}_{\pi_a}Q$ and $Q\dot{\sim}_{\pi_a}R$. Hence, there are asynchronous bisimulations $\mathcal{S}_1$ and $\mathcal{S}_2$ with $P\mathcal{S}_1Q$ and $Q\mathcal{S}_2R$, which implies $P\mathcal{S}_1\mathcal{S}_2R$. Since $\mathcal{S}_1$ and $\mathcal{S}_2$ are $o\tau$-bisimulations, then $\mathcal{S}_1\mathcal{S}_2$ is also an $o\tau$-bisimulation (using the same argument of transitivity of ground bisimulation). Hence we only need to check the condition of closure under composition with arbitrary outputs. Since $\mathcal{S}_1$ and $\mathcal{S}_2$ are asynchronous bisimulations, we know that for any output $\overline{u}\langle x\rangle$, $(\overline{u}\langle x\rangle \mid P)\mathcal{S}_1(\overline{u}\langle x\rangle \mid Q)$ and $(\overline{u}\langle x\rangle \mid Q)\mathcal{S}_2(\overline{u}\langle x\rangle \mid R)$ respectively. Hence we obtain that $(\overline{u}\langle x\rangle \mid P)\mathcal{S}_1\mathcal{S}_2(\overline{u}\langle x\rangle \mid R)$, so $(\overline{u}\langle x\rangle \mid P) \dot{\sim}_{\pi_a}(\overline{u}\langle x\rangle \mid R)$, as required. For item (ii), it is proven that $\dot{\approx}_{\pi_a}$ is closed under substitution, so that by an argument similar to lemma 3.24, we have that it is a congruence. The proof is rather long, but the technical details are not very difficult. We refer the reader to [3] for a detailed account. ☐

Asynchronous bisimilarity is a coarser equivalence relation than the corresponding notion for the synchronous calculus. Intuitively, given that in the asynchronous calculus we can rearrange output actions in any way, more terms are considered equivalent than in the synchronous calculus.

The notion of barbed congruence is useful in the π_a setting, since it was proven in [3] that it coincides with asynchronous congruence.

Proposition 4.5 (Amadio, Castellani, Sangiorgi [3]). $\sim^b_{\pi_a} = \sim_{\pi_a}$

With the notion of bisimilarity congruence established in the asynchronous setting, we can prove the following properties of equators, that make them so attractive, inducing a kind of equivalence relation amongst names.

Proposition 4.6 (Equator properties ([18])).

(i) $a \twoheadleftarrow\!\!\twoheadrightarrow a \;\sim_{\pi_a}\; \nu b.(a \twoheadleftarrow\!\!\twoheadrightarrow b) \;\sim_{\pi_a}\; 0$

(ii) $a \twoheadleftarrow\!\!\!\twoheadrightarrow b \;\sim_{\pi_a}\; b \twoheadleftarrow\!\!\!\twoheadrightarrow a$

(iii) $\nu c.(a \twoheadleftarrow\!\!\!\twoheadrightarrow c \mid c \twoheadleftarrow\!\!\!\twoheadrightarrow b) \;\sim_{\pi_a}\; a \twoheadleftarrow\!\!\!\twoheadrightarrow b$

(iv) $P\{a/b\} \;\approx_{\pi_a}\; \nu b.(a \twoheadleftarrow\!\!\!\twoheadrightarrow b \mid P)$

4.2 Expressiveness: Synchrony versus Asynchrony

As mentioned above, it is easy to simulate asynchrony in terms of synchronous interaction, either by using buffers, or by simplifying the calculus. Is the converse true? The answer depends on the available operators of the calculi that we are comparing.

4.2.1 When is asynchrony enough?

Boudol, and independently Honda and Tokoro showed how to simulate the core π-calculus, i.e. the π-calculus without summation or matching operators, with π_a. This is based on a simple acknowledgement protocol. The idea is to establish a private link between sender and receiver along which the actual message will be transmitted, to avoid interference from the environment.

The sender works as follows: it first creates a private channel which passes to the receiver, and concurrently waits for an acknowledgement from the receiver. With this acknowledgement comes the private link created by the receiver along which the actual message is to be transmitted. Once the sender has received this private link, it is able to transmit the message asynchronously.

The receiver protocol is the following: once it gets the acknowledgement channel from the sender, it creates the new private link which sends back through the acknowledgement channel. Concurrently it blocks, waiting for the actual input in the private link.

Definition 4.7 (Core (synchronous) π to π_a translation ([7])).
The translation $[\![\cdot]\!] : \mathcal{P}_\pi \to \mathcal{P}_{\pi_a}$ is given by the following equations:

$$
\begin{aligned}
[\![0]\!] &\stackrel{def}{=} 0 \\
[\![P \mid Q]\!] &\stackrel{def}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![\nu x.P]\!] &\stackrel{def}{=} \nu x.[\![P]\!] \\
[\![\overline{u}\langle x\rangle.P]\!] &\stackrel{def}{=} \nu a.(\overline{u}\langle a\rangle \mid a(l).(\bar{l}\langle x\rangle \mid [\![P]\!])) && \textit{where } a,l \notin fn(P) \\
[\![u(y).P]\!] &\stackrel{def}{=} u(a).\nu l.(\overline{a}\langle l\rangle \mid l(y).[\![P]\!]) && \textit{where } a,l \notin fn(P)
\end{aligned}
$$

The correctness of the translation is specified by the adequacy with respect to the so-called "testing preorder"[2]. This preorder is based on a notion similar to ob-

[2]According to Boudol, the translation is also complete with respect to structural congruence.

servability from definition 3.31. Here, however, we will write $P \downarrow x$ when P is immediately ready to perform an input on x. This is in contrast with definition 3.31, that represented the capability of immediate interaction, both input and output. The abbreviations $P\downarrow$, $P \Downarrow x$ and $P \Downarrow$ are defined as before (recall that $P\downarrow$ means $\exists x.P \downarrow x$, $P \Downarrow x$ means $\exists P'.P \rightarrow^* P' \wedge P' \downarrow x$, $P \Downarrow$ means $\exists P'.P \rightarrow^* P' \wedge P' \downarrow$). This preorder is defined for both core-π and π_a as follows:

Definition 4.8 (Testing preorder). *The testing preorder $\sqsubseteq$ for $\mathcal{L}$ is defined by: $P \sqsubseteq_{\mathcal{L}} Q$ if and only if for all $\mathcal{L}$-contexts $\mathcal{C}$, $\mathcal{C}[P] \Downarrow_{\mathcal{L}}$ implies $\mathcal{C}[Q] \Downarrow_{\mathcal{L}}$.*

Boudol's correctness criteria is established by saying that the translation faithfully reflects how processes are ordered in terms of their receptiveness, or capability of receiving messages.

Theorem 4.9 (Boudol ([7])). *The translation $[\![\cdot]\!]$ is adequate w.r.t. $\sqsubseteq_\pi$ and $\sqsubseteq_{\pi_a}$, this is, for all terms P, Q, if $[\![P]\!] \sqsubseteq_{\pi_a} [\![Q]\!]$ then $P \sqsubseteq_\pi Q$.*

Proof. We provide only a proof sketch and the reader is referred to [7] for a complete account. First it is proven that the translation preserves and reflects observability, this is, for any P, $P \Downarrow_\pi$ if and only if $[\![P]\!] \Downarrow_{\pi_a}$. Each direction is done by induction on the length of the reductions of the term considered. For instance, in the left to right direction, the base case is when $P \downarrow w$ for some w. Then it is easy to see, by the definition of the translation, that $[\![P]\!] \downarrow w$. When $P \rightarrow^* P'$ and $P' \downarrow w$, for some P' and w, we know that some interactions have taken place. We can use this, and the fact that the translation is compositional to show by induction on the length of the reduction $P \rightarrow^* P'$ that there is a Q such that $[\![P]\!] \rightarrow^* Q$ and $Q \downarrow w$.

Once the preservation and reflection of observability has been established, we can proceed as follows. Suppose that $[\![P]\!] \sqsubseteq_{\pi_a} [\![Q]\!]$. That is, for all π_a-contexts $\mathcal{C}_a$, $\mathcal{C}_a[[\![P]\!]] \Downarrow_{\pi_a}$ implies $\mathcal{C}_a[[\![Q]\!]] \Downarrow_{\pi_a}$. We want to prove that $P \sqsubseteq_\pi Q$, i.e. for any π-context $\mathcal{C}_s$, $\mathcal{C}_s[P] \Downarrow_\pi$ implies $\mathcal{C}_s[Q] \Downarrow_\pi$. Let $\mathcal{C}_s$ be any π-context such that $\mathcal{C}_s[P] \Downarrow_\pi$. Then since observability is preserved by the translation, $[\![\mathcal{C}_s[P]]\!] \Downarrow_{\pi_a}$. Since the translation is compositional, there is a π_a-context $\mathcal{C}_a$ such that $[\![\mathcal{C}_s[P]]\!] = \mathcal{C}_a[[\![P]\!]]$, so $\mathcal{C}_a[[\![P]\!]] \Downarrow_{\pi_a}$. Now, recalling that $[\![P]\!] \sqsubseteq_{\pi_a} [\![Q]\!]$ we have that $\mathcal{C}_a[[\![Q]\!]] \Downarrow_{\pi_a}$. Noting that $[\![\mathcal{C}_s[Q]]\!] = \mathcal{C}_a[[\![Q]\!]]$ by compositionality, we have that $[\![\mathcal{C}_s[Q]]\!] \Downarrow_{\pi_a}$. Finally, since the translation reflects observability we obtain $\mathcal{C}_s[Q] \Downarrow_\pi$, as required. $\square$

4.2.2 When is asynchrony not enough?

The previous result showed how, in general, asynchronous communication can simulate synchronous communication. Yet, to some extent, it seems like synchrony is more powerful. Synchronous communication supposes some form of common agreement. However, in an asynchronous setting, one can imagine that if the parties always behave in exactly the same way, there might never be an agreement. This intuition, led Palamidessi ([36]) to show that it is not always the case that asynchronous interaction can simulate synchronous interaction.

This might appear in contradiction with the previous result, however, in that section, we were considering the core π-calculus, i.e. the calculus without non-deterministic choice. In this section we will show how non-determinism makes a difference between the two calculi.

Palamidessi's proof is based, as the intuition suggested, on the idea that asynchronous systems cannot be guaranteed to break certain symmetries. Particularly, π_a cannot solve the problem, in general, of electing a leader amongst a symmetric network of processes. Here we present her proof, but before we need some preliminaries.

Process networks and hypergraphs

A *process network* is simply a term in standard form $M \equiv \nu\vec{x}.(P_1 \mid P_2 \mid \cdots \mid P_n)$. In the rest of this section, we will rescind from the restriction at the top level, to keep notation simple. We can represent a network of processes as a *hypergraph*. Informally, a hypergraph is a graph in which one arc can connect more than two nodes.

Definition 4.10. *A* ***hypergraph*** *H is a structure (N, A, t), where N and A are finite sets, and $t : A \to \mathcal{P}(N)$. We call the elements of N and A* ***nodes*** *and* ***arcs*** *respectively, and t is called the* ***type function****.*

A hypergraph can represent a network by associating each process with a node, and each free channel with an arc. If a name x is free in P, Q, and R, then it represents an arc between those processes. We usually denote $H(M)$ the hypergraph associated with the process network M. Formally, given $M \equiv P_1 \mid P_2 \mid \cdots \mid P_n$, $H(M) = (N, A, t)$ is a hypergraph where $N \stackrel{def}{=} \{1, 2, ..., n\}$, $A \stackrel{def}{=} fn(M) \setminus \{o\}$ and $t(x) \stackrel{def}{=} \{i \in N : x \in fn(P_i)\}$. Here, o is special name through which M might send output to the external world.

We can also represent the concept of "renaming" a process network in terms of hypergraph maps. The renaming of a process consists of changing all the names according to some function. A "well"-behaved renaming preserves the structure of the communications network. This is based on the notion of *automorphism* in a hypergraph. Informally, an automorphism σ is a map from the hypergraph to itself that preserves the structure.

Definition 4.11. *Let $H = (N, A, t)$ be a hypergraph. An* ***automorphism*** *on H is a pair $\sigma = (\sigma_N, \sigma_A)$ where $\sigma_N : N \to N$ and $\sigma_A : A \to A$ such that for every $x \in A$, if $t(x) = \{i_1, i_2, ..., i_m\}$ then $t(\sigma_A(x)) = \{\sigma_N(i_1), \sigma_N(i_2), ..., \sigma_N(i_m)\}$.*

The composition of automorphisms is defined as $\sigma \circ \sigma' \stackrel{def}{=} (\sigma_N \circ \sigma'_N, \sigma_A \circ \sigma'_A)$. This composition is also an automorphism. The identity automorphism is $id \stackrel{def}{=} (id_N, id_A)$, where for any $i \in N$, $id_N(i) \stackrel{def}{=} i$ and for any $x \in A$, $id_A(x) \stackrel{def}{=} x$. The k-iteration of an automorphism σ is simply the composition k times of σ: $\sigma^k \stackrel{def}{=} \sigma \circ \sigma \circ \cdots \circ \sigma$.

Given a node i in a hypergraph, and an automorphism σ, the set resulting from iterating σ starting in i, $O_\sigma(i) \stackrel{def}{=} \{i, \sigma(i), \sigma^2(i), \sigma^3(i), ..., \sigma^{h-1}(i)\}$ where $\sigma^h(i) = i$, is called the *orbit* of i generated by σ. If the orbit of σ is unique, then for any node, the orbit coincides with the set of all nodes in the hypergraph: i.e., for each i, $O_\sigma(i) = N$ where $H = (N, A, t)$.

In the context of π-calculi, we formally define the "well"-behaved renaming of a process in terms of an automorphism σ of its associated hypergraph as follows.

Definition 4.12. *Let $M \equiv P_1 \mid P_2 \mid \cdots \mid P_n$ be a process network, with an associated hypergraph $H(M)$, and σ and automorphism on $H(M)$. A **well behaved renaming** of a term P_i in M is a function $\sigma_P : \mathcal{P}_\pi \to \mathcal{P}_\pi$ defined inductively on the structure of P_i as follows:*

$$\begin{aligned}
\sigma_P(0) &\stackrel{def}{=} 0 \\
\sigma_P(\nu x.P) &\stackrel{def}{=} \nu x'.\sigma_P(P\{x'/x\}) && \textit{where } x' \textit{ is fresh and } \sigma_A(x') = x' \\
\sigma_P(P \mid Q) &\stackrel{def}{=} \sigma_P(P) \mid \sigma_P(Q) \\
\sigma_P(P \ + \ Q) &\stackrel{def}{=} \sigma_P(P) \ + \ \sigma_P(Q) \\
\sigma_P(\overline{u}\langle x\rangle.P) &\stackrel{def}{=} \overline{\sigma_A(u)}\langle\sigma_A(x)\rangle.\sigma_P(P) \\
\sigma_P(u(x).P) &\stackrel{def}{=} \sigma_A(u)(x').\sigma_P(P\{x'/x\}) && \textit{where } x' \textit{ is fresh and } \sigma_A(x') = x'
\end{aligned}$$

*We call the structure $(\sigma_N, \sigma_A, \sigma_P)$ a **well-behaved automorphism**. Also, for the special channel o, we define $\sigma_A(o) \stackrel{def}{=} o$.*

In the definition above, all bound names are replaced by fresh names, α-conversion is performed on the term and the automorphism is extended on the new fresh names as the identity function.

Notice that since the π_a calculus is a sub-calculus of π, the definition provided also applies for π_a terms, by restricting the function to the subset of π_a terms.

We expect that such well-behaved automorphisms respect the semantics.

Lemma 4.13. *For any terms P, Q and action α, if $P \xrightarrow{\alpha} Q$ then $\sigma_P(P) \xrightarrow{\sigma_P(\alpha)} \sigma_P(Q)$.*

Proof. By induction on the derivation of $P \xrightarrow{\alpha} Q$. ☐

Computation and projection

Given a process network $M \equiv P_1 \mid P_2 \mid \cdots \mid P_n$, a computation $C : M \stackrel{\tilde{\mu}}{\Rightarrow} M^m$, is a sequence of transitions where $\tilde{\mu} = \mu_0\mu_1\mu_2...\mu_{m-1}$ is a sequence of actions. More explicitly, we have:

$$\begin{aligned} P_1 \mid P_2 \mid \cdots \mid P_n &\xrightarrow{\mu_0} P_1^1 \mid P_2^1 \mid \cdots \mid P_n^1 \\ &\xrightarrow{\mu_1} P_1^2 \mid P_2^2 \mid \cdots \mid P_n^2 \\ &\vdots \\ &\xrightarrow{\mu_{m-1}} P_1^m \mid P_2^m \mid \cdots \mid P_n^m \end{aligned}$$

If C is (ω)-infinite we write $M \xrightarrow{\tilde{\mu}}$. We say that a computation C' *extends* C, written $C \ll C'$ if there is a computation C'' such that $C' = CC''$. We write $C' \backslash C$ for C''. The "evolution" that a particular agent P_i makes in the computation C of M is called the *projection* of C over P_i, and is written $Proj(C, P_i)$. This is the sequence

$$P_i \xrightarrow{\alpha_0} P_i^1 \xrightarrow{\alpha_1} P_i^2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{m-1}} P_i^{m-1}$$

If the $u+1$ transition in C was an application of the PAR rule, involving P_i^u, then $P_i^u \xrightarrow{\alpha_u} P_i^{u+1}$ is the premise of that rule. If the transition was an application of the COMM rule, $P_i^u \xrightarrow{\alpha_u} P_i^{u+1}$ is one of the two premises. If the agent was not involved in that transition, $P_i^{u+1} = P_i^u$, i.e. the process is idle.

Symmetric networks and electoral systems

We want to define what do we mean when we say that a system or network is symmetric. Informally, we could say that in a symmetric network each process P_i has a corresponding process P_j, which is a well-behaved renaming of P_i, up-to alpha-conversion.

Definition 4.14. *Consider a process network $M \equiv P_1 \mid P_2 \mid \cdots \mid P_n$ with hypergraph $H(M) = (N, A, t)$, and a well-behaved automorphism $\sigma = (\sigma_N, \sigma_A, \sigma_P)$ on $H(M)$. We say that M is a* ***symmetric network with respect to*** *σ if for each $i \in N$, $P_{\sigma_N(i)} \equiv_\alpha \sigma_P(P_i)$. We say that M is symmetric if it is symmetric w.r.t. all automorphisms of $H(M)$.*

Now we define what is a system that can select a "leader". Informally, an *electoral system* is a process network M with a special output channel o (a free name), in which the agents will agree, sooner or later, which of them is the leader, and will announce it to the environment through o.

Definition 4.15. *A process network $M \equiv P_1 \mid P_2 \mid \cdots \mid P_n$ is an* ***electoral system*** *if for all computations C of M, there exists C' such that $C \ll C'$ and there is an $l \in \{1, 2, ..., n\}$ called the* ***leader****, such that for all agents P_i in M, $Proj(C', P_i)$ contains the action $\overline{o}\langle l \rangle$, and there is no C'' such that $C \ll C''$ that contains the action $\overline{o}\langle l' \rangle$ with $l' \neq l$.*

Symmetric electoral systems in full-π

In the full π calculus it is possible to construct a symmetric electoral system, i.e. a system that has symmetric structure and is capable of always choosing a leader. An example is the process network $M \stackrel{def}{=} P_1 \mid P_2$ where

$$P_1 \stackrel{def}{=} \nu x.\overline{u}\langle x\rangle.\overline{o}\langle l_1\rangle \ + \ v(y).\overline{o}\langle l_2\rangle$$
$$P_2 \stackrel{def}{=} \nu x.\overline{v}\langle x\rangle.\overline{o}\langle l_2\rangle \ + \ u(y).\overline{o}\langle l_1\rangle$$

This system is symmetric w.r.t. $\sigma = (\sigma_N, \sigma_A, \sigma_P)$ where $\sigma_N \stackrel{def}{=} \{(1,2),(2,1)\}$ and $\sigma_A \stackrel{def}{=} \{(u,v),(v,u),(l_1,l_2),(l_2,l_1)\}$. To see that it is also an electoral system, notice that an agreement is always reached in the first transition, i.e. M has two possibilities of interaction, namely: $P_1 \mid P_2 \xrightarrow{\tau} \overline{o}\langle l_1\rangle \mid \overline{o}\langle l_1\rangle$ or $P_1 \mid P_2 \xrightarrow{\tau} \overline{o}\langle l_2\rangle \mid \overline{o}\langle l_2\rangle$.

Confluence in π_a

The core π_a-calculus does not have non-deterministic choice, which means that if a process can perform more than two actions, they have to be performed by parallel subprocesses. Since the actions must be in parallel, the actual order in which they are executed is irrelevant, so it is always possible to reach a common state no matter what the execution path was. This is a "confluence" property.

$$\begin{tikzcd} & P \arrow[dl, "\overline{u}\langle x\rangle"'] \arrow[dr, "v(y)"] & \\ Q \arrow[dr, "v(y)"'] & & Q' \arrow[dl, "\overline{u}\langle x\rangle"] \\ & R & \end{tikzcd}$$

Formally, this property is established by the following lemma ([36]).

Lemma 4.16. *Let $P \in \mathcal{P}_{\pi_a}$. If $P \xrightarrow{\overline{u}\langle x\rangle} Q$ and $P \xrightarrow{v(y)} Q'$, for some $Q, Q' \in \mathcal{P}_{\pi_a}$ and $u, v, x, y \in \mathcal{N}_{\pi_a}$, then there is an $R \in \mathcal{P}_{\pi_a}$ such that $Q \xrightarrow{v(y)} R$ and $Q' \xrightarrow{\overline{u}\langle x\rangle} R$.*

Proof. We know that if both actions are possible, starting from P, then they must occur in parallel within P, i.e. $P \equiv \mathcal{C}_0[\mathcal{C}_1[\overline{u}\langle x\rangle] \mid \mathcal{C}_2[v(y).S]]$ for some contexts $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_2$, and some term S. The transitions $P \xrightarrow{\overline{u}\langle x\rangle} Q$ and $P \xrightarrow{v(y)} Q'$ must have been obtained by applying the PAR_t rule (with other rules as well), where the actions were introduced by the OUT_t axiom $\overline{u}\langle x\rangle \xrightarrow{\overline{u}\langle x\rangle} 0$ and the INP_t axiom $v(z).S \xrightarrow{v(y)} S\{y/z\}$.

Hence $Q \equiv \mathcal{C}_0[\mathcal{C}_1[0] \mid \mathcal{C}_2[v(z).S]]$ and $Q' \equiv \mathcal{C}_0[\mathcal{C}_1[\overline{u}\langle x\rangle] \mid \mathcal{C}_2[S\{y/z\}]]$. Therefore, if we apply PAR$_t$ again (and the other rules of the previous transition), we can construct a proof of $Q \xrightarrow{v(y)} \mathcal{C}_0[0] \mid \mathcal{C}_2[S\{y/z\}]]$ and $Q' \xrightarrow{\overline{u}\langle x\rangle} \mathcal{C}_0[0] \mid \mathcal{C}_2[S\{y/z\}]]$. Define R to be $\mathcal{C}_0[0] \mid \mathcal{C}_2[S\{y/z\}]]$. $\square$

Perennial symmetry

The gap in expressiveness between the full π-calculus and the core π_a-calculus is that in the former, we can construct a symmetric system that eventually will break the symmetry and elect a leader, as shown in the previous example, while in the latter, it is possible that the symmetry never gets broken, and therefore no leader is elected. This is formalized by the following theorem. If we start with a symmetric network, we can always construct a computation, using the confluence lemma, such that the network remains symmetric, and hence leaderless.

Theorem 4.17 (Palamidessi ([36])). *Let $M \equiv P_1 \mid P_2 \mid \cdots \mid P_n$ be a symmetric network w.r.t. σ, where $\sigma \neq id$ is a well-behaved automorphism on $H(M)$, the hypergraph of M, such that σ has only one orbit. Then M is not an electoral system.*

Proof. [3] We prove this by contradiction. Assume that M is an electoral system. We start with an empty computation C_0, and successively extend it, resulting in an infinite sequence $C_0 \ll C_1 \ll C_2 \ll \cdots \ll C_h \ll C_{h+1} \ll \cdots$. We proceed by induction in h, i.e. in the base case C_0 is the empty computation, and we construct C_{h+1} given C_h as follows.

Let C_h be $M \overset{\tilde{\mu}_h}{\Rightarrow} M^h$, we define $C_h \overset{def}{=} CC'$ where C' is the extension $M^h \overset{\tilde{\mu}_{h+1}}{\Rightarrow} M^{h+1}$. We now construct C'. Since M is electoral, an extension of C_h must contain the action $\overline{o}\langle l\rangle$ for some $l \in \{1, 2, ..., n\}$. Let μ be the first action in C', and P_i^h is the agent performing that action.

Suppose that $\mu = \overline{o}\langle l\rangle$. Given that the system is symmetric w.r.t. σ, $P^h_{\sigma(i)} \equiv \sigma(P_i^h)$. Hence $P^h_{\sigma(i)}$ contains an action $\overline{o}\langle\sigma(l)\rangle$. Therefore some extension of C_h has this action, and since M is electoral, $\sigma(l) = l$. This contradicts the condition that $\sigma \neq id$ because σ generates a unique orbit. Hence $\overline{o}\langle l\rangle$ cannot be the first action in C', so μ is either another action, or τ.

Consider the case that $\mu \neq \tau$. We now define M^{h+1}. Let P_i^{h+1} such that

$$P_i^h \xrightarrow{\mu} P_i^{h+1}$$

[3] In this proof, which follows very closely the original, to simplify notation, we will omit the subscripts N, A, and P from the component functions of σ.

By lemma 4.13 and symmetry this transition implies

$$P^h_{\sigma(i)} \xrightarrow{\sigma(\mu)} P^{h+1}_{\sigma(i)}$$
$$P^h_{\sigma^2(i)} \xrightarrow{\sigma^2(\mu)} P^{h+1}_{\sigma^2(i)}$$
$$\vdots$$
$$P^h_{\sigma^{n-1}(i)} \xrightarrow{\sigma^{n-1}(\mu)} P^{h+1}_{\sigma^{n-1}(i)}$$

Given that there is a unique orbit $M^h \equiv P^h_i \mid P^h_{\sigma(i)} \mid P^h_{\sigma^2(i)} \mid \cdots \mid P^h_{\sigma^{n-1}(i)}$. Let $M^{h+1} \equiv P^{h+1}_i \mid P^{h+1}_{\sigma(i)} \mid P^{h+1}_{\sigma^2(i)} \mid \cdots \mid P^{h+1}_{\sigma^{n-1}(i)}$, and the computation C' is $M^h \stackrel{\tilde{\mu}}{\Rightarrow} M^{h+1}$ where $\tilde{\mu} \stackrel{def}{=} \mu\sigma(\mu)\sigma^2(\mu)...\sigma^{n-1}(\mu)$ is the composition of the transitions above. We have that M^{h+1} is still symmetric w.r.t. σ.

Now, consider the case $\mu = \tau$. If this action was the result of an internal action of only one of the components P_i, then we construct M^{h+1} as in the previous case. Otherwise, the τ was the result of an interaction between two components P^h_i and P^h_j with $i \neq j$, and the transitions $P^h_i \xrightarrow{\mu_i} Q_i$ and $P^h_j \xrightarrow{\mu_j} R_j$ where μ_i and μ_j are an input and an output actions respectively (or vice-versa). Since there is a unique orbit for σ we have that for some $r \in \{1, 2, ..., n\}$, $j = \sigma^r(i)$. Let $\theta = \sigma^r$, so $P^h_j = P^h_{\theta(i)}$ and $R_j = R_{\theta(i)}$. Then, $P^h_{\theta(i)} \xrightarrow{\mu_j} R_{\theta(i)}$ which together with $P^h_i \xrightarrow{\mu_i} Q_i$ implies by COMM the transition

$$P^h_i \mid P^h_{\theta(i)} \xrightarrow{\tau} Q_i \mid R_{\theta(i)}$$

The transition $P^h_i \xrightarrow{\mu_i} Q_i$ also implies $P^h_{\theta(i)} \xrightarrow{\theta(\mu_i)} \theta(Q_i)$ by symmetry and lemma 4.13. This transition, and $P^h_{\theta(i)} \xrightarrow{\mu_j} R_{\theta(i)}$ imply by confluence, that there is an R' such that $R_{\theta(i)} \xrightarrow{\theta(\mu_i)} R'$ and $\theta(Q_i) \xrightarrow{\mu_j} R'$. Define $P^{h+1}_{\theta(i)} \stackrel{def}{=} R'$.

By symmetry on $P^h_j \xrightarrow{\mu_j} R_j$, we have $P^h_{\theta(j)} \xrightarrow{\theta(\mu_j)} \theta(R_j)$. Since $j = \theta(i)$, $\theta(j) = \theta^2(i)$, so $P^h_{\theta^2(i)} \xrightarrow{\theta(\mu_j)} \theta(R_j)$. Notice that $\theta(R_j) \equiv R_{\theta^2(i)}$ by symmetry also. The actions $\theta(\mu_i)$ and $\theta(\mu_j)$ are complementary, so we can combine $R_{\theta(i)} \xrightarrow{\theta(\mu_i)} P^{h+1}_{\theta(i)}$ and $P^h_{\theta^2(i)} \xrightarrow{\theta(\mu_j)} R_{\theta^2(i)}$ into an interaction

$$R_{\theta(i)} \mid P^h_{\theta^2(i)} \xrightarrow{\tau} P^{h+1}_{\theta(i)} \mid R_{\theta^2(i)}$$

Applying this argument repeatedly, we obtain

$$
\begin{array}{rrcl}
& R_{\theta^2(i)} \mid P^h_{\theta^3(i)} & \xrightarrow{\tau} & P^{h+1}_{\theta^2(i)} \mid R_{\theta^3(i)} \\
& & \vdots & \\
& R_{\theta^{n-2}(i)} \mid P^h_{\theta^{n-1}(i)} & \xrightarrow{\tau} & P^{h+1}_{\theta^{n-2}(i)} \mid R_{\theta^{n-1}(i)} \\
\text{and} & R_{\theta^{n-1}(i)} & \xrightarrow{\theta^{n-1}(\mu_i)} & P^{h+1}_{\theta^{n-1}(i)}
\end{array}
$$

From the transition $\theta(Q_i) \xrightarrow{\mu_j} P^{h+1}_{\theta(i)}$ we obtain $\theta^n(Q_i) \xrightarrow{\theta^{n-1}(\mu_j)} \theta^{n-1}(P^{h+1}_{\theta(i)})$ by lemma 4.13. Now, $\theta^{n-1}(P^{h+1}_{\theta(i)}) \equiv P^{h+1}_{\theta^n(i)}$ by symmetry, and, since $\theta^n = id$, this transition is the same as $Q_i \xrightarrow{\theta^{n-1}(\mu_j)} P^{h+1}_i$. So we can deduce the following interaction:

$$R_{\theta^{n-1}(i)} \mid Q_i \xrightarrow{\tau} P^{h+1}_{\theta^{n-1}(i)} \mid P^{h+1}_i$$

Finally, the composition of all these τ transitions gives us the computation C'

$$P^h_i \mid P^h_{\theta(i)} \mid P^h_{\theta^2(i)} \mid \cdots \mid P^h_{\theta^{n-1}(i)} \quad \stackrel{\tilde{\tau}}{\Rightarrow} \quad P^{h+1}_i \mid P^{h+1}_{\theta(i)} \mid P^{h+1}_{\theta^2(i)} \mid \cdots \mid P^{h+1}_{\theta^{n-1}(i)}$$

This is, M^h is the left-hand-side, and M^{h+1} the right-hand-side of this computation. Again, M^{h+1} is symmetric w.r.t. σ.

In any case, we can always extend the computation so that the resulting network is still symmetric. Thus we can construct an infinitely long computation, so M cannot be electoral. □

Non-encodability

In order to say that two languages have the same expressive power, we require that any translation $[\![\cdot]\!]$ from one language to the other preserves and reflects the semantics. Palamidessi defined as "reasonable" a semantics which

> "...distinguishes two processes P and Q whenever in some computation of P the actions on certain intended channels are different from those of any computation of Q."[36]

If we restrict ourselves to truly distributed systems, we also require that the translation preserves parallel composition $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$. Also, we might want the translations to respect renaming, i.e. $[\![\sigma(P)]\!] = \sigma([\![P]\!])$. Palamidessi calls a translation satisfying these two criteria a *uniform* translation.

Theorem 4.18 (Palamidessi ([36])). *There is no uniform translation from the full π-calculus into the core π_a-calculus that preserves a "reasonable" semantics.*

Proof. By contradiction. Suppose that $[\![\cdot]\!] : \mathcal{P}_\pi \to \mathcal{P}_{\pi_a}$ is such translation. Let $M \in \mathcal{P}_\pi$ be a symmetric and electoral system, and $M' \in \mathcal{P}_\pi$ a symmetric but non-electoral system. Then, assuming that we have a reasonable semantics for π, M and M' are different in such semantics. Since uniform translations preserve symmetry, $[\![M]\!]$ and $[\![M']\!]$ are also symmetric, but by theorem 4.17, neither is electoral, hence we cannot distinguish them, contradicting that $[\![\cdot]\!]$ preserves the "reasonable" semantics. $\square$

Interpretation

This result seems in contradiction with Boudol's encoding of synchrony in terms of the asynchronous sub-calculus. Notice however that Boudol's encoding does not consider the summation operator. Recall from section 3 that the π^{inp}-calculus allows only input guarded choice, the π^{out}-calculus allows only output guarded choice, the π^{mix} allows mixed choice, and the π^{sep}-calculus allows both input and output guarded summations, but not mixed choice. One way of interpreting Palamidessi's result is that mixed choice is not possible to encode in the asynchronous/separate choice fragment of the π-calculus. The example of a symmetric electoral system in π made use of the mixed guarded choice operator. On the other hand, the construction of theorem 4.17 relied on the confluence lemma, but this lemma does not hold in the presence of mixed-choice!

Hence this result is not just about the relation between synchrony and asynchrony, but about how (mixed)non-determinism increments the expressive power of a synchronous calculus with respect to its asynchronous, choice-free fragment.

We also have to point out that this result imposes strong requirements on translations. Particularly, requiring uniformity w.r.t. parallel composition, rules out non-distributed translations (which introduce mediator processes). When relaxing these constraints, a translation from full-π to the core π_a is possible, as Nestmann shows in [32].

Chapter 5

Internal mobility: the π_I-calculus

Following the same theme of looking for a canonical calculus of mobility, based on simplifying the original π-calculus, Sangiorgi defined the so-called "internal"-π, or π_I-calculus [41] as an intermediate step between CCS and the π-calculus.

The mechanisms of π_I are simpler than those of π, and yet they proved to be responsible for most of the expressive power of mobility while having a simpler theory, closer to that of CCS. The idea was to differentiate between two types of mobility: *internal* and *external*. In internal mobility, the output action can only send private (i.e. bound) names, whereas in external mobility it can send public (i.e. free) names. In π both types of mobility are possible. In π_I only internal mobility is possible.

The π_I-calculus is a sub-calculus of π, so it inherits its syntax with the exception of the free output action. It also inherits the structural congruence and most of the semantics.

Definition 5.1 (Core π_I terms). *Let $\mathcal{N}_{\pi I}$ be an infinite set of names ranged over by $u, v, w, x, y, z, \ldots$. As usual, $\tilde{x}$ stands for a sequence of names $x_1 x_2 \ldots x_n$. We define the set of actions $\mathcal{A}_{\pi I}$, ranged over by $\alpha, \beta, \gamma, \ldots$, and the set of fusion processes $\mathcal{P}_{\pi I}$ ranged over by $P, Q, R, \ldots$ as follows:*

$$\begin{array}{lclclclclclcl} \alpha & ::= & \tau & \mid & \overline{u}(\tilde{x}) & \mid & u(\tilde{x}) & & & & & & \\ P & ::= & 0 & \mid & \alpha.Q & \mid & \nu\tilde{x}.Q & \mid & Q \mid R & \mid & A(\tilde{x}) \end{array}$$

As in π and π_a, by including procedural definitions and calls ($A(\vec{x})$) and allowing recursion, we can define also the replication operator as: $!\,P \stackrel{def}{=} P \mid !\,P$.

As the syntax shows, input and output are symmetric actions but this symmetry is not only syntactic but also semantic as will be shown later. This symmetry is akin to that found in CCS, where for each name x there is a complementary name $\overline{x}$. We can express this idea in terms of the labels of π_I, i.e. the actions, as follows: if $\alpha = u(x)$ then $\overline{\alpha} = \overline{u}(x)$; if $\alpha = \overline{u}(x)$ then $\overline{\alpha} = u(x)$; if $\alpha = \tau$ then $\overline{\alpha} = \tau$. This operation can be extended to arbitrary terms, i.e. we write $\overline{P}$ for P replacing every prefix in P by its dual. This can be inductively defined as follows:

$$\overline{0} \stackrel{def}{=} 0 \qquad \overline{\alpha.P} \stackrel{def}{=} \overline{\alpha}.\overline{P} \qquad \overline{\nu x.P} \stackrel{def}{=} \nu x.\overline{P}$$
$$\overline{P \mid Q} \stackrel{def}{=} \overline{P} \mid \overline{Q} \quad \overline{P\ +\ Q} \stackrel{def}{=} \overline{P}\ +\ \overline{Q}$$

5.1 Semantics

We provide the UTS and the LTS based on the same notion of structural congruence from π-calculus (definition 3.8).

5.1.1 Reductions

Given that free output is not allowed, the only change in the UTS is the COMM rule. The following rule is adopted instead:

$$\frac{}{(u(x).P\ +\ R) \mid (\overline{u}(y).Q\ +\ S) \to \nu y.(P\{y/x\} \mid Q)}\ \mathrm{COMM}_{\pi I}$$

This rule expresses the idea that communication always creates a bound link between just the two processes communicating. If the scope of the link is to be enlarged to a third process this must be expressed explicitly, so it is not possible to rely on linking free (global) names.

5.1.2 Transitions

For the LTS the change is also restricted to the COMM rule. The new rule, making use of the concept of complementary actions is:

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} \nu x.(P' \mid Q')}\ \mathrm{COMM}^t_{\pi I} \quad \text{if } \alpha \neq \tau \text{ and } bn(\alpha) = \{x\}$$

The rest of the rules are the same as those of π-calculus. The input, output and silent actions are uniformly handled by the PREF_t rule.

Some basic properties are shown in the following proposition, that point out the symmetry of the semantics of actions.

Proposition 5.2.

(i) $\overline{\overline{P}} \equiv P$

(ii) $P \xrightarrow{\alpha} Q$ *if and only if* $\overline{P} \xrightarrow{\overline{\alpha}} \overline{Q}$

Proof. *(i)* Structural induction.

(ii) Induction on the derivation. We show only the case for COMM, the rest are routine. Assume that $P \xrightarrow{\tau} Q$ where the last inference was an instance of the COMM rule. Hence $P \equiv P_1 \mid P_2$, and $Q \equiv P_1' \mid P_2'$ where $P_1 \xrightarrow{\alpha} P_1'$ and $P_2 \xrightarrow{\overline{\alpha}} P_2'$. Hence by induction hypothesis $\overline{P_1} \xrightarrow{\overline{\alpha}} \overline{P_1'}$ and $\overline{P_2} \xrightarrow{\alpha} \overline{P_2'}$. So, by applying COMM, we obtain $\overline{P_1} \mid \overline{P_2} \xrightarrow{\tau} \overline{P_1'} \mid \overline{P_2'}$; i.e. $\overline{P} \xrightarrow{\tau} \overline{Q}$.

$\square$

5.1.3 Bisimilarity

The great advantage of simplifying the language is that the associated notion of ground bisimilarity is simplified. In fact, it is very close to that of ground bisimilarity.

Definition 5.3 (π_I bisimilarity). *A binary relation $\mathcal{S} \subseteq \mathcal{P}_{\pi I} \times \mathcal{P}_{\pi I}$ is called a π_I**-ground-simulation** iff for any terms $P, Q \in \mathcal{P}_{\pi I}$, $P\mathcal{S}Q$ implies that*

- *Whenever $P \xrightarrow{\alpha} P'$ and $bn(\alpha) \cap fn(Q) = \emptyset$, there is a Q' such that $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.*

*If $\mathcal{S}^{-1}$ is also a π_I-ground-simulation then $\mathcal{S}$ is called a π_I**-ground-bisimulation**.*

*We say that Q **simulates** P, or that P and Q are **similar**, written $P \dot{\prec}_{\pi I} Q$, iff there is a π_I-ground-simulation $\mathcal{S}$ such that $P\mathcal{S}Q$.*

*We say that P and Q are **bisimilar**, written $P \dot{\sim}_{\pi I} Q$, iff there is a π_I-ground-bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$.*

Note that this definition is the same as ground bisimilarity in the full π-calculus, but restricted to π_I processes. Compared to late and early variants existent in π, this notion is symmetric, and does not require a separate clause for input actions.

This notion can also be extended to its weak variant in the same way as done in section 3.2.

Is this equivalence a process congruence? Yes. That is the advantage of putting a limitation on the language. Before proving this, let us go back to the example in section 3.2. Recall that $P \stackrel{def}{=} \overline{x} \mid y$ and $Q \stackrel{def}{=} \overline{x}.y + y.\overline{x}$. They are bisimilar in π_I as well. In π, under the context $\mathcal{C}[\cdot] \stackrel{def}{=} u(y)$ we have $\mathcal{C}[P] \dot{\not\sim}_{\pi} \mathcal{C}[Q]$. The problem there arose from the fact that we replaced the y by x when we send a message $\overline{u}(x)$. The π notion of bisimilarity (both early and late) requires that, after an input action, the continuation states be themselves bisimilar *under all possible substitutions*. On the other hand, in π_I bisimilarity we do not make such a stringent requirement. The input action is matched regardless of the instantiation. In the example, this means that in π_I we have $\mathcal{C}[P] \dot{\sim}_{\pi I} \mathcal{C}[Q]$ because the action $u(y)$ in $\mathcal{C}[P]$ can always be matched by $\mathcal{C}[Q]$. It is still the case that the equivalence does not preserve arbitrary substitutions, but even so, the equivalence is preserved if we are careful not to substitute for a name that is free. Compare the following to lemma 3.21 in section 3.2.

Lemma 5.4 (Sangiorgi[41]). *If $y \notin fn(P) \cup fn(Q)$ then for any x, $P \dot{\sim}_{\pi I} Q$ implies $P\{y/x\} \dot{\sim}_{\pi I} Q\{y/x\}$.*

The fact that π_I bisimilarity does not force us to match moves for all substitutions, and the previous lemma, paves the way to process congruence. However, we do not need to talk about an "induced" congruence that is preserved under all substitutions in the sense of definition 3.23. It it enough to preserve the operators. So for the process congruence we will use the notation $\sim_{\pi I}$. In the rest of this section we will omit the subscript πI from bisimilarity and bisimilarity congruence.

Proposition 5.5 (Sangiorgi[41]). *$\dot\sim_{\pi I}$ is preserved by all operators. Hence $\dot\sim_{\pi I} = \sim_{\pi I}$*

Proof. To prove that $P \dot\sim Q$ implies $\mathcal{C}[P] \dot\sim \mathcal{C}[Q]$ we analyze each possible case for the elementary contexts.

(i) To prove $\alpha.P \dot\sim \alpha.Q$ we just need to prove that $\mathcal{S} = \{(\alpha.P, \alpha.Q) : P \dot\sim Q\}$ is a bisimulation. This is so, because the transition $\alpha.P \xrightarrow{\alpha} P$ is matched by $\alpha.Q \xrightarrow{\alpha} Q$ with $P \dot\sim Q$. This is so for all actions α, including input, because our definition of bisimilarity does not require instantiation of the object of the action.

(ii) To prove $\nu x.P \dot\sim \nu x.Q$ we just need to prove that $\mathcal{S} = \{(\nu x.P, \nu x.Q) : P \dot\sim Q\}$ is a bisimulation. Suppose that $(\nu x.P, \nu x.Q) \in \mathcal{S}$. The possible move for the first element of the pair is $\nu x.P \xrightarrow{\alpha} \nu x.P'$ where $x \notin n(\alpha)$ and $P \xrightarrow{\alpha} P'$. Since $P \dot\sim Q$, $Q \xrightarrow{\alpha} Q'$ with $P' \dot\sim Q'$. Hence, since the side condition is the same, we can apply RESTR$_t$ and thus $\nu x.Q \xrightarrow{\alpha} \nu x.Q'$. So $(\nu x.P', \nu x.Q') \in \mathcal{S}$ as required.

(iii) To prove $P \mid R \dot\sim Q \mid R$ we just need to prove that $\mathcal{S} = \{(P \mid R, Q \mid R) : P \dot\sim Q\} \cup \dot\sim$ is a bisimulation. Take a pair $(P \mid R, Q \mid R) \in \mathcal{S}$. Suppose that $P \mid R \xrightarrow{\alpha} M$. There are three possible cases depending on where the action originated:

1. It originated from R, i.e. $M \equiv P' \mid R'$ with $R \xrightarrow{\alpha} R'$ and $P' \equiv P$. Then by PAR$_t$, $Q \mid R \xrightarrow{\alpha} Q \mid R'$, and since $P \dot\sim Q$ we have that $(P \mid R', Q \mid R') \in \mathcal{S}$ as required.
2. It originated from P, i.e. $M \equiv P' \mid R'$ with $P \xrightarrow{\alpha} P'$ and $R' \equiv R$. Then, since $P \dot\sim Q$, $Q \xrightarrow{\alpha} Q'$ with $P' \dot\sim Q'$. So by PAR$_t$, $Q \mid R \xrightarrow{\alpha} Q' \mid R$. Hence $(P' \mid R, Q' \mid R) \in \mathcal{S}$ as required.
3. It originated from an interaction between P and R, so $\alpha = \tau$, i.e. $M \equiv \nu x.(P' \mid R')$ with $P \xrightarrow{\beta} P'$ and $R \xrightarrow{\overline{\beta}} R'$ for some action β whose object is x. Since $P \dot\sim Q$, we have that $Q \xrightarrow{\beta} Q'$ with $P' \dot\sim Q'$. By applying COMM$_{\pi I}$, we obtain that $Q \mid R \xrightarrow{\tau} \nu x.(Q' \mid R')$, and by item (ii) (bisimilarity preserves restriction), $M \dot\sim \nu x.(Q' \mid R')$ as required.

□

5.2 Expressiveness

So far we have seen that the π_I theory is simpler than that of π. By simplifying the language, the notion of bisimilarity was also simplified and it turned out to be a congruence. The natural question that arises is whether this was for free. Did we give up on something for obtaining the benefits? We restricted the calculus to only being able to send private names. What does this mean in terms of expressiveness? Certainly we cannot express now, at least directly, the action of exporting names that are not private. But how much has been lost by this? Apparently not too much, according to Sangiorgi. One way to see this is by looking at the kinds of things that can be done in either language. In particular we turn now our attention to the λ-calculus. The ability to simulate the λ-calculus is regarded very highly, since it represents Turing-completeness of a language. It turns out that π_I, as π itself, is powerful enough to encode the (lazy)lambda calculus.

5.2.1 Encoding the λ-calculus in π_I

Recall from chapter 2.4 the encoding of the λ-calculus into π. One of the features of this translation is the use of the free output construct, precisely the one that is not available in π_I. The challenge is how to overcome this. What follows is Sangiorgi's encoding in π_I.

The translation

Definition 5.6 (λ to π). *The translation* $[\![\cdot]\!] : \mathcal{P}_\lambda \rightarrow (\mathcal{N}_{\pi I} \rightarrow \mathcal{P}_{\pi I})$ *is given by the following equations:*

$$
\begin{aligned}
[\![x]\!]u &\stackrel{def}{=} \overline{x}(r).r \hookrightarrow u \\
[\![\lambda x.M]\!]u &\stackrel{def}{=} \overline{u}(w).w(xv).[\![M]\!]v && \textit{with } w, v \textit{ fresh} \\
[\![MN]\!]u &\stackrel{def}{=} \nu v.([\![M]\!]v \mid v(w).\overline{w}(xy).(y \hookrightarrow u \mid \mathsf{cell}(x,N)))
\end{aligned}
$$

where $a \hookrightarrow b \stackrel{def}{=} a(x).\overline{b}(y).y \hookrightarrow x$ *and* $\mathsf{cell}(x,N) \stackrel{def}{=} !\, x(w).[\![N]\!]w$.

The new derived operator can be called "relay link". A process $a \hookrightarrow b$ establishes a link between a and b in the sense that if a process sends something on a, a process listening on b will get a name linked to the name sent through a. This is a powerful derived construct, since it allows us to express a kind of buffer, and thus, an asynchronous form of communication. It does not represent a buffer strictly speaking, since the value sent through b cannot be the same as the one received through a, but it can pass a link to that value. In this sense, the encoding presents a way of delaying the interaction with particular ports as long as possible. This turns out to be enough to mimic the original encoding of λ into π, and thus suggests a possible direct encoding of π into π_I.

The rest of the encoding is very similar to Milner's, but the polarity of some interactions has been inverted. The idea, is that when a function tries to access a name (first rule), the corresponding π_I process sends the cell with that name a link to the port u that has the rest of the arguments, rather than the sending the location u directly. When a lambda abstraction is constructed (second rule), the process informs the overall expression of the location where it will receive its arguments. When a function is applied, the process obtains the location of the lambda abstraction, and then sends it the location of the cell containing the argument and a link to the rest of the arguments.

Correctness

The essence of the correctness proof is found in the properties of the relay operator, which basically express that linked channels respect the behaviour of agents in the sense that linked names act as if they were the same channel. These properties are summarized in the following lemma. It is worth noting the similarities with the properties of equators introduced in the context of π_a.

Lemma 5.7 (Sangiorgi [41]). *Let M be a λ term.*

(i) *If x, y, and z are different names, then* $\nu y.(x \hookrightarrow y \mid y \hookrightarrow z) \;\approx_{\pi I}\; x \hookrightarrow z$

(ii) *If u, x and y are different names and $y \notin fn(M)$, then* $\nu x.(x \hookrightarrow y \mid [\![M]\!]u) \;\approx_{\pi I}\; [\![M\{y/x\}]\!]u$

(iii) *If u and v are different names, then* $\nu u.(u \hookrightarrow v \mid [\![M]\!]u) \;\approx_{\pi I}\; [\![M]\!]v$

Proof. *(i)* By constructing a (weak)bisimulation.
(ii) By induction on the structure of M.
(iii) By induction on the structure of M.

□

As with the π-calculus, we prove that $[\![\cdot]\!]$ is complete w.r.t. $=_\lambda, \approx_{\pi I}$.

Theorem 5.8 (Correctness of λ to π_I translation - Sangiorgi [41]).

$$[\![(\lambda x.M)N]\!] \;\approx_{\pi I}\; [\![M\{N/x\}]\!]$$

Proof.

$$
\begin{aligned}
[\![(\lambda x.M)N]\!] &\equiv \nu v.([\![\lambda x.M]\!]v \mid v(w).\overline{w}(x_1 y).(y \hookrightarrow u \mid \mathsf{cell}(x_1,N)))\\
&\qquad \text{by definition of } [\![\cdot]\!]\\
&\equiv \nu v.(\overline{v}(w_1).w_1(xv_1).[\![M]\!]v_1 \mid v(w).\overline{w}(x_1 y).(y \hookrightarrow u \mid \mathsf{cell}(x_1,N)))\\
&\qquad \text{by definition of } [\![\cdot]\!]\\
&\xrightarrow{\tau} \nu w_1.(w_1(xv_1).[\![M]\!]v_1 \mid \overline{w_1}(x_1 y).(y \hookrightarrow u \mid \mathsf{cell}(x_1,N)))\\
&\qquad \text{by COMM}\\
&\xrightarrow{\tau} \nu x_1 y.(([\![M]\!]\{x_1/x\})y \mid y \hookrightarrow u \mid \mathsf{cell}(x_1,N))\\
&\qquad \text{by COMM}\\
&\approx \nu x_1.(([\![M]\!]\{x_1/x\})u \mid \mathsf{cell}(x_1,N))\\
&\qquad \text{by lemma 5.7(iii)}
\end{aligned}
$$

Finally we can prove

$$\nu x_1.(([\![M]\!]\{x_1/x\})u \mid \mathsf{cell}(x_1,N)) \;\approx\; [\![M\{N/x\}]\!]$$

much in the same way as in the corresponding result for π-calculus (theorem 3.35). □

5.2.2 Encoding external mobility with internal mobility

The previous section showed that π_I can simulate the λ-calculus much in the same way that π does. This suggests that internal mobility has the same power as external mobility, so it is natural to ask whether we can find a fully abstract translation from π to π_I. The answer to this question is positive. Two such encodings have been provided. One by Boreale in [6], and more recently one by Merro in [24]. Both encodings have been provided for the asynchronous setting.

Boreale's encoding is adequate w.r.t. barbed bisimilarity, and uses an intermediate language, called the *local* π calculus, in which only the output capabilities of names may be transmitted, i.e. a name x received in $u(x).P$, cannot be the subject of an input action in the body P[1]. This encoding is heavily based on the concept of relay link introduced in the encoding of the λ-calculus in the previous section.

Merro's encoding is simpler, does not use an intermediate language, and is fully-abstract w.r.t. barbed-congruence. In this section, we present Merro's embedding.

Since this embedding is in the asynchronous setting we have to clarify what is the *asynchronous* π_I-calculus or π_{Ia} for short. At first allowing only bound output

[1]Sangiorgi has also introduced a variant of π_I based on this idea of locality, but with a symmetric treatment, i.e. when a name x is sent in $\overline{u}(x).P$, then it cannot appear in output subject position in P. This variant is called π_I^-.[42]

seems to be incompatible with π_a's approach to asynchrony in which we drop output prefixing from the syntax, because that would mean that names sent would not have any scope at all. We recover this in the internal mobility setting, by restricting the syntax of π_a: a non-blocking output $\overline{u}(x)P$ is the term

$$\nu x.(\overline{u}\langle x\rangle \mid P)$$

This means that in π_{Ia} the term $a(y).(\overline{z}\langle b\rangle \mid P)$ is not allowed, and neither is the term $a(y).(\overline{z}\langle b\rangle.P)$, but the term $a(y).\nu b.(\overline{z}\langle b\rangle \mid P)$ is legal.

Definition 5.9 (π_a to π_{Ia} translation (Merro [24])). *The translation $[\![\cdot]\!] : \mathcal{P}_{\pi_a} \rightarrow \mathcal{P}_{\pi_{Ia}}$ is an homomorphism on all π_a operators except free output, which is translated according to:*

$$[\![\overline{u}\langle x\rangle]\!] \stackrel{def}{=} \nu z.(\overline{u}\langle z\rangle \mid \mathsf{equia}(x,z))$$

where $\mathsf{equia}(a,b) \stackrel{def}{=} !\, a(x).[\![\overline{b}\langle x\rangle]\!] \mid !\, b(x).[\![\overline{a}\langle x\rangle]\!]$

The agents $\mathsf{equia}(a,b)$ are closely related to equators from chapter 4. The following holds for all a, b: $[\![a \twoheadleftarrow\!\!\twoheadrightarrow b]\!] = \mathsf{equia}(a,b)$. Furthermore the properties described in proposition 4.6 also hold for these processes.

The correctness is established by the following. Here we denote $\approx^b_{\pi_{Ia}}$ for weak barbed congruence of π_{Ia} processes, and $\approx^b_{\pi_a}$ for weak barbed congruence of π_a processes.

Theorem 5.10 (Merro [24]). *The translation $[\![\cdot]\!]$ from definition 5.9 is fully abstract w.r.t. weak barbed congruence, i.e. for all $P, Q \in \mathcal{P}_{\pi_a}$ it holds:*

$$P \approx^b_{\pi_a} Q \quad \textit{if and only if} \quad [\![P]\!] \approx^b_{\pi_{Ia}} [\![Q]\!]$$

The proof of this theorem is based on a new notion of bisimilarity, called *synonymous* bisimilarity. We omit the description and proof, which are found in [24], but we simply mention that equators play the role of substitutions.

From this embedding we can conclude that internal mobility can faithfully express external mobility.

Chapter 6

Channel fusions: Fusion and χ-calculi

In the search for the "canonical calculus for concurrency", the central theme has been to look for the "right" notion of behavioural equivalence. In the previous chapters we have studied variants of the π-calculus that follow the philosophy of restricting the full calculus in one way or another so that a simple definition of bisimilarity in these sub-languages can be used as behavioural equivalence. A different approach was taken by the Fusion calculus introduced by Victor and Parrow ([45]), and independently by Fu ([11]) with the so-called χ-calculus. In this "fusion" approach the language is simplified but not restricted. This means that the π-calculus is a sub-calculus of Fusion, and thus Fusion inherits all of π's expressive power. Closely related is the work of Gardner and Wischik in π_F, fusion systems, and symmetric action calculi ([12], [49]).

As the π_I-calculus, the Fusion calculus simplifies π by making the input and output actions symmetric. However it takes the opposite approach. In π, input is binding, but not output. In π_I, both input and output are binding operators. In Fusion neither is binding. This might appear strange, particularly in the case of input, but one should not think of input and output in the same way, since the concept of communication changes. In an interaction, the "sent" and "received" names become identified through what it's called a *fusion*.

In the π-calculus, the effect of communication is local to the receiver of information. Consider for instance the following reduction in π.

$$\nu x, y.(\overline{u}\langle y\rangle.P \mid u(x).Q \mid R) \to \nu x, y.(P \mid Q\{y/x\} \mid R)$$

In this reduction, only Q is affected by the interaction. In the Fusion calculus, this interaction produces a "fusion" between the names x and y, which means that they become identified in their entire scope, thus affecting every agent in that scope:

$$\nu x, y.(\overline{u}\langle y\rangle.P \mid u\langle x\rangle.Q \mid R) \to \nu x, y(P \mid Q \mid R)\{y/x\}$$

These global effects make it appropriate for representing shared state, and in particular for encoding concurrent constraints.

Under this new notion of communication as fusion, it doesn't make sense to make the input operator binding because that would mean that the value sent only affects the receiver. In Fusion, the only binding operator is restriction. We emphasize the symmetry between input and output by adopting a new syntax, dropping the brackets from both of them.

Definition 6.1 (Fusion terms). *Let $\mathcal{N}_F$ be an infinite set of names ranged over by $u, v, w, x, y, z, \ldots$. As usual, $\tilde{x}$ stands for a sequence of names $x_1 x_2 \ldots x_n$. Let φ range over equivalence relations with domain $\mathcal{N}_F$. We define the set of actions $\mathcal{A}_F$, ranged over by $\alpha, \beta, \gamma, \ldots$, and the set of fusion processes $\mathcal{P}_F$ ranged over by $P, Q, R, \ldots$ as follows:*

$$
\begin{array}{lcl}
\alpha & ::= & \overline{u}\langle\tilde{x}\rangle \;\mid\; u\langle\tilde{x}\rangle \;\mid\; \varphi \\
P & ::= & 0 \;\mid\; \alpha.Q \;\mid\; \nu\tilde{x}.Q \;\mid\; Q \mid R \;\mid\; Q + R \\
 & & \mid\; [x = y].Q \;\mid\; [x \neq y].Q \;\mid\; A(\tilde{x})
\end{array}
$$

The action φ is an explicit fusion $\{\tilde{x} = \tilde{y}\}$. The τ action of π-calculus corresponds to the identity fusion, i.e. $\{\tilde{x} = \tilde{x}\}$ and is written 1.

It is often convenient to specify a fusion φ explicitly, but it is not strictly necessary to include it in the syntax. The term denoted $\{x = y\}.P$ is simply syntactic sugar for $\nu u.(\overline{u}\langle x\rangle \mid u\langle y\rangle.P)$. A fusion can cover several names at the same time, i.e. $\{\tilde{x} = \tilde{y}\} = \{x_1 = y_1, x_2 = y_2, \ldots, x_n = y_n\}$ where $\tilde{x} = x_1, x_2, \ldots, x_n$ and $\tilde{y} = y_1, y_2, \ldots, y_n$. The size of a sequence of names $\tilde{x}$ is denoted $|\tilde{x}|$.

As in the previous variants, by including procedural definitions and calls ($A(\vec{x})$) and allowing recursion, we can define also the replication operator as: $!\,P \stackrel{def}{=} P \mid !\,P$.

It is easy to see that the π-calculus, as well as π_I and π_a are sub-calculi of Fusion. If binding is "forced" on an input or output action, α-conversion and scope extrusion guarantee that the fusion is realized, but the result will be that of restricting the effect of the interaction, thus, simulating both π and π_I; i.e.

$$
\begin{array}{rcll}
\overline{u}\langle y\rangle.P \mid \nu x.u\langle x\rangle.Q & \equiv & \overline{u}\langle y\rangle.P \mid \nu x'.u\langle x'\rangle.Q\{x'/x\} & \text{where} \quad x' \notin n(P) \\
 & \equiv & \nu x'.(\overline{u}\langle y\rangle.P \mid u\langle x'\rangle.Q\{x'/x\}) & \\
 & \rightarrow & \nu x'.(P \mid Q\{x'/x\})\{y/x'\} & \\
 & \equiv & P \mid \nu x'.Q\{x'/x\}\{y/x'\} & \\
 & \equiv & P \mid \nu x.Q\{y/x\} &
\end{array}
$$

The original presentation of the Fusion calculus, called the *Update* calculus was monadic, i.e. only one name at a time could be communicated. The full Fusion calculus is polyadic, and this permits the identification of two names received or sent through a channel, for example:

$$\overline{u}\langle zz\rangle.P \mid u\langle xy\rangle.Q \quad \rightarrow \quad (P \mid Q)\{z/x\}\{z/y\}$$

Agents of the form $u\langle zz\rangle$, which input the same name twice, are called *catalyst* agents.

Another interesting example of the power of fusions is the *delayed input* operator, written $u(x)P$. Delayed input corresponds to non-blocking receive. The process that issues an input action is allowed to continue; if there is an output action on x in P, P blocks until the input action $u\langle x\rangle$ meets a corresponding output on u that instantiates x; then the substitution is performed. In the Fusion calculus this can be achieved simply by placing an input action in parallel with the process, i.e. $\nu x.(u\langle x\rangle \mid P)$[1], so the fusion performs the substitution. Thus, delayed input can be seen as "asynchronous receive". This cannot be encoded directly in π because the binding of the name received extends into the process performing the input action, i.e. in fusion the binding of x in $u\langle x\rangle \mid P$ extends to P, but not so in π.

The fact that π is a subset of Fusion and the examples shown here suggest that the Fusion calculus has greater expressive power than π. However, it turns out that the Fusion calculus can be encoded in π. This will be shown in section 6.3.2.

6.1 Semantics

Fusion inherits the same basic concept of structural congruence from π, but the reduction relation and the labelled transition relation are different. In order to provide the semantics of fusions, we need to determine the exact meaning of the effects of fusions as substitution. In the following, σ ranges over name substitutions.

Definition 6.2. *A substitution σ* ***agrees*** *with a fusion φ if for any $x, y \in \mathcal{N}_F$, $x\varphi y$ if and only if $\sigma(x) = \sigma(y)$. A substitution σ is said to be a* ***substitutive effect*** *of a fusion φ if σ agrees with φ and for any $x, y \in \mathcal{N}_F$, $\sigma(x) = y$ implies $x\varphi y$.*

For example, the substitutive effects of the fusion $\{a = b\}$ are $\{a/b\}$, and $\{b/a\}$. The identity fusion 1 has only one substitutive effect, namely, the identity substitution $\{x/x\}$, for any x.

For notational convenience we define $\varphi \setminus x$ as the equivalence relation that results from taking out all references of x from φ, except for the identity. Formally $\varphi \setminus x \stackrel{def}{=} \varphi \cap ((\mathcal{N}_F - \{x\}) \times (\mathcal{N}_F - \{x\})) \cup \{(x, x)\}$. For example $\{x = y, y = z\} \setminus y = \{x = z\}$. Also, we consider the domain and range of a substitution as $\mathrm{dom}(\sigma) = \{x : \sigma(x) \neq x\}$ and $\mathrm{ran}(\sigma) = \{\sigma(x) : \sigma(x) \neq x\}$.

[1]Note the similarity of this construct with the non-blocking output of π_I, as in $\nu x.(\overline{u}\langle x\rangle \mid P)$

6.1.1 Reductions

For the reduction semantics, it is enough to replace COMM with the following axiom:

$$\frac{}{\nu\tilde{z}.((\overline{u}\langle\tilde{x}\rangle.P + P') \mid (u\langle\tilde{y}\rangle.Q + Q') \mid R) \to (P \mid Q \mid R)\sigma}\ \mathrm{COMM}_F$$
$$\text{if } |\tilde{x}| = |\tilde{y}|,\ \sigma \text{ agrees with } \{\tilde{x} = \tilde{y}\},\ \mathrm{dom}(\sigma) = \tilde{z}, \text{ and } \mathrm{ran}(\sigma) \cap \tilde{z} = \emptyset$$

The rest of the rules are the same as those of π-calculus.

6.1.2 Transitions

For the labelled transition semantics the transitions will be annotated not only by input or output actions, but by fusions as well. We have to replace COMM with:

$$\frac{P \xrightarrow{u\langle\tilde{x}\rangle} P' \quad Q \xrightarrow{\overline{u}\langle\tilde{y}\rangle} Q'}{P \mid Q \xrightarrow{\{\tilde{x}=\tilde{y}\}} P' \mid Q'}\ \mathrm{COMM}^t_F \quad \text{if } |\tilde{x}| = |\tilde{y}|$$

As in π_I, The input, output and silent actions are uniformly handled by the PREF_t rule.

Finally we also add an additional rule for dealing with the scope of variables in a fusion.

$$\frac{P \xrightarrow{\varphi} P'}{\nu x.P \xrightarrow{\varphi\backslash x} P'\{y/x\}}\ \mathrm{SCOPE}^t_F \quad \text{if } x\varphi y \text{ and } x \neq y$$

This requires a little explanation. The label φ must be a fusion. The SCOPE rule states that if P can evolve into P' by performing the fusion φ in which two different names x and y are identified, then $\nu x.P$ can evolve into P' replacing x by y, in a fusion that hides x. This rule expresses the visible effect of fusions.

The rest of the rules are the same as those of π-calculus.

6.1.3 Bisimilarity

The Fusion calculus also enjoys a simple notion of bisimulation.

Definition 6.3 (Fusion bisimilarity). *A relation $\mathcal{S} \subseteq \mathcal{P}_F \times \mathcal{P}_F$ is called a* ***strong fusion simulation*** *iff $P\mathcal{S}Q$ implies:*

- *Whenever $P \xrightarrow{\gamma} P'$ and $bn(\gamma) \cap fn(Q) = \emptyset$ then for some Q', $Q \xrightarrow{\gamma} Q'$ and $P'\sigma\mathcal{S}Q'\sigma$ for some substitutive effect σ of γ.*

If $\mathcal{S}^{-1}$ is also a strong fusion simulation, then $\mathcal{S}$ is called a ***strong open bisimulation****. We say that $P\dot{\prec}_F Q$ if there is a strong fusion simulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\prec}_F$ is called* ***strong fusion similarity****. We say that $P\dot{\sim}_F Q$ if there is a strong fusion bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$. $\dot{\sim}_F$ is called* ***strong fusion bisimilarity****.*

As usual, we obtain the corresponding weak bisimilarity $\dot{\approx}_F$ by replacing the strong transitions with weak transitions.

This definition of bisimilarity coincides with ground-bisimilarity in the treatment of input and output actions, but special care has to be given to fusions. If the action γ was a fusion, we compare the continuations only after the substitution has taken place.

Unfortunately, fusion bisimulation is not a congruence, for reasons similar to ground bisimilarity not being a congruence in the full-π-calculus. Hence, equivalence is defined as follows:

Definition 6.4 (Hyperequivalence). *A **hyperbisimulation** is a fusion bisimulation closed under substitution. We say that two processes P and Q are **hyperequivalent**, written $P \sim_F Q$ if there is a hyperbisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$.*

We define similarly weak hyperequivalence ($\approx_F$).

When considering the language without mismatch, the following has been established[2]:

Theorem 6.5 (Victor ([45])). *Hyperequivalence is a congruence.*

There is a close relationship between equators as described in chapter 4, and fusions. With the notion of hyperequivalence established, it is easy to prove the following properties, analogous to proposition 4.6:

Proposition 6.6 (Fusion properties).

(i) $\{a = a\} \sim_F \nu b.(\{a = b\}) \sim_F 0$

(ii) $\{a = b\} \sim_F \{b = a\}$

(iii) $\nu c.(\{a = c\} \mid \{c = b\}) \approx_F \{a = b\}$

(iv) $P\{a/b\} \approx_F \nu b.(\{a = b\} \mid P)$

6.2 Some variants of Fusion

In this section we introduce two simplifications of the Fusion calculus that turn out to have some important significance from the expressiveness point of view: the *asynchronous fusion* calculus ([23]) and the fusion calculus of *solos* ([19]).

[2] Fu and Yang show in [11] the problems introduced by the mismatch operator, and propose some variants to the definition of bisimulation.

6.2.1 Asynchronous fusion

This variation of Fusion treats the sending action as non-blocking in the style of π_a. As in π_a this is achieved by dropping the continuations of output actions. By analogy with π_a we also get rid of the choice operator. The syntax is

$$P \quad ::= \quad 0 \quad | \quad \varphi \quad | \quad u\langle\tilde{x}\rangle.Q \quad | \quad \overline{u}\langle\tilde{x}\rangle \quad | \quad \nu\tilde{x}.Q \quad | \quad Q \mid R \quad | \quad A(\tilde{x})$$

The change in the reduction relation is simple; we replace the COMM rule by:

$$\frac{}{\nu\tilde{z}.(\overline{u}\langle\tilde{x}\rangle \mid u\langle\tilde{y}\rangle.Q \mid R) \to (Q \mid R)\sigma}\ \mathrm{COMM}_{Fa}$$
$$\text{if } |\tilde{x}| = |\tilde{y}|,\ \sigma \text{ agrees with } \{\tilde{x} = \tilde{y}\},\ \mathrm{dom}(\sigma) = \tilde{z}, \text{ and } \mathrm{ran}(\sigma) \cap \tilde{z} = \emptyset$$

For the labelled transitions we keep the same rules of Fusion, including COMM, but replace PREF^t_F with INP and OUT as was done in π_a, i.e.

$$\frac{}{\overline{u}\langle\tilde{x}\rangle \xrightarrow{\overline{u}\langle\tilde{x}\rangle} 0}\ \mathrm{OUT}^t_{Fa} \qquad \frac{}{u\langle\tilde{x}\rangle.P \xrightarrow{u\langle\tilde{x}\rangle} P}\ \mathrm{INP}^t_{Fa}$$

6.2.2 Solos

The asynchronous Fusion calculus simplifies Fusion, but breaks the original symmetry of input and output. It is possible however to recover the symmetry by dropping continuations from input actions as well, i.e. all input actions are delayed, or non-blocking "receive" operations. We call the resulting calculus "Solos" [19]. The syntax is

$$P \quad ::= \quad 0 \quad | \quad \varphi \quad | \quad u\langle\tilde{x}\rangle \quad | \quad \overline{u}\langle\tilde{x}\rangle \quad | \quad \nu\tilde{x}.Q \quad | \quad Q \mid R \quad | \quad A(\tilde{x})$$

The change in the reduction relation is simple; we replace the COMM rule by:

$$\frac{}{\nu\tilde{z}.(\overline{u}\langle\tilde{x}\rangle \mid u\langle\tilde{y}\rangle \mid R) \to R\sigma}\ \mathrm{COMM}_{Fs}$$
$$\text{if } |\tilde{x}| = |\tilde{y}|,\ \sigma \text{ agrees with } \{\tilde{x} = \tilde{y}\},\ \mathrm{dom}(\sigma) = \tilde{z}, \text{ and } \mathrm{ran}(\sigma) \cap \tilde{z} = \emptyset$$

For the labelled transitions we keep the same rules of Fusion, including COMM, but replace PREF^t_F with

$$\frac{}{\alpha \xrightarrow{\alpha} 0}\ \mathrm{PREF}^t_{Fs}$$

Surprisingly, this simplification does not limit the expressive power; Fusion can be encoded in Solos.

6.3 Expressiveness

In this section we explore the expressiveness relations of Fusion, both between the Fusion variants alone, and between Fusion and π-calculi.

6.3.1 From Fusion to Solos to Asynch-Fusion and back

It should be clear that Asynchronous Fusion is a subset of Fusion, and Solos is a subset of Asynchronous Fusion. Laneve and Victor have proved in [19] that the Fusion can be encoded in Solos (and therefore in Asynch-Fusion).

Laneve and Victor provide two encodings of Fusion in Solos, both fully-abstract with respect to barbed bisimulation. The first encoding is compositional, but makes use of the match operator. The second encoding does not use match, but is not compositional, or uniform in the sense of Palamidessi's definition (section 4.2.2). In both encodings, catalyst agents (e.g. $u\langle zz\rangle$) play a central role.

We present now the first encoding.

Definition 6.7 (Fusion to Solos translation ([19])). *The translation $[\![\cdot]\!]$ from Fusion processes to Solos processes is an homomorphism on all operators except input and output which are handled by the following:*

$$[\![u\langle\tilde{x}\rangle.P]\!] \stackrel{def}{=} \nu wz.(u\langle\tilde{x}zww\rangle \mid [z=w][\![P]\!])$$
$$[\![\overline{u}\langle\tilde{x}\rangle.P]\!] \stackrel{def}{=} \nu wz.(\overline{u}\langle\tilde{x}wwz\rangle \mid [z=w][\![P]\!])$$

This encoding enjoys a pleasing symmetry in the translation of the complementary actions. The correctness of this translation is established with respect to barbed bisimulation for fusion. The notion of barbed bisimulation is analogous to the one described in definition 3.32, where observability is defined as in definition 3.31 changing the case of binding input with Fusion's free input. We denote $\dot{\sim}^b_F$ and $\dot{\approx}^b_F$ respectively the largest strong and weak barbed bisimulation for Fusion processes, and $\sim^b_F$ and $\approx^b_F$ strong barbed congruence and weak barbed congruence.

Theorem 6.8 (Laneve, Victor ([19])). *The translation $[\![\cdot]\!]$ of definition 6.7 is fully abstract with respect to strong barbed bisimulation.*

We omit the complete proof, since it is rather simple, following the lines of theorem

2.10, but we show the interesting part, i.e. how a basic interaction is simulated:

$$
\begin{aligned}
&[\![\nu x.(\overline{u}\langle y\rangle.P \mid u\langle x\rangle.Q)]\!] \\
&\stackrel{def}{=} \nu x.(\nu zw.(\overline{u}\langle ywwz\rangle \mid [z=w][\![P]\!]) \mid \nu zw.(u\langle xzww\rangle \mid [z=w][\![Q]\!])) \\
&\equiv \nu x w_1 w_2 z_1 z_2.(\overline{u}\langle y w_1 w_1 z_1\rangle \mid [z_1 = w_1][\![P]\!] \mid u\langle x z_2 w_2 w_2\rangle \mid [z_2 = w_2][\![Q]\!]) \\
&\to \nu z_1.([z_1 = w_1][\![P]\!] \mid [z_2 = w_2][\![Q]\!])\{y/x, z_1/w_1, z_1/w_2, z_1/z_2\} \\
&\equiv ([\![P]\!] \mid [\![Q]\!])\{y/x\} \\
&\equiv [\![(P \mid Q)\{y/x\}]\!]
\end{aligned}
$$

The encoding of this section can be extended to handle separate-guarded choice by using the mismatch operator as follows:

$$
\begin{aligned}
[\![\Sigma_I u_i\langle\tilde{x}_i\rangle.P_i]\!] &\stackrel{def}{=} \nu wz.\Pi_I[w \neq z](u_i\langle\tilde{x}_i zww\rangle \mid [w=z][\![P_i]\!]) \\
[\![\Sigma_I \overline{u_i}\langle\tilde{x}_i\rangle.P_i]\!] &\stackrel{def}{=} \nu wz.\Pi_I[w \neq z](\overline{u_i}\langle\tilde{x}_i wwz\rangle \mid [w=z][\![P_i]\!])
\end{aligned}
$$

This extension can be applied to translate single prefixes $\alpha.P$, by treating them as a special case of a summation with only one summand.

6.3.2 From π to Fusion and back

We have seen that the Fusion calculus simplifies the π calculus, and yet it extends it. The embedding of the π-calculus in Fusion is very simple: the input operator of π is mapped to input with explicit binding of the object of the input action, and the rest of the operators are the same:

$$u(x).P \;\mapsto\; \nu x.u\langle x\rangle.P$$

An immediate corollary is that we can encode easily the λ calculus in Fusion.

Several aspects of Fusion, such as the catalyst operators, and delayed input suggest that the expressiveness of Fusion is greater than that of π, since none of those operations are directly encodable in π. However, it turns out that this is not the case. Fusion is encodable in π.

Recall from the previous section that any Fusion term can be encoded in terms of Solos, and this is automatically a term in Asynchronous-Fusion. In [23] Merro provides a fully-abstract encoding of Asynch-Fusion in π_a.

Note that since the full-π calculus is encodable in Fusion, this encodability of Fusion in π_a appears to be in contradiction with Palamidessi's theorem on the impossibility of encoding the full π-calculus into π_a. However Palamidessi's impossibility result restricted the types of translation to be "uniform" and "reasonable", but as Nestmann showed in [32], when relaxing those constraints on the translations, it is actually possible to go from full-π to π_a. Merro's encoding, as well as Victor and Laneve's are not uniform, and therefore there is no contradiction with Palamidessi's result. Furthermore, these encodings cover only separate-guarded choice, and not mixed-choice.

In this section we present Merro's encoding of Asynch-Fusion into π_a. In this translation we extend the notion of equator to a polyadic setting by defining $\tilde{a} \twoheadleftarrow\!\!\twoheadrightarrow \tilde{b}$ to denote $a_1 \twoheadleftarrow\!\!\twoheadrightarrow b_1 \mid a_2 \twoheadleftarrow\!\!\twoheadrightarrow b_2 \mid \cdots \mid a_n \twoheadleftarrow\!\!\twoheadrightarrow b_n$.

Definition 6.9 (Asynchronous-Fusion to core π_a translation ([23])). *The translation $[\![\cdot]\!]$ from core π_a processes to Asynchronous Fusion processes is an homomorphism on all operators except input which is handled by the following:*

$$[\![u\langle\tilde{x}\rangle.P]\!] \stackrel{def}{=} u(\tilde{z}).(\tilde{z} \twoheadleftarrow\!\!\twoheadrightarrow \tilde{x} \mid [\![P]\!]) \qquad \textit{where } z \notin fn(P)$$

This encoding highlights the close relation between fusions and equators. The correctness is established with respect to *closed barbed congruence*, which is barbed congruence as described above, but instead of being closed under arbitrary contexts, it is closed under contexts whose free names appear only in output subject position (analogous to local π).

Theorem 6.10 (Merro ([23])). *The translation of definition 6.9 is fully-abstract with respect to closed barbed congruence.*

Again, we omit the complete proof of correctness, referring the reader to [23], but we show how the basic fusion interaction takes place. Of particular interest is how the global effects of fusions are simulated through equators.

$$\begin{array}{lll}
& [\![\nu x.(\overline{u}\langle y\rangle \mid u\langle x\rangle.Q \mid R)]\!] & \\
\stackrel{def}{=} & \nu x.(\overline{u}\langle y\rangle \mid u(z).(z \twoheadleftarrow\!\!\twoheadrightarrow x \mid [\![Q]\!]) \mid [\![R]\!]) & \text{by def 6.9} \\
\rightarrow & \nu x.(y \twoheadleftarrow\!\!\twoheadrightarrow x \mid [\![Q]\!] \mid [\![R]\!]) & \text{by COMM}_a \\
\stackrel{def}{=} & \nu x.(y \twoheadleftarrow\!\!\twoheadrightarrow x \mid [\![Q \mid R]\!]) & \text{by def 6.9} \\
\approx_{\pi_a} & [\![Q \mid R]\!]\{y/x\} & \text{by proposition 4.6(iv)} \\
\equiv & [\![(Q \mid R)\{y/x\}]\!] & \text{as required}
\end{array}$$

Note that the substitution property of equators (proposition 4.6(iv)) holds only in the asynchronous π calculus and not in the synchronous π. However, we can encode the synchronous Fusion calculus in π_a by first translating a synchronous Fusion term into Solos, as was shown in the previous section, and then we apply this encoding, since Solos is a subset of Asynchronous Fusion.

Chapter 7

Concurrent Constraint Programming

So far we have focused on calculi specially geared towards mobility. Now we look at a paradigm with a different focus. The family of *Concurrent Constraint Programming* languages ([43], [39], [35]), or CCP for short, is a paradigm based on the shared memory model of communication. In the basic model, a system consists of a shared *constraint store*, and a collection of agents or processes that perform two basic operations: *telling* a constraint, i.e. adding a constraint to the store, and *asking* if a constraint is entailed by the store. The ask operator is blocking, so the agent performing an ask will continue processing only when the constraint is satisfied by the store. The tell operation on the other hand is non-blocking, thus making this an asynchronous model. In the basic model, the constraint store is monotone in the sense that one can only add constraints, but cannot remove them.[1]

The syntax of CCP is defined as follows:

Definition 7.1 (CCP Terms). *Assume a set of names $\mathcal{N}_{cc}$, a set of assertions or constraints $\mathcal{A}_{cc}$, ranged over by σ, ψ, φ, etc. The set of CC-terms, denoted $\mathcal{P}_{cc}$, ranged over by P, Q, R, etc, is defined inductively by the syntax shown in table 7.1.*

The Nil, Restriction, Parallel composition, Summation and Identifier operators play the same roles as their π-calculi counterparts. Tell, posts a constraint in the global store. Ask, queries the store to see if its condition is entailed. If so, it continues processing. If not, the agent asking, blocks until its condition is satisfied by the store.

Since a system is specified by stating constraints over variables, a variable does not necessarily have a specific value. A variable can have a partially defined value, in the sense that a constraint is a restriction on the variable, determining the subset in which its value lies, but not fixing the variable to one particular point.[2] Computation

[1] This assumption is dropped in a variant of CCP called "Linear CCP".

[2] In the context of Logic Programming, such variables are usually called "logical variables".

P	$::=$	$\top$	Nil
	$\mid$	$tell(\sigma)$	Tell
	$\mid$	$ask(\sigma) \to Q$	Ask
	$\mid$	$\exists x.Q$	Restriction
	$\mid$	$Q \wedge R$	Parallel composition
	$\mid$	$Q + R$	Summation
	$\mid$	$A(\tilde{x})$	Procedural call

Table 7.1: The syntax of CCP

proceeds by reducing constraints as much as possible, thus narrowing or refining the possible values of variables.

The declarative nature of CCP suggests a close relationship with Logic. This turned out to be a tight connection. Mendler, Panangaden, Scott and Seely [22] established a precise link, in categorical terms, between CCP and (a subset of) Intuitionistic Logic. This is a link in the sense that they are both instances of a more general (category theoretical) concept called a "hyperdoctrine". This means that one can read CCP programs as sentences: telling a constraint corresponds to simply asserting the formula of the constraint; asking if a constraint is true, and then executing a process corresponds to (a limited form of) logical implication; parallel composition corresponds to logical conjunction; and the restriction operator corresponds to existential quantification. This correspondence is not merely syntactic, but semantic. Computation in CCP can be interpreted as logical inference. The reduction of a term is a proof. In a similar fashion, Linear CCP is linked to Girard's Linear Logic[13]. This relation of CCP with logic is very significant, however, in this chapter we will emphasise the operational view of CCP rather than the logical perspective.

CCP is not one particular programming language. It is a family of languages. It is parametrized by a *constraint system* (CS for short) which specifies what kind of constraints the store handles. A CS is a language for talking about the entities that the programs deal with, together with an "entailment" relation that provides the mechanism for answering queries. The store can be seen then, as a formula representing the combination of the constraints.

An example of a typical CS has variables and numbers as terms, and constraints are equations and inequations between them. For instance, $X \leq 8, U = X, Y \geq U$ are elements in this kind of CS. Entailment is assumed to be compatible with arithmetic, e.g. the formulas given above entail $U \leq 8$, $U \leq Y$, $Y \geq X$, etc.

Another useful CS is the Herbrand constraint system. In this system, the basic entities are names representing variables or constant symbols, and terms are function or predicate symbols with names and terms as arguments. For example, a Herbrand

CS may have elements such as the formulas $R(X, k)$, $R(a, g(X), b)$, $f(g(b), Y) = f(X, h(X))$, etc., where a, b, k are constant symbols, X, Y are variables, $R, =$ are predicate symbols, and f, g, h are function symbols. Entailment is based on the notion of unification between terms related by the special equality predicate ($=$). For instance, the formula $f(g(b), Y) = f(X, h(X))$ entails $X = g(b)$, $Y = h(g(b))$, etc.

7.1 Semantics

7.1.1 The constraint system

The semantics of CCP is built upon the notion of constraint system. This notion abstracts away the inner workings of the query-answer mechanism. In the following definition ([39]) we use the notation $\mathcal{P}_f(A)$ for the set of *finite* subsets of A, and $A \subseteq_f B$ to mean that A is a *finite* subset of B.

Definition 7.2 (Constraint Systems). *A structure $(\mathcal{A}, \vdash)$ is called a* ***constraint system****, if*

- *$\mathcal{A}$ is a non-empty, countable set of* ***assertions*** *or (primitive)***constraints*** *(according to some syntax).*
- *$\vdash \subseteq \mathcal{P}_f(\mathcal{A}) \times \mathcal{P}_f(\mathcal{A})$, called an* ***entailment*** *relation, satisfies the following, for any $\sigma, \psi, \varphi \in \mathcal{P}_f(\mathcal{A})$:*
  - *(i) If $\psi \subseteq \sigma$ then $\sigma \vdash \psi$*
  - *(ii) If $\sigma \vdash \varphi$ and $\varphi \vdash \psi$ then $\sigma \vdash \psi$*

We abbreviate $\sigma \vdash \{p\}$ as $\sigma \vdash p$. A ***store*** *or* ***element*** *of $(\mathcal{A}, \vdash)$ is a set of assertions σ such that if $\sigma \subseteq \mathcal{P}_f(\mathcal{A})$ and if for any $p \in \mathcal{A}$ and $\sigma' \subseteq_f \sigma$ such that $\sigma' \vdash p$ then $p \in \sigma$. The set of all elements of $(\mathcal{A}, \vdash)$ is denoted $|\mathcal{A}_{cc}|$. Two stores are equivalent, written $\sigma \vdash\dashv \psi$ iff $\sigma \vdash \psi$ and $\psi \vdash \sigma$.*

This definition is extended to handle the concept of hiding private names from the store, by the notion of *cylindric constraint systems* ([39]). The idea is to model hiding of a variable x with a function $\dot{\exists}_x$ that takes a store as argument and returns the store without any references of x. We also extend the notion constraint system to handle name equations, modeled by the so-called *diagonal elements.*

Definition 7.3. *A structure $(\mathcal{A}, \vdash, \mathrm{Var}, \mathcal{H})$ is called a* ***cylindric constraint system*** *if:*

(i) $(\mathcal{A}, \vdash)$ *is a constraint system.*

(ii) Var *is an infinite set of variables.*

(iii) $\mathcal{H} = \{\dot{\exists}_x : \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A}) \mid x \in \mathrm{Var}\}$ *such that for each $x, y \in \mathrm{Var}$, $\sigma, \psi \in \mathcal{P}(\mathcal{A})$:*

- $\sigma \vdash \dot{\exists}_x \sigma$
- *if* $\sigma \vdash \psi$ *then* $\dot{\exists}_x \sigma \vdash \dot{\exists}_x \psi$
- $\dot{\exists}_x(\sigma \cup \dot{\exists}_x \psi) \dashv\vdash \dot{\exists}_x \sigma \cup \dot{\exists}_x \psi$
- $\dot{\exists}_x \dot{\exists}_y \sigma \dashv\vdash \dot{\exists}_y \dot{\exists}_x \sigma$

(iv) For every $x, y \in \mathrm{Var}$ *there is a* $d_{xy} \in \mathcal{A}$*, called a* ***diagonal element****, such that:*

- $\emptyset \vdash d_{xx}$
- *if* $x \neq y$ *then* $\{d_{xy}\} \dashv\vdash \dot{\exists}_z\{d_{xy}, d_{yz}\}$
- $\{d_{xy}\} \cup \dot{\exists}_x(\sigma \cup \{d_{xy}\}) \vdash \sigma$

Since a store is a set containing assertions, we can think of it as the logical conjunction of these assertions. The empty store corresponds to true, an inconsistent store corresponds to false, the function $\dot{\exists}_x$, corresponds, as the notation suggests, to existential quantification, and a diagonal element d_{xy} represents the formula $x = y$. [3]

We extend the syntax of CCP so that the argument of *tell* and the argument of *ask* are elements in $|\mathcal{A}_{cc}|$, not just primitive constraints in $\mathcal{A}_{cc}$. The notation $tell(\sigma_1 \cup \sigma_2)$, also written $tell(\sigma_1 \wedge \sigma_2)$, should be interpreted as adding the logical conjunction of σ_1 and σ_2 to the global store.

In the rest of this section we assume that we have fixed a cylindric constraint system $(\mathcal{A}_{cc}, \vdash, \mathcal{N}_{cc}, \mathcal{H})$.

7.1.2 Structural congruence

As with π-calculi, we define a notion of structural congruence in terms of the concept of process congruence, which has to be adapted to the context of CCP.

Definition 7.4 (CC-process congruence). *A* ***CC-process congruence*** $\cong_{cc} \subseteq \mathcal{P}_{cc} \times \mathcal{P}_{cc}$ *is an equivalence relation among agents such that for all* $P, P' \in \mathcal{P}_{cc}$ *if* $P \cong_{cc} P'$ *then:*

(i) For any $\sigma, \sigma' \in |\mathcal{A}_{cc}|$ *such that* $\sigma \dashv\vdash \sigma'$,

- $tell(\sigma) \cong_{cc} tell(\sigma')$
- $ask(\sigma) \to P \cong_{cc} ask(\sigma') \to P'$

(ii) For any name x, $\exists x.P \cong_{cc} \exists x.P'$

(iii) For any agent Q,

[3] Notice that in a set φ containing only diagonal elements, $\dot{\exists}_x \varphi$ coincides with $\varphi \setminus x$ as defined in the context of the Fusion calculus in section 6.1.

- $P \wedge Q \cong_{cc} P' \wedge Q$
- $Q \wedge P \cong_{cc} Q \wedge P'$
- $P + Q \cong_{cc} P' + Q$
- $Q + P \cong_{cc} Q + P'$

Now we can define the structural congruence in the same way we did for π-calculi. We omit the formal definitions for substitution and alpha-conversion. They are the standard rules to avoid name capturing, as in the lambda calculus or in the π-calculus.

Definition 7.5 (CC-structural congruence). *The CC-structural congruence* $\equiv_{cc} \subseteq \mathcal{P}_{cc} \times \mathcal{P}_{cc}$ *is the CC-process congruence that satisfies :*

(i) $P \equiv Q$ *if* $P \equiv_\alpha Q$

(ii) $(\mathcal{P}_{cc}, \wedge, \top)$ *is an Abelian (commutative) monoid:*

- $P \wedge \top \equiv P$
- $P \wedge Q \equiv Q \wedge P$
- $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$

(iii)
- $P + Q \equiv Q + P$
- $(P + Q) + R \equiv P + (Q + R)$

(iv) $\exists x.\top \equiv \top$

(v) $\exists x.\exists y.P \equiv \exists y.\exists x.P$

(vi) $P \wedge \exists x.Q \equiv \exists x.(P \wedge Q)$ *if* $x \notin fn(P)$

(vii) $tell(\sigma_1) \wedge tell(\sigma_2) \equiv_{cc} tell(\psi)$ *if* $\sigma_1 \cup \sigma_2 \dashv\vdash \psi$

The axioms for structural congruence are the same as the corresponding axioms in π-calculi from definition 3.8, interpreting 0 as $\top$, νx as $\exists x$, $\mid$ as $\wedge$, $+$ as $+$, taking out the axioms for match and mismatch, and adding the last axiom for combining constraints. Notice that if the elements of the constraint system are only diagonal elements, i.e. name equations, we could also have an axiom analogous to the match axiom, corresponding to ask as in:

$$ask(x = y) \to \exists z.P \equiv \exists z.ask(x = y) \to P \quad \text{if } z \text{ is a different name of } x \text{ and } y.$$

Since we are considering any constraint system, and not restricting ourselves to name equations, we do not use this axiom in this presentation of CCP, but we will be able to show that these terms are bisimilar.

$$\frac{}{tell(\sigma_1) \wedge tell(\sigma_2) \to tell(\psi)} \text{TELL} \quad \text{if } \sigma_1 \cup \sigma_2 \vdash\!\dashv \psi$$

$$\frac{}{(ask(\psi) \to P) \wedge tell(\sigma) \to P \wedge tell(\sigma)} \text{ASK} \quad \text{if } \sigma \vdash \psi$$

$$\frac{P \to P'}{P \wedge Q \to P' \wedge Q} \text{PAR} \qquad \frac{P \to P'}{P + Q \to P'} \text{SUM}$$

$$\frac{P \to P'}{\exists x.P \to \exists x.P'} \text{RESTR} \qquad \frac{P\{\tilde{y}/\tilde{x}\} \to P'}{A(\tilde{y}) \to P'} \text{ID} \quad \text{if } A(\tilde{x}) \stackrel{def}{=} P$$

$$\frac{P \to Q}{P' \to Q'} \text{CONGR} \quad \text{if } P \equiv P' \text{ and } Q \equiv Q'$$

Table 7.2: CCP reduction rules

7.1.3 Reductions

We now introduce the reduction semantics of CCP.[4] The UTS $(\mathcal{P}_{cc}, \mathcal{CS}, \equiv_{cc}, \to_{cc})$, is parametrized by the cylindric constraint system $\mathcal{CS} = (\mathcal{A}_{cc}, \vdash, \mathcal{N}_{cc}, \mathcal{H})$. As usual, we omit the subscript cc from the congruence relation and from the reduction relation when it is clear that we are talking about CC terms.

Definition 7.6 (CC reduction). *The reduction relation* $\to \subseteq \mathcal{P}_{cc} \times \mathcal{P}_{cc}$ *is the smallest relation over CC-processes satisfying the rules in table 7.2.*

Notice that the difference with the UTS for π-calculi is in the interaction rule (ask) and the tell rule, which assume the existence of a constraint system. In this presentation of the semantics, we can view the store as just another process or set of processes in parallel with the rest of the system. The store is transparently distributed, and not centralized. These rules also make explicit the permanent nature of constraints: the effect of telling a constraint is equivalent to having a process perennially asserting the constraint, so it remains active even after an interaction with an ask. [5]

[4]The definition provided here is based on the semantics for the ρ-calculus ([34]), which extends the standard CCP model with first-order functional abstraction. Here we present only the part of the semantics considered to be "pure CCP", i.e., according to the model introduced in this chapter, without reference to functional abstraction.

[5]In Linear CCP, the constraint would disappear after such interaction.

$$\frac{}{tell(\phi) \xrightarrow{(\sigma,\sigma\cup\phi)} \top}\ \mathrm{TELL}_t \qquad \frac{}{ask(\phi) \to P \xrightarrow{(\sigma,\sigma)} P}\ \mathrm{ASK}_t \text{ if } \sigma \vdash \phi$$

$$\frac{P \xrightarrow{(\sigma,\sigma')} P'}{P \ + \ Q \xrightarrow{(\sigma,\sigma')} P'}\ \mathrm{SUM}_t \qquad \frac{P \xrightarrow{(\sigma,\sigma')} P'}{P \wedge Q \xrightarrow{(\sigma,\sigma')} P' \wedge Q}\ \mathrm{PAR}_t$$

$$\frac{P \xrightarrow{(\dot{\exists}_x \sigma,\phi)} P'}{\exists x.P \xrightarrow{(\sigma,\sigma\cup\dot{\exists}_x \phi)} \exists x.(tell(\phi) \ \wedge \ P')}\ \mathrm{RESTR}^1_t$$

$$\frac{P \xrightarrow{(\phi\cup\dot{\exists}_x \sigma,\phi')} P'}{\exists x.(tell(\phi) \ \wedge \ P) \xrightarrow{(\sigma,\sigma\cup\dot{\exists}_x \phi')} \exists x.(tell(\phi') \ \wedge \ P')}\ \mathrm{RESTR}^2_t$$

$$\frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{(\sigma,\sigma')} P'}{A(\tilde{y}) \xrightarrow{(\sigma,\sigma')} P'}\ \mathrm{ID}_t \text{ if } A(\tilde{x}) \stackrel{def}{=} P$$

$$\frac{P \xrightarrow{(\sigma,\sigma')} Q}{P' \xrightarrow{(\sigma,\sigma')} Q'}\ \mathrm{CONG}_t \quad \text{ if } P \equiv P' \text{ and } Q \equiv Q'$$

Table 7.3: CCP transition rules

7.1.4 Transitions

There are several approaches for presenting the LTS of CCP. In the first form, labels are actions $\sigma!$ and $\sigma?$, representing tell and ask respectively. A second presentation, contrasting with the UTS, makes an explicit separation between the store and the processes. Labels in this presentation are actually pairs of stores, that represent the state of the store before and after the transition takes place. We write this in two ways: $P \xrightarrow{(\sigma,\sigma')} P'$ or equivalently, $(P,\sigma) \to (P',\sigma')$. This presentation, based on pairs of stores as labels, is the one given here.

The LTS $(\mathcal{P}_{cc}, \mathcal{CS}, \equiv_{cc}, \to_{cc})$, is parametrized by the cylindric constraint system $\mathcal{CS} = (\mathcal{A}_{cc}, \vdash, \mathcal{N}_{cc}, \mathcal{H})$.

Definition 7.7. *The transition relation* $\stackrel{(\sigma,\sigma')}{\to} \subseteq \mathcal{P}_{cc} \times |\mathcal{A}_{cc}| \times \mathcal{P}_{cc} \times |\mathcal{A}_{cc}|$ *over agents in* $\mathcal{P}_{cc}$ *and stores in* $|\mathcal{A}_{cc}|$ *is the least relation satisfying the rules in table 7.3*

The rules are straight-forward. The only rules that require some explanation are the restriction rules. In RESTR$_t^1$, when an agent $\exists x.P$ is evaluated, all constraints in σ about x are hidden from the agent since the x in P is local, and thus, different from that in σ. Also, any information produced by P about the local x has to be hidden from the global store, thus, the information added to the global store is $\dot{\exists}_x\phi$, where ϕ can be seen as a local store for P'. The result is an agent of the form $\exists x.(tell(\phi) \wedge P')$. The rule RESTR$_t^2$ is analogous, taking care of the case when an agent has a local store.

We write $P \xrightarrow{(\sigma,\sigma')}{}^{*} P'$ for the transitive closure of the transition relation, i.e. there is a finite transition sequence from (P, σ) to (P', σ').

7.1.5 Correspondence between the semantics

The correspondence between the semantics lies on the idea that the constraint store can be thought of as another process, just like any agent. We show how the UTS captures the LTS.

Proposition 7.8. *For any $P, P' \in \mathcal{P}_{cc}$, and $\sigma, \sigma' \in |\mathcal{A}_{cc}|$, $P \xrightarrow{(\sigma,\sigma')} P'$ only if $P \wedge tell(\sigma) \rightarrow P' \wedge tell(\sigma')$ where $\sigma' \dashv\vdash \sigma \cup \psi$ for some $\psi \in |\mathcal{A}_{cc}|$.*

Proof. We use induction on the derivation of $P \xrightarrow{(\sigma,\sigma')} P'$.

Case 1: The last inference is an instance of TELL$_t$: $P \equiv tell(\phi)$, $P' \equiv \top$, and $\sigma' \dashv\vdash \sigma \cup \phi$. This is matched by the TELL axiom: $P \wedge tell(\sigma) \equiv tell(\phi) \wedge tell(\sigma) \rightarrow tell(\sigma') \equiv \top \wedge tell(\sigma') \equiv P' \wedge tell(\sigma')$.

Case 2: The last inference is an instance of ASK$_t$: $P \equiv ask(\phi) \rightarrow P'$, $\sigma' \dashv\vdash \sigma$ and $\sigma \vdash \phi$. This is matched by the ASK axiom $P \wedge tell(\sigma) \equiv (ask(\phi) \rightarrow P') \wedge tell(\sigma) \rightarrow P' \wedge tell(\sigma)$.

Case 3: The last inference is an instance of PAR$_t$: $P \equiv Q \wedge R$, and $P' \equiv Q' \wedge R$. So by a shorter inference, $Q \xrightarrow{(\sigma,\sigma')} Q'$. By induction hypothesis we have $Q \wedge tell(\sigma) \rightarrow Q' \wedge tell(\sigma')$. Then by PAR and CONGR we obtain $Q \wedge R \wedge tell(\sigma) \rightarrow Q' \wedge R \wedge tell(\sigma')$ as required.

Case 4: The last inference is an instance of RESTR$_t^1$: $P \equiv \exists x.Q$, $P' \equiv \exists x.(tell(\phi) \wedge Q')$, and $\sigma' \dashv\vdash \sigma \cup \dot{\exists}_x\phi$. Hence, by a shorter inference $Q \xrightarrow{(\dot{\exists}_x\sigma,\phi)} Q'$. So by induction hypothesis $Q \wedge tell(\dot{\exists}_x\sigma) \rightarrow Q' \wedge tell(\phi)$. By RESTR, we have $\exists x.(Q \wedge tell(\dot{\exists}_x\sigma)) \rightarrow \exists x.(Q' \wedge tell(\phi))$. The LHS is congruent to $\exists x.Q \wedge tell(\dot{\exists}_x\sigma)$ by scope extrusion. On the other hand, since $\phi \dashv\vdash \phi \cup \dot{\exists}_x\phi$, we know that $tell(\phi) \equiv tell(\phi) \wedge tell(\dot{\exists}_x\phi)$, so the RHS is congruent to $\exists x.(Q' \wedge tell(\phi) \wedge tell(\dot{\exists}_x\phi))$ which by scope extrusion is congruent to $\exists x.(Q' \wedge tell(\phi)) \wedge tell(\dot{\exists}_x\phi)$. So we have that $\exists x.Q \wedge tell(\dot{\exists}_x\sigma) \rightarrow \exists x.(Q' \wedge tell(\phi)) \wedge tell(\dot{\exists}_x\phi)$, and by applying PAR we have: $\exists x.Q \wedge tell(\dot{\exists}_x\sigma) \wedge tell(\sigma) \rightarrow \exists x.(Q' \wedge tell(\phi)) \wedge tell(\dot{\exists}_x\phi) \wedge tell(\sigma)$. Now, noticing that $\dot{\exists}_x\sigma \cup \sigma \vdash\!\dashv \sigma$, we apply CONGR and obtain $\exists x.Q \wedge tell(\sigma) \rightarrow \exists x.(Q' \wedge tell(\phi)) \wedge tell(\dot{\exists}_x\phi \cup \sigma)$ as required.

The case of RESTR$_t^2$ is similar to case 4. The remaining cases, for SUM, ID, and CONGR mimic the case for PAR. □

7.1.6 Bisimilarity

The definition of bisimilarity for CCP ([22]) is essentially standard but takes into account the role of the stores. The labels of transitions (pairs of stores) need not be literally the same for matching moves; we can relax this to allow labels to match each other when the stores in the labels are logically equivalent.

Definition 7.9. *A binary relation $\mathcal{S} \subseteq (\mathcal{P}_{cc} \times |\mathcal{A}_{cc}|) \times (\mathcal{P}_{cc} \times |\mathcal{A}_{cc}|)$ is called a (weak)**bisimulation** iff for any terms $P, Q \in \mathcal{P}_{cc}$ and $\sigma, \psi \in |\mathcal{A}_{cc}|$, $(P,\sigma)\mathcal{S}(Q,\psi)$ implies that*

(i) Whenever $P \xrightarrow{(\sigma',\sigma'')} P'$ and $\sigma' \vdash\!\dashv \sigma \cup \phi$ where ϕ has only global variables, then $Q \xrightarrow{(\psi',\psi'')}{}^{*} Q'$, $\psi' \vdash\!\dashv \psi \cup \phi$, $\psi' \vdash\!\dashv \sigma'$, $\psi'' \vdash\!\dashv \sigma''$ and $(P',\sigma'')\mathcal{S}(Q',\psi'')$

(ii) vice-versa

*We say that (P,σ) and (Q,ψ) are **bisimilar**, written $(P,\sigma)\dot{\approx}_{cc}(Q,\psi)$ iff there is a bisimulation $\mathcal{S}$ such that $(P,\sigma)\mathcal{S}(Q,\psi)$. We write $P\dot{\approx}_{cc}Q$ for $(P,\mathsf{true})\dot{\approx}_{cc}(Q,\mathsf{true})$.*

Notice that by unwinding the definition for the process bisimilarity $P \dot{\approx}_{cc} Q$ as $(P,\mathsf{true})\dot{\approx}_{cc}(Q,\mathsf{true})$ we have that if $P \xrightarrow{(\phi,\sigma'')} P'$ where ϕ has only global variables, then $Q \xrightarrow{(\phi,\psi'')}{}^{*} Q'$, $\psi'' \vdash\!\dashv \sigma''$ and $(P',\sigma'')\mathcal{S}(Q',\psi'')$. Since this is defined for any ϕ that has only global variables, we obtain a notion in which P and Q match each other's moves, not only in the empty store, but in any store ϕ with only global variables. However the continuations (P' and Q') need not match each other in all such stores, but in the resulting stores σ'' and ψ''.

This notion of bisimilarity preserves all operators, and thus is an appropriate notion of process equivalence.

Proposition 7.10. *$\dot{\approx}_{cc}$ is a CC-process congruence.*

Proof. Assume that $\phi \in |\mathcal{A}_{cc}|$ is a (possibly empty) formula involving only global variables.

Tell: Define $\mathcal{S} \stackrel{def}{=} \{((tell(\varphi_1),\sigma),(tell(\varphi_2),\psi)) \ : \ \varphi_1 \vdash\!\dashv \varphi_2 \text{ and } \sigma \vdash\!\dashv \psi\}$. We show that $\mathcal{S}$ is a bisimulation. The only move a tell can do is $tell(\varphi_1) \xrightarrow{(\sigma',\sigma'')} \top$, where $\sigma' \vdash\!\dashv \sigma \cup \phi$. By definition of TELL$_t$, $\sigma'' \vdash\!\dashv \sigma' \cup \varphi_1$. This move is matched by $tell(\varphi_2) \xrightarrow{(\psi',\psi'')} \top$, where $\psi' \vdash\!\dashv \psi \cup \phi$. Since $\sigma \vdash\!\dashv \psi$ we have $\sigma' \vdash\!\dashv \psi'$, and since $\varphi_1 \vdash\!\dashv \varphi_2$ we have $\sigma'' \vdash\!\dashv \psi''$. Having established this, it is clear that $((tell(\varphi_1), \mathsf{true}), (tell(\varphi_2), \mathsf{true})) \in \mathcal{S}$, so $tell(\varphi_1) \dot{\approx}_{cc}\, tell(\varphi_2)$.

Ask: Let $\mathcal{S} \stackrel{def}{=} \{((ask(\varphi_1) \to P, \sigma), (ask(\varphi_2) \to Q, \psi)) \ : \ \varphi_1 \vdash\!\dashv \varphi_2,\ P \dot{\approx}_{cc} Q$ and $\sigma \vdash\!\dashv \psi\}$. We show that $\mathcal{S}$ is a bisimulation. Suppose that the first element of the pair makes the move $ask(\varphi_1) \to P \xrightarrow{(\sigma',\sigma')} P$ where $\sigma' \vdash\!\dashv \sigma \cup \phi$, and $\sigma' \vdash \varphi_1$. Since $\sigma \vdash\!\dashv \psi$, $\sigma' \vdash\!\dashv \psi'$ where $\psi' \vdash\!\dashv \psi \cup \phi$. Hence, since $\varphi_1 \vdash\!\dashv \varphi_2$, we have that $\psi' \vdash \varphi_2$, so the transition can be matched by $ask(\varphi_2) \to Q \xrightarrow{(\psi',\psi')} Q$ with $P \dot{\approx}_{cc} Q$. So $\mathcal{S}$ is a bisimulation. Clearly $((ask(\varphi_1) \to P, \mathsf{true}), (ask(\varphi_2) \to Q, \mathsf{true})) \in \mathcal{S}$, so $ask(\varphi_1) \to P \dot{\approx}_{cc}\, ask(\varphi_2) \to Q$.

Restriction: Let $\mathcal{S} \stackrel{def}{=} \{((\exists x.(tell(\varphi_1) \wedge P), \sigma), (\exists x.(tell(\varphi_2) \wedge Q), \psi)) \ : \ (P, \varphi_1 \cup \dot{\exists}_x \sigma) \dot{\approx}_{cc} (Q, \varphi_2 \cup \dot{\exists}_x \psi), \varphi_1 \vdash\!\dashv \varphi_2,$ and $\sigma \vdash\!\dashv \psi\}$. We first show that $\mathcal{S}$ is a bisimulation. Suppose that the left element of the pair makes the move $\exists x.(tell(\varphi_1) \wedge P) \xrightarrow{(\sigma',\sigma' \cup \dot{\exists}_x \varphi_1')} \exists x.(tell(\varphi_1') \wedge P')$ where as usual $\sigma' \vdash\!\dashv \sigma \cup \phi$. By a shorter inference we know that $P \xrightarrow{(\varphi_1 \cup \dot{\exists}_x \sigma', \varphi_1')} P'$. Since $(P, \varphi_1 \cup \dot{\exists}_x \sigma) \dot{\approx}_{cc} (Q, \varphi_2 \cup \dot{\exists}_x \psi)$, this move is matched by $Q \xrightarrow{(\varphi_2 \cup \dot{\exists}_x \sigma', \varphi_2')} Q'$ where $(P', \varphi_1') \dot{\approx}_{cc} (Q', \varphi_2')$ with $\varphi_1' \vdash\!\dashv \varphi_2'$ and therefore $\dot{\exists}_x \varphi_1' \vdash\!\dashv \dot{\exists}_x \varphi_2'$. Then, by RESTR$_t^2$ we have that $\exists x.(tell(\varphi_2) \wedge Q) \xrightarrow{(\sigma',\sigma' \cup \dot{\exists}_x \varphi_2')} \exists x.(tell(\varphi_2') \wedge Q')$. Notice that since $(P', \varphi_1') \dot{\approx}_{cc} (Q', \varphi_2')$ we also have $(P', \varphi_1' \cup \dot{\exists}_x \sigma'') \dot{\approx}_{cc} (Q', \varphi_2' \cup \dot{\exists}_x \psi'')$ for any σ'' and ψ'' such that $\sigma'' \vdash\!\dashv \psi''$. Therefore we have that $((\exists x.(tell(\varphi_1') \wedge P'), \sigma''), (\exists x.(tell(\varphi_2') \wedge Q'), \psi'')) \in \mathcal{S}$ as required. It is easy to check that $((\exists x.(tell(\varphi_1) \wedge P), \mathsf{true}), (\exists x.(tell(\varphi_2) \wedge Q), \mathsf{true})) \in \mathcal{S}$ when $\varphi_1 \vdash\!\dashv \varphi_2$, and $(P, \varphi_1) \dot{\approx}_{cc} (Q, \varphi_2)$, which implies that if $(P, \mathsf{true}) \dot{\approx}_{cc} (Q, \mathsf{true})$ then $(\exists x.P, \mathsf{true}) \dot{\approx}_{cc} (\exists x.Q, \mathsf{true})$ as required.

Parallel: Let $\mathcal{S} \stackrel{def}{=} \{((P \wedge R, \sigma), (Q \wedge R)) \ : \ \sigma \vdash\!\dashv \psi$ and $(P, \sigma) \dot{\approx}_{cc} (Q, \psi)\}$. Suppose that $P \wedge R \xrightarrow{(\sigma',\sigma'')} P' \wedge R'$ where $\sigma' \vdash\!\dashv \sigma \cup \phi$. There are two possibilities depending on which agent performed the transition:

Case 1: $R' \equiv R$ and $P \xrightarrow{(\sigma',\sigma'')} P'$. Since $(P, \sigma) \dot{\approx}_{cc} (Q, \psi)$ then $Q \xrightarrow{(\psi',\psi'')} Q'$ where $(P', \sigma'') \dot{\approx}_{cc} (Q', \psi'')$, $\psi' \vdash\!\dashv \psi \cup \phi$, and $\sigma'' \vdash\!\dashv \psi''$. By PAR$_t$ we have $Q \wedge R \xrightarrow{(\psi',\psi'')} Q' \wedge R$, so $((P' \wedge R, \sigma''), (Q' \wedge R, \psi'')) \in \mathcal{S}$ as required.

Case 2: $P' \equiv P$ and $R \xrightarrow{(\sigma',\sigma'')} R'$. Since $\sigma'' \vdash\!\dashv \sigma \cup \phi \cup \tau$, where τ was introduced by R, and since $(P, \sigma) \dot{\approx}_{cc} (Q, \psi)$ then $(P, \sigma'') \dot{\approx}_{cc} (Q, \psi'')$ where $\psi'' \vdash\!\dashv \sigma''$. Therefore $((P \wedge R', \sigma''), (Q \wedge R', \psi'')) \in \mathcal{S}$ as required.

The rest of the cases are similar. ∎

Bisimilarity satisfies some interesting properties. In particular, we like to stress the close relation between diagonal elements in the constraint system, with substitution, equators and fusions. This is stated by the following:

Proposition 7.11 (([22])). *For any names a, b, c the following hold:*

(i) $tell(a = a) \ \approx_{cc}\ \exists b.(tell(a = b)) \ \approx_{cc}\ \top$

(ii) $\exists c.(tell(a = c) \wedge tell(c = b)) \ \approx_{cc}\ tell(a = b)$

(iii) $P\{a/b\} \ \approx_{cc}\ \exists b.(tell(a = b) \wedge P)$

7.2 Expressiveness

CCP is a powerful paradigm, however, it presents a very different perspective to that of mobile process calculi. It is natural to ask what is the relationship between these two paradigms for concurrent computation. In particular we would like to know whether mobile process calculi can capture the expressive power of CCP, and conversely whether CCP is apt for modelling mobile processes. The first question was answered positively by Victor and Parrow in [46], where they provided an encoding of an extension of CCP known as the γ-calculus([44]), into the π-calculus. The converse however has remained an open issue. Some people have argued that CCP is powerful enough to simulate mobility (see [21] for example), but these approaches are all based on extensions of CCP, not on the core language. We are interested to know if the core language is enough. The main contribution of this thesis is that this is not the case: the core CCP is not mobile.

7.2.1 From CCP to π

We will now consider a particular CC language in which the constraint system consists of name equations. We call this language CCP(x=y). We provide an encoding of this language into the π-calculus, which is a simplification of the encoding given by Victor and Parrow in [46]. We do not present the full encoding since theirs is a translation from the γ-calculus, which is an extension of the basic CCP model, and at this moment we are not concerned with the extended language but just with the core.

The translation

The key of the embedding is that CCP names are encoded not as π-names but as agents, called *handlers*. It is useful to think of these handlers as objects (in the OOP sense) which have three operations on them: 1) returning its value, i.e. its own port of access, a π-name; 2) updating its value to point to a new handler; and 3) checking for equality with another handler. The idea then is that when we tell the constraint $x = y$ this is simulated by updating the handler of x to "point" to the handler of y. When we ask if an equation is entailed, roughly speaking, we "call" the checking method of one of the names with the other as parameter.

There are two kinds of handler, denoted $V(\underline{x})$ and $R(\underline{x}, \underline{y})$. A variable x that has been updated to a reference y, is handled by a *relay* handler $R(\underline{x}, \underline{y})$; otherwise it is handled by $V(\underline{x})$. A relay handler, will simply relay the message it receives through $\underline{x}$, to $\underline{y}$. When a variable handled by $V(\underline{x})$ is queried for its value, it returns its port x. When it is updated to y, then it changes its state and becomes handled by $R(\underline{x}, \underline{y})$. When queried for equality with z, it will check if the references are identical; if they are it answers yes; if they are not it will do a busy-wait loop until it receives the same reference (its own port), while still accepting other requests.

Notice that in a relay agent $R(\underline{x}, \underline{z})$, the reference z can itself point to another relay agent, say $R(\underline{z}, \underline{w})$. This means that the set of handlers forms a forest, and variables that are equal are handled by agents in the same tree. We call such trees *equivalence trees.*

For each CCP name x we introduce the *handler interface* $\underline{x}$ which is defined as $\underline{x} \stackrel{def}{=} x, x_{\mathsf{val}}, x_{\mathsf{upd}}, x_{\mathsf{eq}}$. These can be thought of as the pointer to the handler, and its method names in OOP terminology. These are explained as follows:

- The plain x is the main reference to the handler for the name. This is the reference checked by equality.

- The name x_{val} is the port where the handler receives queries for its value. Through this channel the handler receives two names s and r, which are the identity of the source of the query (s) and the channel through which the answer will sent (r).

- The name x_{upd} is the port where the handler receives a request for updating. Through this channel the new reference $\underline{z}$ is received by the handler.

- The name x_{eq} is the port where the handler is queried for equality with another variable. It receives the reference to the other variable $\underline{u}$, and a channel y. If the two variables are in the same equivalence tree, a signal is sent through y. [6]

We now define the encoding itself. Recall that we are considering only name equations, so the assertions in the constraint system are of the form $u = v$. We call $\mathcal{P}_{cc}^{=}$ the subset of CC terms with only name equations as constraints. The following

[6]Notice that the negative case is not handled. The original encoding did handle it, because there was an explicit separation between names and variables, so it was possible to decide when two names were different. Here, since we do not make such separation, such a query is not handled. If the variables are different, then equality is not entailed so the agent asking for equality will be blocked waiting for a signal in channel y.

$$
\begin{aligned}
V(\underline{x}) \stackrel{def}{=}\ & x_{\mathsf{val}}(r,s).\overline{r}\langle\underline{x}\rangle.V(\underline{x}) \\
& + \ x_{\mathsf{upd}}(\underline{u}). \\
& \quad ([x=u]V(\underline{x}) \\
& \quad\ \ + \ [x \neq u]\nu r.\overline{u_{\mathsf{val}}}\langle r,x\rangle.r(\underline{w}).([x=w]V(\underline{x}) \ + \ [x \neq w]R(\underline{x},\underline{w}))) \\
& + \ x_{\mathsf{eq}}(\underline{u},y). \\
& \quad (V(\underline{x}) \mid \nu r.\overline{u_{\mathsf{val}}}\langle r,x\rangle.r(\underline{w}).([x=w]\overline{y} \ + \ [x \neq w]\overline{x_{\mathsf{eq}}}\langle\underline{w},y\rangle)) \\
R(\underline{x},\underline{u}) \stackrel{def}{=}\ & !\, x_{\mathsf{val}}(r,s).([u=s]\overline{r}\langle\underline{s}\rangle \ + \ [u \neq s]\overline{u_{\mathsf{val}}}\langle r,s\rangle) \\
& \mid\ !\, x_{\mathsf{upd}}(\underline{w}).\overline{u_{\mathsf{upd}}}\langle\underline{w}\rangle \\
& \mid\ !\, x_{\mathsf{eq}}(\underline{w},y).\overline{u_{\mathsf{eq}}}\langle\underline{w},y\rangle
\end{aligned}
$$

Table 7.4: Variable handlers

translation is *compositional* i.e. the translation of a term depends only on the translation of its sub-terms. It is also *uniform* since the parallel composition of CC terms is simulated by the parallel composition of the respective translations. This means that the translation is truly distributed.

Definition 7.12 (CCP to π translation).
The translation $[\![\cdot]\!] : \mathcal{P}_{cc}^{=} \rightarrow \mathcal{P}_{\pi}$ is given by the following equations:

$$
\begin{aligned}
[\![\top]\!] &\stackrel{def}{=} 0 \\
[\![P \wedge Q]\!] &\stackrel{def}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![P \ + \ Q]\!] &\stackrel{def}{=} [\![P]\!] \ + \ [\![Q]\!] \\
[\![\exists x.P]\!] &\stackrel{def}{=} \nu\underline{x}.(V(\underline{x}) \mid [\![P]\!]) \\
[\![tell(u=v)]\!] &\stackrel{def}{=} \overline{u_{\mathsf{upd}}}\langle\underline{v}\rangle \\
[\![ask(u=v) \rightarrow P]\!] &\stackrel{def}{=} \nu y.(\overline{u_{\mathsf{eq}}}\langle\underline{v},y\rangle \mid y.[\![P]\!])
\end{aligned}
$$

where $V(\underline{x})$ and $R(\underline{x},\underline{u})$ are defined in table 7.4.

It should be clear how this translation works. The relay handler for x, $R(\underline{x},\underline{u})$, as explained before, simply relays the messages it receives to the handler for u. When a variable handler is asked to be updated it first checks if the new reference is already itself, in which case it continues in the same state $V(\underline{x})$. Otherwise, it obtains the value of the new reference, checks again in case the other variable has been updated in the meantime, and if they are still different, it creates the relay agent. When testing for equality it creates a copy of itself to keep accepting requests, while asking the other variable for its value and comparing references.

Correctness

This translation is correct in the sense that it is fully abstract with respect to weak barbed congruence. Recall from section 3.2 that the notion of barbed bisimulation was defined for the UTS of π calculus with an additional "observation" predicate. We can define such a predicate in the context of CCP(x=y), and thus inducing a notion of weak barbed bisimilarity and weak barbed congruence for CCP.

Definition 7.13 (Observability in CCP(x=y) ([46])). *The predicate* $\downarrow \subseteq \mathcal{P}_{cc}^{=} \times \mathcal{N}_{cc}$ *is defined inductively as follows:*

(i) $\exists y.P \downarrow x$ *if* $x \neq y$ *and* $P \downarrow x$

(ii) $(P \wedge Q) \downarrow x$ *if* $P \downarrow x$ *or* $Q \downarrow x$

(iii) $(P + Q) \downarrow x$ *if* $P \downarrow x$ *or* $Q \downarrow x$

We say that an agent P *is* ***observable*** *at a name* x *if* $P \downarrow x$*.*

This notion induces the definition of weak barbed bisimilarity ($\dot{\approx}_{cc}^{b}$) and weak barbed congruence ($\approx_{cc}^{b}$) in the same way as was done in section 3.2. As in π calculi, processes $P \wedge Q$ and $P + Q$ observe the same names, but they are not barbed bisimilar since they do not necessarily match each other's reductions. Take for instance $R_1 \stackrel{def}{=} tell(\varphi) \wedge ask(\varphi) \to P$, and $R_2 \stackrel{def}{=} tell(\varphi) + ask(\varphi) \to P$. They are certainly not barbed bisimilar because $R_1 \to P$ while R_2 cannot match this move unless it is in a particular context.

The correctness of the translation is established by the following theorem.

Theorem 7.14 (Victor and Parrow [46]).
The translation $[\![\cdot]\!]$ *of definition 7.12 is fully-abstract with respect to weak barbed congruence, i.e. for any terms* $P, Q \in \mathcal{P}_{cc}^{=}$*,*

$$P \approx_{cc}^{b} Q \text{ if and only if } [\![P]\!] \approx_{\pi}^{b} [\![Q]\!]$$

Proof. We only provide a proof sketch, and the reader is referred to [46] for a more detailed account. The strategy for the proof follows the same line as was shown in theorem 2.10. That proof was in relation to an LTS. In this case we are dealing with a UTS and an observability predicate. The strategy in this case is the following: first show that the translation preserves and reflects both (unlabelled) reductions *and* observations; and second, show that preservation and reflection of these two relations implies full-abstraction. □

7.2.2 From π to CCP

Here we provide the main result of this work. We prove that the π-calculus is more powerful than the core CCP. Our argument is based on the impossibility in CCP to establish private-shared channels between processes. This is, if two processes want to create a common link, it will necessarily be accessible to every other agent, and hence it will not be private. In other words, internal mobility cannot be simulated in CCP. In the rest of this section, we will show this gap between CCP and π_I, which as we have seen, is a sub-calculus of π.

The main effect of communication in π-calculi is substitution of names. In CCP, the only way to simulate substitution is through diagonal elements, i.e. name equations. We argue that these equations will force the "domain of influence" of the names involved to grow from their static, syntactic scope to the global scope, becoming accessible to any process listening.

Observability

We start by defining when are names accessible to a process.

Definition 7.15 (Observability in CC). *We say that a CC term P observes the name x under the constraint store σ, written $(P, \sigma) \downarrow_{cc} x$ iff there is a $y \in fn(P)$ such that $\sigma \vdash x = y$.*

With this definition it holds that a process observes its own free names under any store. Also, that in the empty store P observes x if and only if x is free in P.

An analogous definition can be made in π_I. This definition of observability is weaker than the previous definitions discussed in the context of barbed bisimulation for π-calculi. We only require for a name to be free in a process in order to observe it. That is, an accessible, or observable name is one through which the process could potentially interact with the environment, but this interaction might not be immediately possible, as required in the definitions of π-observability considered before.

Definition 7.16 (Observability in π_I). *We say that a π term P observes the name x written $P \downarrow_{\pi I} x$ iff $x \in fn(P)$.*

As usual, we will ignore subscripts whenever it is clear from the context to which concept we are referring.

Domain of influence

The following property represents the notion that a name is either accessible by only one process in a system, or accessible by all the processes of the system.

Definition 7.17. *Given a CC term $M \equiv \exists \vec{u}.(P \wedge Q \wedge R)$, a store σ, and a name $u \in \vec{u}$ we say that u has **1-3 domain of influence** in M under σ if it is observable by either exactly one or exactly three of P, Q and R under the store σ.*

In the following examples of names x has 1-3 domain of influence in M under σ where M and σ are:

1. $\sigma \equiv \mathsf{true}$, $P \equiv \mathit{tell}(x \geq 1)$, $Q \equiv \mathit{tell}(x \leq 5)$, $R \equiv \mathit{tell}(x \geq 2)$
2. $\sigma \equiv \mathsf{true}$, $P \equiv \mathit{tell}(x \geq 1)$, $Q \equiv \mathit{tell}(x \leq 5)$, $R \equiv \mathit{tell}(x = u) \wedge \mathit{tell}(u \geq 2)$
3. $\sigma \equiv x = y$, $P \equiv \mathit{tell}(x \geq 1)$, $Q \equiv \mathit{tell}(x \leq 5)$, $R \equiv \mathit{tell}(y \geq 2)$

An example of a term and store not satisfying the property is:

$$\sigma \equiv \mathsf{true},\ P \equiv \mathit{tell}(x \geq 1),\ Q \equiv \mathit{tell}(x \leq 5),\ R \equiv \mathit{tell}(y \geq 2)$$

The corresponding notion in π_I is analogous, noting that there is no reference to a constraint store.

Definition 7.18. *Given a π_I term $M \equiv \nu\vec{u}.(P \mid Q \mid R)$ and a name $u \in \vec{u}$ we say that u has **1-3 domain of influence** in M if it is observable by either exactly one or exactly three of P, Q and R.*

Eavesdropping

The most significant notion of π-calculi is mobility, and this allows for the dynamic evolution of the scope or domain of influence of names. The 1-3 domain of influence property above, is a static property, i.e. it considers a "snapshot" of the system. We are interested in finding out whether CCP allows mobility. This is of course, related to how the domain of influence of names evolve, and therefore we must focus our attention in some dynamic property. In particular we are interested in what happens when a process communicates a private name to another process. If we try to model internal mobility, we need to establish a secure, private channel between two processes, so that no third party is able to listen or "eavesdrop" trough that channel. We now formalize what do we mean by "eavesdropping" with the following property, which states that names in a system that are accessible by only one agent, will either remain private or become accessible by all, i.e. the 1-3 domain of influence property is invariant.

Definition 7.19 (Eavesdropping in CC). *Consider a system $M \equiv \exists\vec{u}.(P \wedge Q \wedge R)$ in standard form, and a store σ. We say that M satisfies the "CC-eavesdropping" property under σ if whenever the following are satisfied*

(i) *All names in $\vec{u}$ have 1-3 domain of influence in M under σ,*

(ii) $(M, \sigma) \rightarrow (M', \sigma')$, *and*

(iii) *No names in $\vec{u}$ disappear from any of P, Q, and R*

then all names in $\vec{u}$ have 1-3 domain of influence in M' under σ'

Now the gap in expressiveness between CCP and π_I becomes apparent. The following theorem states that it is not possible to create one private channel between two of the processes, while keeping all other names public or restricted to one process.

Theorem 7.20. *All CC terms satisfy the eavesdropping property under any store.*

Proof. Assume that the conditions are satisfied but there is a name $x \in \vec{u}$ such that it does not have 1-3 domain of influence in M' under σ', i.e. it becomes observable by only two of the processes in M'. Assume, without loss of generality that P' and Q' are the processes observing x under σ'. Given that x has 1-3 domain of influence in M under σ, we have two cases:

1. $(P, \sigma) \downarrow x$, $(Q, \sigma) \downarrow x$, and $(R, \sigma) \downarrow x$, or
2. $(P, \sigma) \downarrow x$, $(Q, \sigma) \not\downarrow x$, and $(R, \sigma) \not\downarrow x$

In case 1, given that names do not disappear, then we have that x is observable by P', Q' and R' under σ', so we obtain a contradiction.

In case 2, since $(Q', \sigma') \downarrow x$, then there is a $y \in fn(Q')$ (and also $y \in fn(Q)$) such that $\sigma' \vdash x = y$. However this is only possible if there is a $z \in fn(Q)$ such that $\sigma' \vdash x = z \wedge z = y$, because Q could not observe x. Hence there is an action $tell(x = z)$ in P, because it was the only agent observing x. So both P and Q observe z. Since z has 1-3 domain of influence in M under σ (by hypothesis) then R also observes z under σ. Therefore R' observes z under σ' because names do not disappear, and since $\sigma' \vdash x = z$, we have that R' also observes x under σ', again yielding a contradiction. □

The eavesdropping property can be stated for π_I terms.

Definition 7.21. (Eavesdropping in π_I) *Consider a π_I system $M \equiv \nu\vec{u}.(P \mid Q \mid R)$ in standard form. We say that M satisfies the "π_I-eavesdropping" property if whenever the following are satisfied*

1. *All names in $\vec{u}$ have 1-3 domain of influence in M,*
2. *$M \to M'$, and*
3. *No names in $\vec{u}$ disappear from any of P, Q, and R*

then all names in $\vec{u}$ have 1-3 domain of influence in M'

Clearly there are π_I terms that do not satisfy this property, and that is precisely the gap in expressiveness. Consider the following term:

$$\begin{aligned} M_0 &\equiv \nu u.(P \mid Q \mid R) \qquad \text{where} \\ P &\equiv \overline{u}(x).P' \\ Q &\equiv u(y).Q' \\ R &\equiv \overline{u}(z).R' \end{aligned}$$

Such that $u \in fn(P') \cap fn(Q')$. This term is equivalent, by alpha conversion and scope extrusion to:

$$\begin{aligned} M &\equiv \nu ux.(P \mid Q \mid R) \qquad \text{where} \\ P &\equiv \overline{u}\langle x\rangle.P' \\ Q &\equiv u(y).Q' \\ R &\equiv \overline{u}\langle z\rangle.R' \end{aligned}$$

Where $x \notin fn(Q') \cup fn(R')$[7]. So both x and u have 1-3 domain of influence in M, i.e. they are not shared exclusively by two processes. This term reduces to M' where

$$M' \equiv \nu ux.(P' \mid Q'\{x/y\} \mid R) \qquad \text{where } x \notin fn(R)$$

Clearly M_0 does not satisfy the eavesdropping property. To see this note that: 1) u is free in the three agents, thus having 1-3 domain of influence in M'; 2) x is observable only by P in the initial state; 3) in the final state x is observable by both P' and $Q'\{x/y\}$ but it is still not observable by R.

Non-encodability

The dual notion of eavesdropping is security.

Definition 7.22 (CC security). *Let $M \equiv \exists\vec{u}.(P \wedge Q \wedge R)$ be in standard form, and a store σ. We say that M is **2-3 secure** under σ if it does not satisfy the CC-eavesdropping property under σ.*

By theorem 7.20 we have that there is no CC term that is 2-3 secure.

The equivalent notion for π_I terms is as follows:

Definition 7.23 (π_I security). *Let $M \equiv \nu\vec{u}.(P \mid Q \mid R)$ be in standard form. We say that M is **2-3 secure** if it does not satisfy the π_I-eavesdropping property.*

The following definition allow us to link the eaves-dropping property for CC terms and π_I terms.

Definition 7.24 (Security preservation). *Let $[\![\cdot]\!]$ be a translation from π_I to CC. We say that $[\![\cdot]\!]$ **preserves security** iff for all π_I terms M, if M is 2-3 secure then $[\![M]\!]$ is 2-3 secure under the empty store.*

From this definition and theorem 7.20 we obtain the following result:

Corollary 7.25. *There is no translation from π_I to CC that preserves security.*

[7] Strictly speaking these are not π_I terms since they involve free output, yet, since they where obtained by alpha conversion and scope extrusion, their semantics remains the same, i.e. $M \sim_\pi M_0$

Proof. Suppose that there is a translation $[\![\cdot]\!]$ that preserves security. Consider the term M_0 from the example, it is easy to see that M_0 is 2-3 secure. On the other hand we have that $[\![M]\!]$ is also 2-3 secure under the empty store because $[\![\cdot]\!]$ preserves security, but this is a contradiction, since by theorem 7.20 we know that all CC terms satisfy the CC-eavesdropping property, and therefore no CC term is 2-3 secure for any store. ☐

7.2.3 CCP and fusions

Since our previous result showed that CCP cannot model internal mobility, the immediate corollary is that any calculus that includes π_I as sub-calculus is more expressive than CCP. This includes the Fusion and χ calculi. However there seems to be some close relation between those calculi and CCP(x=y), since in both, name equations play a central role. In both, they have as main effect, the substitution of equivalent names, and in both, this effect is global. The gap in expressiveness relies on the fact that in Fusion it is possible to restrict such effects through the hiding operator, whereas in CCP, hiding is unable to "contain" the effect of an equation.

It is easy to see that CCP(x=y) can be encoded in Fusion. Victor and Parrow provide such embedding in [47]. The original encoding considered γ calculus with explicit separation of constants and variables. Here we provide a simplification of such translation.

Theorem 7.26 (CCP(x=y) to Fusion translation). *The translation $[\![\cdot]\!]$ from CCP(x=y) to Fusion is defined inductively as follows:*

$$
\begin{aligned}
[\![\top]\!] &\stackrel{def}{=} 0 \\
[\![P \wedge Q]\!] &\stackrel{def}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![P + Q]\!] &\stackrel{def}{=} [\![P]\!] + [\![Q]\!] \\
[\![\exists x.P]\!] &\stackrel{def}{=} \nu x.[\![P]\!] \\
[\![tell(u = v)]\!] &\stackrel{def}{=} \{u = v\} \\
[\![ask(u = v) \rightarrow P]\!] &\stackrel{def}{=} [u = v][\![P]\!]
\end{aligned}
$$

Note in particular that diagonal elements are translated into fusions, and the ask operator is mapped to the match operator. We omit the proof, but we show how the basic interaction is simulated.

$$
\begin{array}{lll}
& [\![\exists y.(tell(x=y) \wedge ask(x=y) \rightarrow P)]\!] & \\
\stackrel{def}{=} & \nu y.(\{x=y\} \mid [x=y][\![P]\!]) & \text{by def 7.26} \\
\rightarrow & [x=x]([\![P]\!]\{x/y\}) & \text{by COMM}_F \\
\rightarrow & [\![P]\!]\{x/y\} & \text{by MATCH}_F \\
\equiv & [\![P\{x/y\}]\!] & \text{as required}
\end{array}
$$

With this simple translation it becomes apparent the expressivity of Fusion, which easily embeds both π and CCP(x=y) in a simple, symmetric framework.

7.2.4 The γ and ρ calculi

For completeness we end this chapter with a brief mention of two variants of CCP, known as the γ-calculus and the ρ calculus.

The γ calculus, developed by Smolka ([44]) is essentially an extension of CCP(x=y) with first-order functional abstraction (i.e. only names can be passed as arguments) and explicit cells. Two presentations have been provided, one that distinguishes between variables and constants and another that doesn't.

The γ-calculus was generalised by Niehren, Smolka and Müller to handle any constraint system, not just name equations. This extension is called the ρ calculus ([34]). The γ calculus coincides with the $\rho(x=y)$-calculus. The ρ-calculus without constraints, written $\rho(\emptyset)$ is a subset of the asynchronous π-calculus with the match operator. The full ρ calculus is written $\rho(CS)$, to emphasise that it is parametrized by a constraint system CS, and it contains CCP as a sub-calculus. It is illuminating to see that what in $\rho(\emptyset)$ corresponds to π_a's match operator, in $\rho(x=y)$ it plays the role of an ask operator for testing existence of diagonal elements.

Chapter 8

Conclusions

8.1 Summary

Let us summarize the relationships studied in this thesis. Table 8.1 summarizes the variants of calculi discussed.

Figure 8.2 shows the main relations between variants on the π-calculi, with respect to non-deterministic choice and asynchrony, where the arrow types are shown in figure 8.1. The references for these are given as follows, in historical order:

1. Honda and Tokoro's encoding, found in [17], developed independently by Boudol in [7]. We describe Boudol's encoding in section 4.2.1.

2. Nestmann and Pierce's encoding, found in [33] and [32].

3. Palamidessi's impossibility result. The reference is [36]. We describe this result in section 4.2.2.

4. Nestmann's encoding from [32].

Figure 8.3 shows the main relations with respect to internal mobility and locality. The references are:

1. Boreale's fully abstract encoding of external mobility in the internal fragment, used the local (asynchronous) π calculus, π_{La} as an intermediate language. The π_I^- is a variant of π_I with "symmetric" locality. The reference for this encodings is [6].

2. Merro's fully abstract encoding was given in [24], and is described here in section 5.2.2.

3. This is Palamidessi's result shown in the previous figure.

4. This is the composition of Nestmann's encodings.

Calculus name	Other names	Description
Core π	π_s	It includes the following operators: input and free output prefix, restriction, parallel composition.
Asynchronous π	π_a, Aπ	Output is not a prefix; Does not include choice.
Internal π	π_I, Iπ	Output prefix is binding.
Local π	π_L, Lπ	Object names of input prefix cannot appear as subject of an input action in the body.
π^{inp}	-	Only allows input-guarded choice
π^{out}	-	Only allows output-guarded choice
π^{sep}	-	Only allows separated guarded choice
Full π	π^{mix}	Includes match and mismatch, and mixed-guarded choice
Fusion	χ, π_F, Fπ, CUP	Input is not binding. Interaction "fuses" names.
Update	-	Monadic Fusion.
Asynchronous Fusion	χ_a, π_{Fa}, $AF\pi$	Input is not binding, and output is not prefix.
Solos	-	Input is not binding. Neither input nor output are prefix operators.
CCP	CCP($\mathcal{CS}$)	Includes tell, ask, restriction, parallel composition, choice. It is parametrized by a constraint system $\mathcal{CS}$.
ρ	$\rho(\mathcal{CS})$	CCP plus first-order functional abstraction.
γ	$\rho(x = y)$	It is ρ restricted to name equations.

Table 8.1: Calculi for mobility and calculi for concurrent constraints

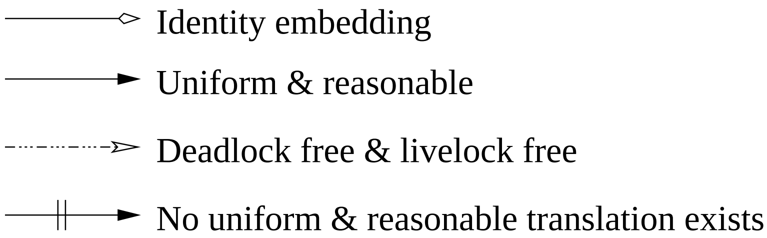

Figure 8.1: Main types of translations between calculi

In figure 8.4 the main expressiveness relations from the point of view of fusion calculi are shown. In this diagram we denote Fusion^{sep} as the subset of Fusion that allows separate-guarded choice, but not mixed-choice.

1. The gap between monadic Fusion or Update calculus and the polyadic calculus is shown in [45].

2. This is Victor and Parrow's encoding of γ-calculus into π. The reference is [46], and is described here in section 7.2.1.

3. Victor and Parrow's encoding of γ in Fusion is found in [47], and described in section 7.2.3.

4. Merro's encoding is found in [23], and described in section 6.3.2.

5. Laneve and Victor's encoding is found in [19] and described in section 6.3.1.

6. Boreale and Merro's encodings, described in the previous figure.

7. This is the impossibility result described in section 7.2.2.

Similarly in figure 8.5 for CCP languages in relation to π-calculi.

1. Victor and Parrow's encoding of γ in Fusion.

2. This is the impossibility result described in section 7.2.2.

3. This is the composition of Laneve and Victor's encoding of Fusion in Solos, and Merro's encoding of Asynchronous Fusion in π_a.

4. Victor and Parrow's encoding of γ-calculus into π.

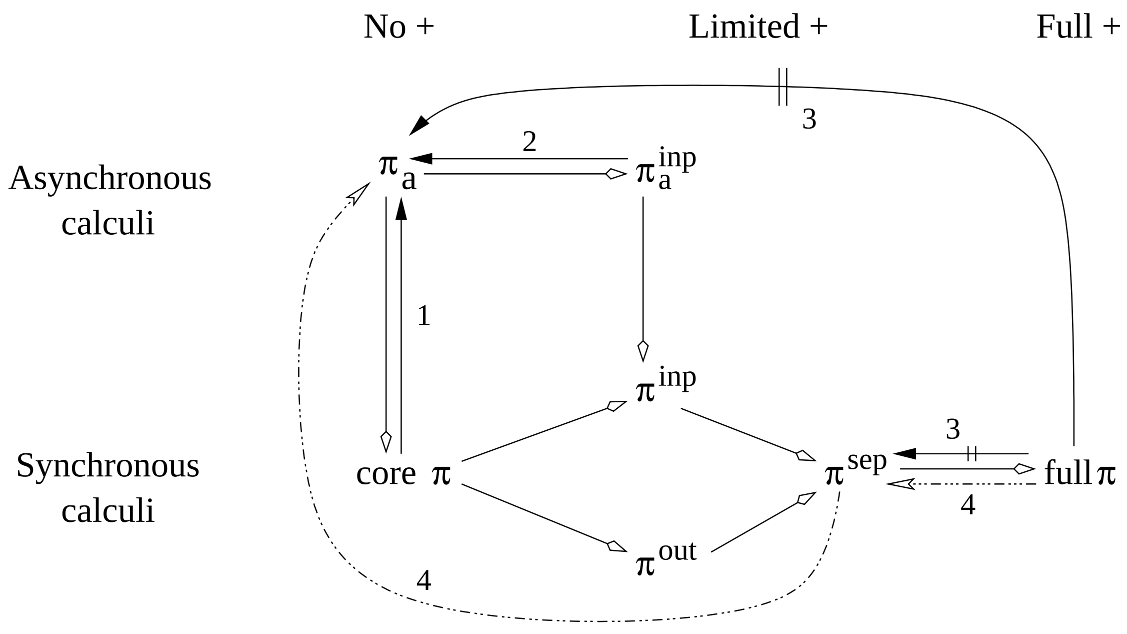

Figure 8.2: Expressiveness relations of asynchrony and choice within π calculi

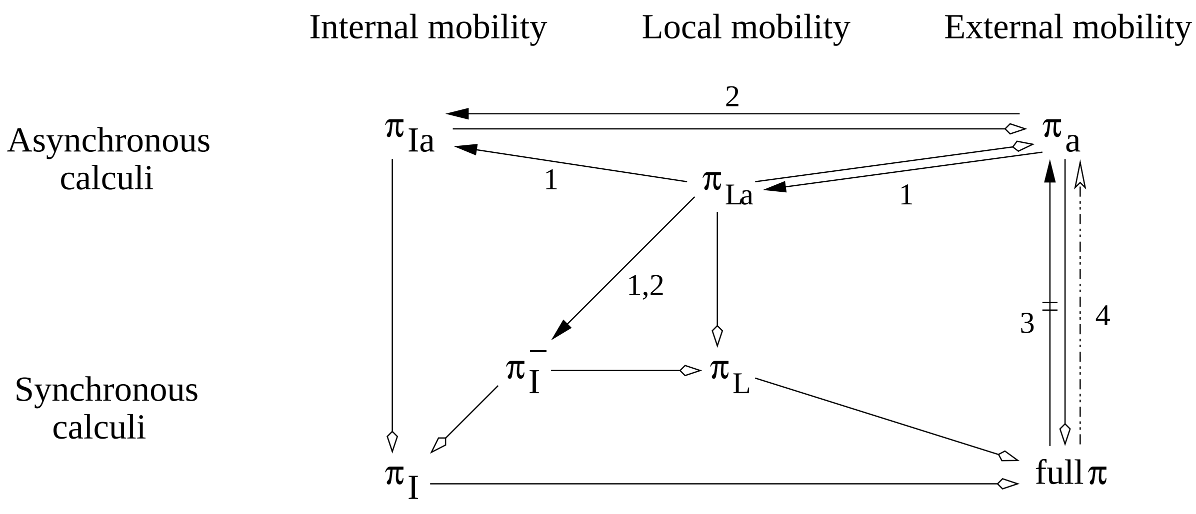

Figure 8.3: Expressiveness relations of internal mobility and locality within π calculi

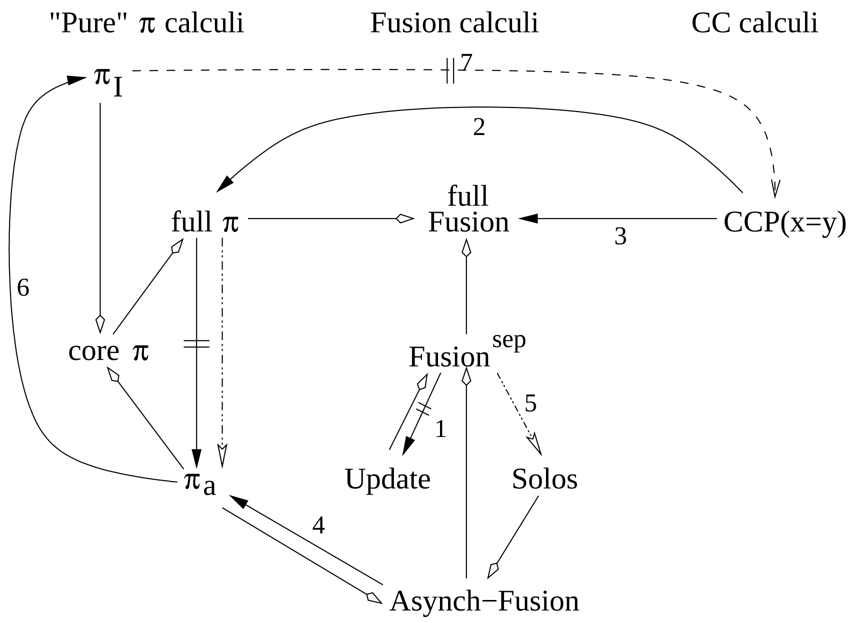

Figure 8.4: Expressiveness relations between Fusion and π calculi

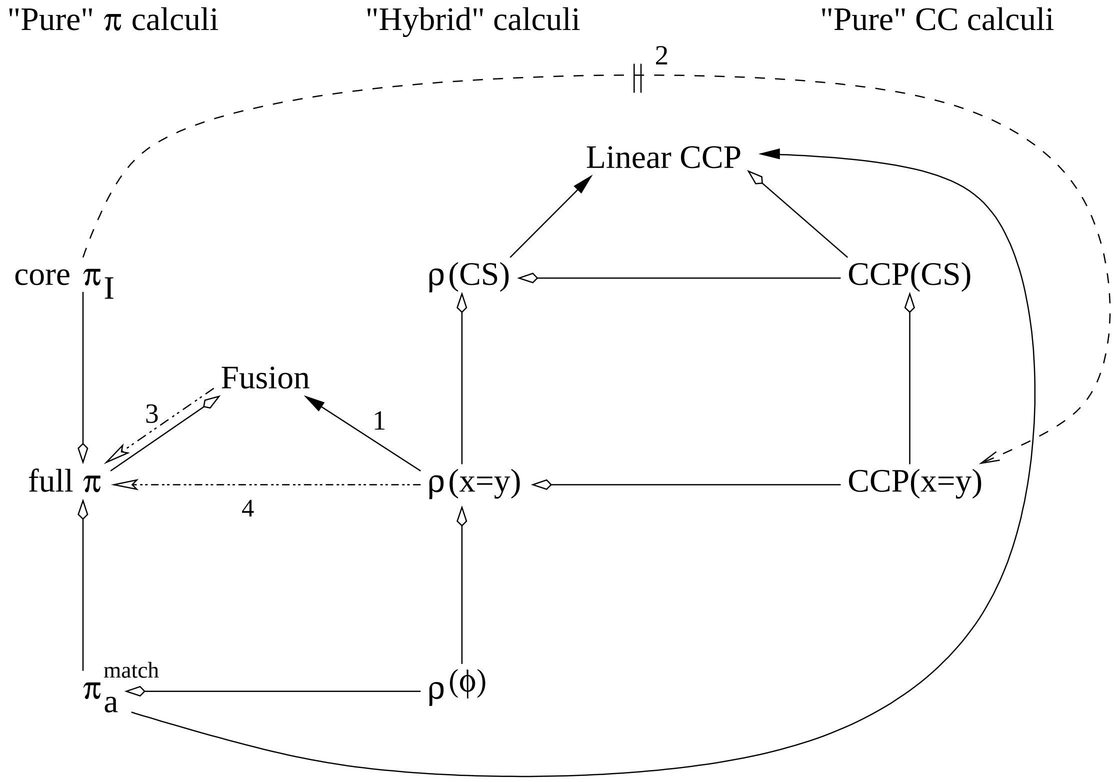

Figure 8.5: Expressiveness relations between CCP and π calculi

8.2 Recent developments and related work

Given the limitations of scope, time and space we have not dealt in this thesis with several relevant aspects to concurrency theory, such as axiomatisations of the different notions of bisimilarity, other variants proposed on the different calculi presented, category theoretic approaches, modal logics for mobile process algebra, truly concurrent semantics or the relations between concurrent calculi and linear logic. We will try now to provide a very brief summary of recent work in expressiveness for process algebras.

Saraswat et al. have studied several variants of CCP. One with particular interest from our point of view is the Higher Order-Linear CCP ([21]). This variant has been proven to support mobility. In particular it is the "linear" fragment the one responsible for it. In this variant, based on linear logic, the ask operator corresponds to linear implication, i.e. the condition is consumed once it is satisfied. This means that the store is not monotone and allows for updates. This gives the power necessary to simulate mobility, as well as the notion of state in general.

Some hybrid variants that mix CCP-like languages with π-like languages have been proposed, with the objective of modelling multi-agent systems with some global common knowledge. An example can be found in [9].

Many other variants of CCP have been studied to bring in different concepts. Some of these include: 1) Default CCP, which relates CCP to default logic rather that intuitionistic logic. 2) Timed CCP, which includes an explicit notion of time in the systems modelled. 3) Hybrid CCP, which combines discrete and continuous states. 4) Probabilistic CCP, in which names stand for random variables for which it is possible to specify a domain and a distribution.

On the front of π-calculi, several ideas have been followed such as the Spi-calculus of Abadi and Gordon ([1]), for describing security protocols. Another interesting development inspired on π-calculi, though not directly based on it, is the calculi for *mobile ambients* of Cardelli and Gordon ([8]) concerning larger domains of processes, and their boundaries. These calculi deal with named collections of processes that have an explicit boundary. Interaction between processes can occur only within the boundary, but there are basic operations for moving ambients into, or from, other ambients, and for merging them. In the so-called *safe* ambients, these movements can occur only with mutual agreement from the parties involved.

Recently Zimmer provided, in [50], an encoding of the π-calculus in the pure safe ambients, thus showing the expressive power of the ambient moving primitives. The immediate corollary of this, together with Victor and Parrow's result on the encoding of CCP into π, is that CCP can be translated into ambients. To our knowledge, there has been no concrete result on whether the π-calculus can simulate ambients or not. We speculate that CCP cannot simulate ambients because, as our result showed, the problem of CCP is a problem of security, given its inability to establish private channels. By contrast, the essence of ambient calculi lies in establishing boundaries between collections of processes, and thus, the establishment of private links is fundamental.

All these have been concentrated on the extension of π-calculus, or the development of languages inspired by it. A different line of research has been pursued by Milner and his collaborators by giving π-calculi and other forms of languages for modelling interaction a more general framework. This is the subject of the so called *Action calculi*. It is presented in category theory, thus allowing reasoning about interaction at a more abstract level. Also, in the same way that π-calculi have been generalized to action calculi, fusion calculi (Fusion, Solos, π_F, χ) have been generalized by Gardner and Wischik to *symmetric action calculi* and more recently *fusion systems* ([12], [49]).

A particularly interesting and recent result, related to the semantics of concurrency and the search for appropriate notions of behavioural equivalence, is that of Leifer and Milner [20], in which they provide a method for transforming a UTS into an LTS so that the induced notion of bisimilarity turns out to be a process congruence.

8.3 Future work

The main result of this thesis, the expressiveness gap between pure CCP and π-calculi, in terms of mobility, serves as justification for several extensions that have been proposed for CC languages, such as the ρ-calculus, or Linear CCP. However, all these extensions take the CCP model away from its close relation with Intuitionistic Logic. On the other hand, Linear CCP has an analogous relation with Linear Logic. The encodings of mobility in Linear CCP and Higher Order Linear CCP ([21]) make use of universal quantification as an operator in the language. We speculate that it is not necessary to completely transform the model from the core CCP to Linear CCP in order to obtain the full expressive power of mobility. We suspect that adding just universal quantification to the core CCP, is enough to simulate mobility, while at the same time making the relation with Intuitionistic Logic tighter. This encoding, and the categorical interpretation of core CCP plus universal quantification should be investigated.

Another issue concerning the relation between CCP and logic that requires further exploration is the generalization of the ask operator. We know that ask corresponds to a limited form of implication, in which the condition is not an arbitrary formula, but a formula without quantifiers or implications. The generalization of ask, to receive as condition an arbitrary process should be investigated in terms of its computational content.

The modelling of more complex constraint systems in π and fusion calculi is also a topic that deserves attention, as well as the role of type systems in CCP, and the relation between such type systems and the so-called *sorting disciplines* found in π-calculi.

Finally, Milner's Action Calculi, as mentioned above, is a general framework for the study of interaction, and has been successfully applied to the study of models as diverse as Petri Nets, λ-calculi and π-calculi. Studying CCP from this framework might lead to a new perspective on constraints as interaction.

8.4 Final remarks

Throughout this thesis we emphasised the close relation between the concepts of substitution, equator, diagonal elements and fusions. We saw how equators play the role of name equations by simulating substitution, and a similar remark can be made about fusions. We also saw how fusions where implemented in π in terms of equators. Also, we described how putting a diagonal element in a constraint store can be simulated by a fusion, and how asking whether an equation holds is simulated by the match operator.

One particularly interesting relation amongst the calculi studied is that between CCP and Solos. Recall that in Solos there are no continuations to input and output actions, so in some sense, it is a calculus of pure fusions, and yet it can encode the full Fusion calculus, which in turn means that has a greater expressive power than the core CCP. Since pure fusions seem to correspond to diagonal elements in CCP, it seems strange to regard Solos as having more expressive power than CCP. However, as our result pointed out, in CCP one cannot "contain" the effect of equations. The fact that in Solos a fusion might result from separate input and output actions, which might be in parallel agents, and that the media of communication is not global, means that the restriction operator has a more powerful grip, a finer control, over the influence of names, and therefore, of name equations. In CCP an equation is always issued by one agent. In Solos, separate, parallel agents might cause the equation. In other words, the treatment of name equations as atomic actions in CCP, in contrast with name equations in Solos, which are the result of more primitive actions, limits the control over the domains of influence of names.

This thesis has been a study of concurrency. One of the main objectives of theories for concurrency is to show the inherent power of parallel composition taken as a primitive notion. There is an incredible diversity of paradigms that include concurrency as a basic operation. We have concentrated on those spanning from mobile processes to concurrent constraints. Mobile process calculi have proven to be very a powerful and flexible paradigm, where a wide variety of programming idioms are easily expressible. On the other hand, Concurrent Constraint Programming takes a higher level view of programming to the concurrency world. In its most basic form, this higher level of abstraction is attained at the cost of loosing some expressiveness.

According to Milner ([28]), the goal of process calculi is to distill the essence of concurrency. In this respect, the aim is to find some canonical theory which is general enough to express and reason about all kinds of concurrent systems. We do not know if this goal is indeed achievable, given the diversity of approaches to the treatment of concurrency. For this reason, understanding the relations between different approaches is a fundamental issue. This thesis has been an attempt to give a very broad view of these relations, but in virtue of the richness found in concurrency, we didn't cover many relevant issues. The intention of this work has been to highlight what are some of the fundamental gaps between different theories. This hopefully will shed some light into the essence of concurrency.

Bibliography

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] Gul A. Agha. *Actors—A Model of Concurrent Computation for Distributed Systems*. MIT Press, 1986.

[3] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

[4] H. P. Barendregt. *The Lambda Calculus*. Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, revised edition, 1991.

[5] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[6] Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205–226, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 163–178.

[7] Gérard Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[8] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Proceedings of the First International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '98), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'98), (Lisbon, Portugal, March/April 1998)*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.

[9] Frank S. de Boer, Rogier M. van Eijk, Wiebe van der Hoek, and John-Jules Ch. Meyer. Failure semantics for the exchange of information in multi-agent systems. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on*

Concurrency Theory (CONCUR 2000), volume 1877 of *LNCS*, pages 214–228. Springer-Verlag, 2000.

[10] Uffe Engberg and Mogens Nielsen. A calculus of communicating systems with label-passing. Report DAIMI PB-208, Computer Science Department, University of Aarhus, Denmark, 1986.

[11] Yuxi Fu and Zhenrong Yang. The ground congruence for chi calculus. In Sanjiv Kapoor and Sanjiva Prasad, editors, *20th Conference on Foundations of Software Technology and Theoretical Computer Science (New Delhi, India, December 2000)*, volume 1974 of *LNCS*, pages 385–396. Springer, December 2000.

[12] Philippa Gardner. From process calculi to process frameworks. In Catuscia Palamidessi, editor, *CONCUR 2000: Concurrency Theory (11th International Conference, University Park, PA, USA)*, volume 1877 of *LNCS*, pages 69–88. Springer, August 2000.

[13] Jean Yves Girard. Linear logic: it's syntax and semantics.

[14] Carl Hewitt. A universal modular ACTOR formalism for AI. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*, pages 235–245, 1973.

[15] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science, 1985.

[16] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. *Lecture Notes in Computer Science*, 512:133–??, 1991.

[17] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In M[ario] Tokoro, O[scar] Nierstrasz, and P[eter] Wegner, editors, *Object-Based Concurrent Computing 1991*, volume 612 of *LNCS*, pages 21–51. Springer, 1992.

[18] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. An extract appeared in *Proceedings of FSTTCS '93*, LNCS 761.

[19] Cosimo Laneve and Björn Victor. Solos in concert. In Jiří Wiederman, Peter van Emde Boas, and Mogens Nielsen, editors, *26th Colloquium on Automata, Languages and Programming (ICALP) (Prague, Czech Republic)*, volume 1644 of *LNCS*, pages 513–523. Springer, July 1999.

[20] James Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proceedings of the 11th International Conference of Concurrency Theory, CONCUR 2000*, volume 1877 of *LNCS*. Springer-Verlag, 2000.

[21] Patrick Lincoln and Vijay Saraswat. Higher-order, linear, concurrent constraint programming. Manuscript, January 1993.

[22] Nax P. Mendler, Prakash Panangaden, Phillip J. Scott, and R. A. G. Seely. A logical view of concurrent constraint programming. *Nordic Journal of Computing*, 1995.

[23] Massimo Merro. On the expressiveness of chi, update, and fusion calculi. In Catuscia Palamidessi and Ilaria Castellani, editors, *EXPRESS '98: Expressiveness in Concurrency (Nice, France, September 7, 1998)*, volume 16.2 of *ENTCS*. Elsevier Science Publishers, 1998.

[24] Massimo Merro. On equators in asynchronous name-passing calculi without matching. In Ilaria Castellani and Björn Victor, editors, *EXPRESS '99: Expressiveness in Concurrency (Eindhoven, The Netherlands, August 23, 1999)*, volume 27 of *ENTCS*. Elsevier Science Publishers, 1999.

[25] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[26] Robin Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, LFCS, University of Edinburgh, 1991.

[27] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990, and in *Proceedings of ICALP '91*, LNCS 443.

[28] Robin Milner. Elements of interaction. *Communications of the ACM*, 36(1):78–89, 1993. Turing Award Lecture.

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

[30] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i/ii. Technical Report ECS-LFCS-89-85 and -86, LFCS, University of Edinburgh, 1989.

[31] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Nineteenth Colloquium on Automata, Languages and Programming (ICALP) (Wien, Austria)*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[32] Uwe Nestmann. What is a good encoding of guarded choice? Technical Report RS-97-45, BRICS, 1997.

[33] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory*

(7th International Conference, Pisa, Italy, August 1996, Proceedings), volume 1119 of *LNCS*, pages 179–194. Springer, 1996. Latest full version as report BRICS-RS-99-42, Universities of Aalborg and Århus, Denmark, 1999. To appear in *Journal of Information and Computation.*

[34] Joachim Niehren and Martin Müller. Constraints for free in concurrent computation. In *Proceedings of the Asian Computer Science Conference ACSC'95*, volume 1023 of *LNCS*, 1995.

[35] Catuscia Palamidessi. Concurrent constraint programming. *Lecture Notes in Computer Science*, 850:1–??, 1994.

[36] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of POPL'97*, 1997.

[37] P. Panangaden. The expressive power of indeterminate primitives in asynchronous computation. *Lecture Notes in Computer Science*, 1026:124–??, 1995.

[38] Prakash Panangaden. Notes on the lambda calculus. COURSE NOTES, 1997.

[39] Prakash Panangaden, Vijay A. Saraswat, and Martin Rinard. The semantic foundations of concurrent constraint programming. In ACM, editor, *POPL '91. Proceedings of the eighteenth annual ACM symposium on Principles of programming languages, January 21–23, 1991, Orlando, FL*, pages 333–352, New York, NY, USA, 1991. ACM Press.

[40] Davide Sangiorgi. A theory of bisimulation for the π-calculus. Technical Report ECS-LFCS-93-270, LFCS, University of Edinburgh, 1993.

[41] Davide Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. Technical Report 2539, INRIA, 1995.

[42] Davide Sangiorgi. The π-calculus and its family: interaction and equivalences., April 2000.

[43] Vijay Saraswat. *Concurrent Constraint Programming Languages*. Phd thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, May 1988.

[44] Gert Smolka. A foundation for higher-order concurrent constraint programming. In Jean-Pierre Jouannaud, editor, *1st International Conference on Constraints in Computational Logics*, Lecture Notes in Computer Science, vol. 845, pages 50–72, München, Germany, 7–9 September 1994. Springer-Verlag.

[45] Björn Victor. *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*. PhD thesis, Uppsala University, 1998.

[46] Björn Victor and Joachim Parrow. Constraints as processes. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory (7th International Conference, Pisa, Italy, August 1996, Proceedings)*, volume 1119 of *LNCS*, pages 389–405. Springer, 1996.

[47] Björn Victor and Joachim Parrow. Concurrent constraints in the fusion calculus (extended abstract). In K. G. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of ICALP'98*, volume 1443 of *LNCS*. Springer-Verlag, 1998.

[48] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, Cambridge, Massachusetts, 1994.

[49] Lucian Wischik and Philippa Gardner. Symmetric action calculi (abstract). In Ilaria Castellani and Björn Victor, editors, *EXPRESS '99: Expressiveness in Concurrency (Eindhoven, The Netherlands, August 23, 1999)*, volume 27 of *ENTCS*. Elsevier Science Publishers, 1999.

[50] Pascal Zimmer. On the expressiveness of pure mobile ambients. In Luca Aceto and Björn Victor, editors, *EXPRESS '00: Preliminary Proceedings of the 7th International Workshop on Expressiveness in Concurrency (State College, USA, August 21, 2000)*, volume NS-00-2 of *BRICS Notes Series*, pages 81–104, 2000.