THE UNIVERSITY *of* EDINBURGH

Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms

Davide Sangiorgi

Doctor of Philosophy

University of Edinburgh

1992

Abstract

We study *mobile systems*, i.e. systems with a dynamically changing communication topology, from a process algebras point of view. Mobility can be introduced in process algebras by allowing names or terms to be transmitted. We distinguish these two approaches as *first-order* and *higher-order*. The major target of the thesis is the comparison between them.

The prototypical calculus in the first-order paradigm is the *π-calculus*. By generalising its sort discipline we derive an ω-order extension called *Higher-Order π-calculus* (HOπ). We show that such an extension does not add expressiveness to the π-calculus: Higher-order processes can be faithfully compiled down to first-order, and respecting the behavioural equivalence we adopted in the calculi. Such an equivalence is based on the notion of bisimulation, a fundamental concept of process algebras. Unfortunately, the standard definition of bisimulation is unsatisfactory in a higher-order calculus because it is over-discriminating. To overcome the problem, we propose *barbed bisimulation*. Its advantage is that it can be defined *uniformly* in different calculi because it only requires that the calculus possesses an *interaction* or *reduction* relation. As a test for barbed bisimulation, we show that in CCS and π-calculus, it allows us to recover the familiar bisimulation-based equivalences. We also give simpler characterisations of the equivalences utilised in HOπ. For this we exploit a special kind of agents called *triggers*, with which it is possible to reason fairly efficiently in a higher-order calculus notwithstanding the complexity of its transitions.

Finally, we use the compilation from HOπ to π-calculus to investigate Milner's

To my parents

and to the memory of my grandmother Rambelli Maria

Acknowledgements

I simply cannot express how much I am indebted to my supervisor Robin Milner. He was always willing to discuss my work and ready to give enlightening suggestions. His precious guidance has deeply marked not only my research, but also my way of doing research; and his enthusiasm has kept live my own throughout the development of the thesis.

I also owe a lot to my former italian supervisor, Andrea Maggiolo-Schettini: For convincing me to apply to Edinburgh when I was about to join some company and afterwards, for his constant support and friendship. Thanks also to Emanuela Fachini and Rocco De Nicola for their encouragement.

The Computer Science Department in Edinburgh has offered a stimulating and exiting environment for my work. In particular, I would like to thank the past and present members of the Concurrency Club, for the fruitful feedback and many interesting discussions.

Thanks to Stuart Anderson, Glenn Bruns, Faron Moller, Andrea Maggiolo-Schettini, Peter Sewell and David Turner who read (parts of) a draft of this thesis and helped my with my (hopeless) English.

Finally, I am most grateful to all friends who made my stay in Edinburgh enjoyable; in particular Claudio, David, Ian, Marcelo, Roberto, Søren. A special mention to Laurence, for her company and for having helped me to stay serene during the final writing of the thesis.

The work has been supported by a scholarship from the Consiglio Nazionale delle Ricerche, Italy.

* * *

This is a revised version of my thesis; I have benefited a lot from the detailed criticism of my examiners Matthew Hennessy and Gordon Plotkin.

Main Notations

The following tables summarise the notation used in this thesis. The page numbers refer to the page where the notation is first defined or used.

Process Theory

Symbol	Description	Page
P, Q, R, T	processes	19
A	agents	21
F, G, E	abstractions	21
$C[\cdot]$	context	22
$a, b, \ldots$	names	19
$\mathbf{0}$	inactive process	20
$\sum_{i \in I} \alpha_i.P_i$	process sum	19
$P + Q$	binary sum	20
$P \mid Q$	parallel composition	19
$\prod_{i=1}^{n} P_i$	multiple parallel composition	23
$\boldsymbol{\nu}\, a\, P$	restriction	19
$D \stackrel{def}{=} (\tilde{x})P$, $D \stackrel{def}{=} (\tilde{U})P$	constant definition	19
$D\langle\tilde{x}\rangle$, $D\langle\widetilde{K}\rangle$	constant application	19
$X\langle K\rangle$	variable application	34
$A\langle\widetilde{K}\rangle$	meta application	37
$(x)A$, $(X)A$,	(typical) abstraction	19
$!\, P$	replication	23
$[x = y]$	matching	19

$\tau.P$	silent prefix	23
$m \star F$	prefix abbreviation	70
$P\,\{m := F\}$	special replication-abbreviation	70
α	prefix	19
$x(\tilde{y})$, $x(\tilde{U})$,	input prefix	19
$\overline{x}\langle\tilde{y}\rangle$, $\overline{x}\langle\widetilde{K}\rangle$	output prefix	19
$\{\tilde{x}/\tilde{y}\}$, $\{\widetilde{K}/\tilde{U}\}$	substitution	22
$fn(A)$, $bn(A)$	free and bound names of A	22
s, S	subject and object sorts	24
El	element sort	36
Pr_π	π-calculus processes	19
$HO\pi, HO\pi^\circ$	closed and open HOπ agents	35
$THO\pi, THO\pi^\circ$	closed and open triggered agents	77

Process Semantics

μ	action	30
$fn(\mu)$, $bn(\mu)$, $n(\mu)$	free names, bound names and names of μ	30
$\xrightarrow{\mu}$	labelled transition system	30
$\longrightarrow$	reduction relation	29
$\Longrightarrow$	reflexive and transitive closure of $\longrightarrow$	41
$=$	equality up-to α-conversion	23
$\equiv$	structural congruence	28
$\sim_e$, $\approx_e$	strong and weak early bisimulation	45
$\dot{\sim}$, $\dot{\approx}$	strong and weak barbed bisimulation	50
$\sim$, $\approx$	strong and weak barbed equivalence	51
$\sim^c$, $\approx^c$	strong and weak barbed congruence	51
$\simeq_{Ct}$, $\cong_{Ct}$	strong and weak context bisimulation	65
$\simeq_{Tr}$, $\cong_{Tr}$	strong and weak triggered bisimulation	88
$\simeq_{Nr}$, $\cong_{Nr}$	strong and weak normal bisimulation	94

Table of Contents

Chapter 1

Introduction

Concurrent systems are one of the most challenging areas of research in computer science. The notion however, is not exclusive to the "computer world". For instance, a colony of ants and an aggregate of elementary particles are examples from biology and physics. We view a concurrent system as the composition of independent entities — the *processes* — which may interact and exchange messages.

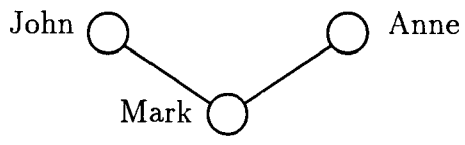
The analysis and description of concurrent systems is a difficult task. The overall behaviour of the system is the result of the interaction of its individual components and their consequent evolution; moreover the occurrence of such interactions may not be deterministically predictable and different interactions may take place at the same time.

Numerous models and methods for reasoning about concurrent systems has been proposed. Among these, *Process Algebra* is one of the most successful. In process algebras both the analysis and the description of a system are carried out in an algebraic setting. Robin Milner's work on the Calculus of Communicating Systems (CCS) [Mil80] is generally accepted as initiator and landmark in the field. (A more refined version of the theory of CCS is in [Mil89]). Inspired by the λ-calculus, CCS is based upon the selection of a few primitive constructors each embodying a distinct and intuitive idea. But the choice of constructors strikingly differs from λ-calculus. Perhaps this is not too surprising, given the

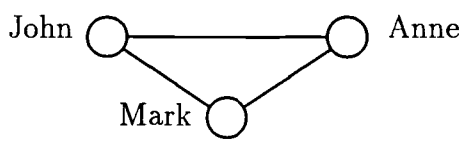
different (and in some sense opposite) objectives of the two calculi. The λ-calculus studies functions and their applicative behaviour; it essentially describes sequential computations. By contrast, CCS aims at parallel computations; interaction and communication are the central idea.

A limitation of CCS (as well as of the other "traditional" process algebras like CSP [Hoa85], ACP [BK84,BK85], and MEIJE [AB84]) is that the set of ports used by a term to perform communications with other terms, is fixed by its syntax. This prevents CCS from effectively describing *mobile* systems, i.e. systems with a dynamically changing communication topology.

To give an example of mobility, consider an abstract view of an electronic mail system with three users John, Mark and Anne in which only John and Anne do not know each other's address. Pictorially we can view the system as follows:



Suppose that subsequently John and Anne want to communicate with each other. They could do so by going through Mark, but a better way — probably quicker and also safer for their privacy — is to ask Mark for the missing address. This results in the appearance of a connection between John and Anne:



Thus the messages which have been exchanged in the system have affected the communication interface of some of its subcomponents. This is the essence of the notion of mobility.

Mobility is common in operating systems. Take the case of a resource which has a single owner at any time but whose ownership can be changed as time passes; or *process migration* [DO91], where tasks or processes can be exchanged among processors to optimise processor usage. Another example from parallel

programming is a procedure which can be called simultaneously by any number of processes; here the problem is to guarantee that each activated instance of the procedure returns its result to the correct calling process. At best this can be modelled in CCS with a summation indexed over the callers, each of which uses a different port to interact with the procedure. The drawback is that all potential callers must be known in advance; moreover this representation does not capture the mobility which is involved.

The purpose of this thesis is to add to the understanding of the phenomenon of mobility in process algebras. The terms *first-order paradigm* and *higher-order paradigm* identify two approaches to mobility in process algebras. In the higher-order paradigm, mobility is achieved by allowing agents, namely processes or parametrised processes, to be passed as values in a communication; in the first-order paradigm only ports, or names, can be transmitted. The former is closer to λ-calculus, in that a *term* can be bound to a variable; the latter resembles the way in which object oriented programming languages allow objects to refer to one another. In fact, in [Mil90a] the two approaches are called by Milner *function* and *object paradigm*, respectively.

In the next section we give a brief overview of the major attempts to express mobility in models for concurrent systems. Thereafter, we discuss the motivations for the main research directions in the thesis. The comparison between first and higher-order paradigm is a central theme. A considerable amount of effort has been devoted to the development of a higher-order calculus, in particular in its semantics. This because although various higher-order calculi have already appeared, there is not a well-established mathematical treatment of the higher-order paradigm. The situation is different at first-order. The π-calculus is the prototypical calculus and its elegant theory has been explored by Robin Milner, Joachim Parrow and David Walker in [MPW92,MPW91]. Indeed, the higher-order calculus we develop is founded on the π-calculus. Calculi for mobile systems allow elegant representation of the λ-calculus. The investigation into the representation of func-

tions as processes is the topic of the last part of the thesis. The starting point is Milner's work on the encoding of λ-calculus into π-calculus presented in [Mil90a].

1.1 Background

The ancestor of all concurrency models with mobility is probably Hewitt's *actor system* ([Hew77,HB77,Agh86,Agh90]). Actors are active objects which communicate each other via asynchronous messages, themselves thought of as actors. The actor's behaviour, which may be viewed as its local state, specifies the response to an incoming message. An actor can only send messages to those actors of which it has *acquaintance*. The dynamicity of the system derives from the possibility both of creating new actors and of modifying an actor's acquaintance during the computation. The latter because upon receiving a message an actor may add to its acquaintances any name mentioned in the message and because acquaintances may be forgotten during computation. The model has had a considerable success and has given rise to an intensive research on system architecture and object-oriented programming language constructs. However, from a theoretical point of view it has suffered from the lack of a convincing mathematical treatment of its foundational concepts. An example is the mechanism for generating new names for actors and for guaranteeing their unicity. Also the treatment of behavioural equivalences and of system compositionality (according to which a system has a structure and can be analysed in terms of its subcomponent) is not quite satisfactory, especially when compared with their treatment in process algebras. Interesting work developing of a process algebra for actor-like objects is being carried out by Honda and Tokoro (see [HT]).

Graph-grammars [Ehr79] provide an elegant way to model systems with a dynamic behaviour. In [JR89,JR90], Janssens and Rozenberg use graph-grammars to give semantics to a (restricted) version of the actor systems. The nodes of a graph represent actors and the directed edges represent acquaintances; the dynamic evol-

ution of an actor is described in terms of transformations of configuration graphs.

A graph-grammar approach might also be interesting to introduce mobility in *Petri Nets* [Rei85], whose standard definition requires a static, fixed structure. Studies on the connection between Petri Nets and graph-grammars have been carried out among the others, by Kreowsky [Kre80], Pinna and Maggiolo-Schettini [PMS90]; but they do not consider explicitly mobility. Engelfriet, Leih and Rosenberg take a different direction in [ELR90b,ELR90a]. Their model is a generalisation of Petri Nets aiming at formalising properties of object-based systems. The places of a Petri Net are enriched with extra structure which records the essential state information of the object they represent. For this, three functions are used: One maps each place to the object it represents; another indicates the mode of such object in that state (unborn, alive, dead); the last gives the content of its local memory, possibly including references to other objects. A transition of the Petri Net corresponds to the change of state of one (or more) objects. Creation or destruction of objects and modification of their acquaintance sets are modelled as special instances of such transitions. However, the description of mobility remains rather indirect.

The object paradigm has been investigated in the setting of algebraic parametrised specification by Astesiano and his colleagues in a series of papers ([ARW86, AR87,AG88]). However, to the purposes of this thesis, the most relevant studies dealing with communication of behaviour expressions are [Bou89,Nie89,Tho90].

The γ-calculus introduced by Gerard Boudol in [Bou89] is an extension of λ-calculus with a communication mechanism inspired by CCS. It has two parallel constructs. The first, called interleaving, does not allow communication between agents. The second, called cooperation, forces two agents to communicate on every port and as a consequence, it is non-associative. No restriction or hiding operator is considered. Boudol uses the γ-calculus to represent the λ-calculus. He shows that the latter is a subcalculus of former, in the sense that β-reduction is an instance of the interaction law. The algebraic theory of γ-calculus and its

expressiveness in the modelling of concurrent processes remain to be properly investigated.

Flemming Nielson's *Typed Parallel Language* [Nie89] tries to integrate process algebra and the λ-calculus. The focus is mainly on types, as a means of ensuring more reliable programs. Thus the starting point is the *typed λ-calculus* rather than its untyped version. Processes also have types, which record their communication possibilities. Unfortunately the syntax is rather elaborate and the use of types is arduous; for instance it forces a heavy side condition in the definition of parallel composition. But the attempt towards a notion of type for processes is very interesting and in our view deserves further investigation.

The most conspicuous effort towards the development of an algebra of higher-order processes is by Thomsen [Tho90]. His Calculus of Higher Order Communicating Systems (CHOCS) is an extension of CCS in the sense that the basic operators in the two calculi are the same; the difference is that in CHOCS a communication always causes the exchange of a process. A salient feature of CHOCS is the *dynamic binding* for the restriction operator, as opposed to *static binding* which Thomsen experiments within Plain CHOCS ([Tho90, Chapter 5] and [Tho89]). An example will illustrate the difference. In the notation of this thesis, '$\boldsymbol{\nu}\, b$' represents restriction on the name b, '$\overline{a}\langle P\rangle$' the output of the process P, '$a(X)$' input prefixing and '|' parallel composition. The following is a legal interaction in CHOCS:

$$a(X).X \mid \boldsymbol{\nu}\, b\left(\overline{a}\langle P\rangle.Q\right) \longrightarrow P \mid \boldsymbol{\nu}\, b\, Q$$

The reduction destroys the privacy of the possible b-link between P and Q, since the free occurrences of b in P have evaded the restriction which embraced them. In static binding this is not allowed. Restriction becomes a formal binder like the abstraction operator of λ-calculus and as such, subject to α-conversion. Its peculiarity is that its scope may change because of a communication; thus in Plain CHOCS the above interaction becomes

$$a(X).X \mid \boldsymbol{\nu}\, b\left(\overline{a}\langle P\rangle.Q\right) \longrightarrow \boldsymbol{\nu}\, b\,(P \mid Q)$$

The main advantage of dynamic binding is a simple operational semantics. But this does not compensate for the difficulty in the analysis of a program from its text. In this thesis we deal only with static binding.

The merit of being the first to use static binding and to understand how to handle it algebraically goes to Mogens Nielsen and Uffe Engberg [EN86]. Their Extended CCS (ECCS) follows the first-order paradigm, previously attempted by Astesiano and Zucca [AZ84] who used label expressions to emulate parametric channels. The success of ECCS was frustrated by its heavy syntactic definition. For instance there are three different kinds of variables, ranging over labels, expression behaviours and values, and the interplay between them is often confusing. A simplification of the theory of ECCS was pursued by Robin Milner, Joachim Parrow and David Walker with π-calculus [MPW92]: They achieve an elegant uniform syntactic and semantic framework in which names and agents are the only entities involved. Subsequently, a more refined version of the π-calculus was proposed by Milner [Mil91], where most notably, *sorts* and communication of *tuples* are added. This calculus is at the centre of the attention in this thesis and will be presented in detail in Chapter 2.

We conclude by mentioning firstly Oscar Nierstrasz's *Object Calculus* [Nie], in which he tries to combine π-calculus and λ-calculus with the purpose of defining a uniform framework for the semantics of concurrent object-oriented languages. Secondly, the languages DyNe [KS85], Nil [SY85], PFL [Hol83], LCCS [Let92] and FACILE [GMP89]. All allow for dynamic linkage reconfiguration, but the emphasis is more on programming language and implementation issues.

1.2 Representability of the higher-order paradigm at the first-order

What makes the π-calculus attractive is the strength of its elementary theory and its reduced complexity. The choice of the first-order paradigm was motivated by

the belief that reference-passing is enough to represent more involved operations like process-passing. This thesis validates that claim, showing that no power — only convenience — is gained by moving to the higher-order paradigm.

To this end, we introduce a new calculus, called *Higher-Order π-calculus* (HOπ), which enriches the π-calculus with explicit higher-order communications. In the HOπ not only names, but also processes and parametrised processes of arbitrarily high order, can be transmitted. In this sense, if the ordinary π-calculus is of first-order and Thomsen's Plain CHOCS is of second-order, then HOπ is of ω-order. We show that the HOπ is *representable* within the π-calculus. This proves that the first-order paradigm, being by far simpler, should be taken as *basic.* Such a conclusion takes away the interest in the opposite direction, namely the representability of the π-calculus within a language using purely communications of agents, which in fact has not been investigated in this thesis.

But what does it mean that a given *source* language is representable within a given *target* language? We can identify essentially three phases:

(1) Formal definition of the semantics of the two languages;

(2) Definition of the encoding from the source to the target language;

(3) Proof of the correctness of the encoding w.r.t. the semantics given.

As regards (2), it is enough to add that for practical usefulness, the encoding — besides being reasonably "simple" — must be *compositional*, that is its definition on a term should only depend upon the definition on its immediate constituents.

Some further comment on (1) and (3) is worthwhile. There are two major approaches for giving semantics to a programming language. In the *denotational approach* a valuation function maps a program directly to its mathematical meaning or *denotation.* Here the correctness of an encoding can be investigated by considering the relationship between the meaning of a term in the source language and the meaning of its translation into the target language. Denotational semantics has been very successful in modelling many sequential languages; programs are typically viewed as computational functions from the domain of input

values to the domain of output values. However, to date there has not been an equally satisfactory denotational treatment of concurrency. The tools employed for sequential programs are inadequate, since an account of the behaviour of a concurrent system must also take into account the *internal* states it can reach.

The predominant approach to the semantics of concurrent systems is *operational.* Following Plotkin [Plo81], the possible evolutions of a process are described in terms of a *transition system* in which the rules are defined inductively on the *structure* of a term. The major drawback of operational semantics is that it is too concrete. Thus, transition systems have to be quotiented by *equivalence* relations which abstract from unwanted details. The operational method necessitates a different approach to translation-correctness, where behaviours rather than meanings are compared. For the purpose of proving the correctness of the encoding, the choice of the behavioural equivalence, besides being "interesting", should be uniform on the calculi. Moreover, we want the encoding to be *fully abstract*, i.e. two source language terms should be equivalent if and only if their translations are equivalent. But since this does not reveal *how* this respectfulness is achieved, the result should be completed with the *operational correspondence* between a term and its translation (i.e. the connection between their transitions).

With the full abstraction demand, we have taken a *strong* point of view on representability. Indeed, while soundness is a necessary property, one might well consider milder forms of completeness, for instance by limiting the testing on target terms to encodings of source contexts. We asked for full abstraction because we wish to use the target terms in *any* contexts; and when two source terms are indistinguishable, their encodings should *always* be interchangeable. In other words, we want to be able to switch freely between the two calculi. In our case, where the source language is HOπ and the target language is π-calculus, this allows us on the one hand to make use of the abstraction power of HOπ, which comes from its ω-order nature. On the other hand, to rely on the more elementary and intuitive theory of π-calculus when reasoning over agents; in virtue of the

representability result this theory can be lifted up to HOπ.

1.3 Bisimulation in higher-order process calculi

Behavioural equivalences proposed for process algebras may be classified as *extensional* (or *interleaving*) where concurrency is reduced to sequentiality plus nondeterminism, or as *intensional* (or *true concurrent*) where parallel composition is considered a primitive operator. The transport of a well-established equivalence to the higher-order setting is not always straightforward. We focus here on (interleaving) bisimulation equivalence, widely accepted as the finest extensional equivalence one would want to impose. It was originally proposed by Milner and Park [Mil80,Par81] in the early 80's and since then has become one of the most stable concepts in the theory of concurrency.

Traditionally, a bisimulation is defined on top of a *labelled* transition system, and requires that an action be matched by another only if they have identical labels. But this is certainly too strong in calculi expressing mobility. For instance it does not respect α-conversion. Now, with name passing the damage is limited to this and can be repaired by adding appropriate side conditions in the definition of bisimulation; but in higher-order calculi the damage goes well beyond. Obvious algebraic laws. such as the commutativity of parallel composition, are lost: For instance take the processes $\overline{a}\langle P \mid Q\rangle.\mathbf{0}$ and $\overline{a}\langle Q \mid P\rangle.\mathbf{0}$ (where $\mathbf{0}$ represents the inactive or null process). They can be be distinguished because the first performs an output action which transmits $P \mid Q$, whereas the second transmits $Q \mid P$, which in general is syntactically different from $P \mid Q$. The approach taken by Thomsen in [Tho90], following earlier ideas by Astesiano and Boudol [AG88,Bou89], is to require *bisimilarity* rather than *identity* of the processes emitted in a higher-order output action (*higher-order bisimulation*). This seems a conceivable requirement, but one can object that it is still too strong. To see this, take

$$P \stackrel{def}{=} \overline{a}\langle \mathbf{0}\rangle.\mathbf{0} \qquad\qquad Q \stackrel{def}{=} (\boldsymbol{\nu}\, x)\overline{a}\langle \overline{x}.\mathbf{0}\rangle.\mathbf{0}$$

The processes P and Q differ in the value carried by $\overline{a}$, which is $\mathbf{0}$ in the former, and $\overline{x}.\mathbf{0}$ in the latter. But since the name x is restricted, process $\overline{x}.\mathbf{0}$ will never find a partner to communicate with. It is therefore a deadlocked process, and as such semantically the same as $\mathbf{0}$. Hence, in any context P and Q give rise to the same interactions, and accordingly, should be considered equivalent. Unfortunately, they are distinguished by higher-order bisimulation, since $\overline{x}.\mathbf{0}$ and $\mathbf{0}$ are not equivalent. One could think of adjusting this example by imposing a different treatment of the outermost restriction in Q, thus comparing $\mathbf{0}$ with $(\boldsymbol{\nu}\, x)\overline{x}.\mathbf{0}$ rather than $\overline{x}.\mathbf{0}$. But this is certainly wrong and in fact would be disastrous in other situations. For instance, it would equate the processes

$$(\boldsymbol{\nu}\, x)\overline{a}\langle\overline{x}.R\rangle.x \qquad \text{and} \qquad (\boldsymbol{\nu}\, x)\overline{a}\langle\mathbf{0}\rangle.x$$

which by contrast may have completely different possibilities of interactions (the former can communicate at x with the recipient of $\langle\overline{x}.R\rangle$ and thus activate a copy of R). This choice would also yield the law

$$(\boldsymbol{\nu}\, b)\overline{a}\langle P\rangle.Q = \overline{a}\langle\boldsymbol{\nu}\, b\, P\rangle.Q$$

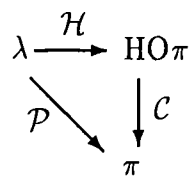
Yet in the first process all copies of P activated by its recipient share the name b, whereas in the second one, the name b is private to each copy.

Our aim though, was not merely to solve this specific problem of higher-order process algebras. An important requirement for us was that the solution proposed could be used *uniformly* in different calculi. First of all because this would provide us with a fundamental tool for comparing them, the kind of issue with which this thesis is mainly concerned. Secondly, because it would reveal something of the "essence" of a natural bisimulation and help us to understand whether bisimulations put forward in various process algebras are indeed instances of a more general and common notion. We claim to have achieved this by introducing the notion of *barbed bisimulation*, which focuses on the interaction (or reduction) relation of the calculus. There are other reasons for our interest in barbed bisimulation; for these we refer to Section 3.1.

1.4 Encoding of λ-calculus

In [Mil90a] Milner examines the encoding of the λ-calculus into the π-calculus; more precisely, he shows how the *lazy* and *call-by-value* λ-evaluation strategies [Abr89, Plo75] can be faithfully mimicked.

We have made use of the representability of the HOπ in the π-calculus to investigate these encodings. The clue for this is the following. W.r.t. the π-calculus, the higher-order machinery of the HOπ gives a simpler description of λ-calculus, with a closer correspondence between reduction on λ-terms and on their process counterparts. Such a HOπ description is mapped by our compilation from HOπ into π-calculus directly — at least for lazy λ-calculus — onto Milner's encoding. That is, if $\mathcal{P}$ and $\mathcal{H}$ are respectively, the π-calculus and HOπ encoding, and $\mathcal{C}$ is the compilation from HOπ to π-calculus, then the following diagram commutes:



Thus one can reason with $\mathcal{H}$ and infer all results obtained onto $\mathcal{P}$ as well.

Our study of the λ-calculus encodings is not intended to be just an example of the usefulness of the representability of HOπ in π-calculus. Virtually all of the different recent generalisations of process calculi with the capability of treating — directly or indirectly — processes as first class objects have put forward attempts at embedding the λ-calculus. A deep comparison between a process calculus and λ-calculus is interesting for several reasons. From the process calculus point of view, besides being a remarkable test of its expressiveness, it usually contributes much towards getting a deeper insight into its theory. From the λ-calculus point of view, it provides the means to study λ-terms in more powerful contexts than the pure (sequential) λ-contexts (namely contexts where sophisticated parallel and non-deterministic operators can be expressed). This is important when considering

the integration of functional and concurrent calculi: For example, we might want to know when two functional terms can be exchanged without affecting the behaviour of the process in which they are used. Moreover an encoding of the λ-calculus into a process algebra allows the study of the former using the instruments developed in the latter. For instance, an important behavioural equivalence upon process terms could give rise to a new interesting equivalence on λ-terms. Also, the importance of some λ-calculus evaluation strategy is strengthened if it can be shown to be efficiently encoded in a "successful" process calculus, with the result of intensifying the study of this strategy or even to find new appealing refinements to it.

Other motivations for describing functions as processes are to provide a semantic foundation for languages which combine concurrent and functional programming and to develop parallel implementations of functional languages.

1.5 Summary of the thesis

We summarise here the contents of the different chapters of the thesis.

We start Chapter 2 with the polyadic π-calculus: We review its syntax, its notion of sort and its operational semantics. By extending the sort discipline of π-calculus we derive the *Higher-Order π-calculus* (HOπ). The syntax and operational semantics of HOπ are natural generalisations of those of the π-calculus. In the last part of the chapter, we present some well-established theory and some conventions related to the notion of bisimulation which will be used throughout the thesis.

In Chapter 3 we seek for a way of defining uniformly bisimulation-based equivalences over different calculi. As discussed in Section 1.3, this is especially important for higher-order process calculi, in which the standard definition of bisimulation induces an over-discriminating relation. The idea is to try to achieve this by equipping a global observer with a *minimal* ability to observe actions and/or process states. We examine first the case in which there are *no* observables and

only the *interaction* or *reduction* relation is present. Unfortunately this does not give enough discriminating power. Therefore we add the capability of observing some properties of the states. The predicates which are used for this give — in a process calculus — visibility of the channel at which an action occurs. We call the resulting bisimulation relation *barbed bisimulation*. By parametrising barbed bisimulation over the class of *static* contexts and over the class of all contexts, we define *barbed equivalence* and *barbed congruence*, respectively. We prove that in CCS and π-calculus, barbed equivalence and congruence coincide with the well-known bisimulation-based equivalences.

In Chapter 4 we deepen the analysis of barbed bisimulation and congruence in HOπ. The target is to derive as simple as possible characterisations for them. A central rôle is played by the *triggers*, intuitively elementary agents whose only functionality is to activate a copy of another agent and provide it with the possible arguments. Using triggers, any subagent of a given agent can be factorised out: This is the content of the *factorisation theorem*. Building on this, we introduce the subclass of *triggered agents*, in which every agent emitted in an output or "expected" in an input is a trigger. Thus higher-order communications have become homogeneous and have lost all their potential richness and variety. This greatly simplifies the reasoning over agents. In particular, on triggered agents barbed equivalence coincides with *triggered bisimulation*. The advantage of the latter is that the clause for higher-order outputs just requires *identity* (modulo α-conversion) on the labels of two matching actions and the clause for higher-order inputs only contemplates the input of a fresh trigger. We then define a mapping $\mathcal{T}$ which transforms every agent into a triggered agent. By exploiting $\mathcal{T}$, we are able to prove a simple characterisation of barbed equivalence on the whole class of HOπ agents (not necessarily in triggered form), called *normal bisimulation*. This is not as simple as triggered bisimulation, but at least no universal quantifications is present in the definition.

In Chapter 5, we show how the higher-order constructs of the HOπ can be

effectively compiled down to π-calculus. Formally, the compilation $\mathcal{C}$ is derived in two steps. Firstly the mapping $\mathcal{T}$ introduced in Chapter 4, which maps agents to triggered agents; secondly, a map from triggered agents to first-order agents (and first-order sortings). We show the operational correspondence between an HOπ agent and its encoding π-calculus agent and prove that $\mathcal{C}$ respects barbed equivalence and congruence. We conclude the chapter by comparing our work with Bent Thomsen's, which first attempted the translation of a higher-order calculus, namely Plain CHOCS, into the π-calculus.

In Chapter 6 we carry out our investigation into the representation of functions as processes. The starting point is Milner's work on the encoding of lazy and call-by-value λ-calculus into π-calculus. The chapter is ideally divided into two parts. In the first, we present analogous encodings into HOπ. The higher-order nature of HOπ makes them easier to understand and to handle than those into π-calculus; moreover the two can be compared via the compilation $\mathcal{C}$. On the lazy encodings, this yields a commutative diagram. For call-by-value the situation is less sharp. On the one hand because in his original work [Mil90a], Milner gives two encodings; on the other hand because $\mathcal{C}$ does does not return either. Seemingly, to obtain them some code transformation is necessary. The study of these transformations suggests a correction in Milner's encodings, which improves their faithfulness to the call-by-value discipline. The study also reveals a problem in Milner's second encoding, for which β-reduction in not valid. In the second part of the chapter we focus on the lazy λ-calculus encodings, more attractive because of their apparent canonicity. Given the correspondence proved in the first part, we can work with our HOπ encoding and extend the results to Milner's encoding as well. We show that they give rise to a λ-model, in which a weak form of extensionality holds. The model is not fully abstract though. To obtain full abstraction we follow two directions: In the *restrictive* approach the semantic domain of processes is cut down; in the *expansive* approach λ-calculus is enriched with *constants* to obtain a *direct* characterisation of the equivalence induced by the encodings.

In Chapter 7 we comment on the results obtained and we present directions for potential future work.

Chapter 2

From π-calculus to Higher-Order π-calculus

We start the chapter by reviewing the basic concepts of the (polyadic and sorted) π-calculus. We first present its syntax. Then its operational semantics, both in terms of a reduction relation and of a labeled transition system. The presentation is essentially the same as in [MPW92,Mil91], to which the reader is referred for more details. The major difference is that we allow infinite sums and agents with infinite free names in the syntax.

In the π-calculus only names can be communicated, and hence only "first-order" sortings are employed. However, the sorting discipline naturally lends itself to a higher-order extension, following which we have derived a calculus called the *Higher-Order π-calculus*, briefly *HOπ*. In the HOπ not only names, but also processes and abstractions over processes of arbitrarily high order, can be exchanged. As such it is an ω-order calculus; then Thomsen's Plain CHOCS, in which exclusively processes can be communicated, can be seen as a particular second-order case of HOπ.

The HOπ is introduced in the second part of the chapter. Its syntax and operational semantics are derived with natural generalisations of those for the π-calculus. Again, we give the operational semantics both in terms of a reduc-

tion relation and of a labelled transition system. In the last part of the chapter we present some conventions regarding bisimulations and we review the major "bisimulation up-to" techniques, used to facilitate the construction of bisimulations. We conclude with the definition of early bisimulation and congruence for the π-calculus.

NOTATION. Let us first set out some general notation for the thesis.

- We use $\mathcal{R}$ to range over relations. We write $(P, Q) \in \mathcal{R}$ also as $P\ \mathcal{R}\ Q$.
- $\mathcal{R}^{-1}$ denotes the inverse of $\mathcal{R}$, i.e. $\mathcal{R}^{-1} = \{(P, Q)\ :\ (Q, P) \in \mathcal{R}\}$.
- The binary composition of two relations $\mathcal{R}$ and $\mathcal{R}'$ is written as $\mathcal{R}\mathcal{R}'$. Thus $P\mathcal{R}\mathcal{R}'Q$ means that T exists s.t. $P\,\mathcal{R}\,T$ and $T\,\mathcal{R}'\,Q$.
- We use a tilde to indicate a *tuple*, that is an ordered sequence of elements. For instance, if x stands for a name, then $\tilde{x}$ is a tuple of names. A tuple can be *empty*, but can also be *infinite*.
- We group brackets. Thus $(x_1) \ldots (x_n)$ becomes $(x_1, \ldots, x_n)$ and $\langle x_1 \rangle, \ldots, \langle x_n \rangle$ becomes $\langle x_1, \ldots, x_n \rangle$.
- $\emptyset$ is the empty set.
- $\cup$, $\cap$ are the familiar union and intersection of sets. Sometimes, if a set H_2 contains a single element h, we abbreviate $H_1 \cup H_2$ as $H_1 \cup h$ (similarly for $\cap$).
- For sets H_1, H_2, the set of elements which are in H_1 but are not in H_2 is $H_1 - H_2$.

The polyadic π-calculus

2.1 Syntax of π-calculus

2.1.1 The language

We use $a, b, c, \ldots, x, y, z, \ldots$ to range over the class of names (we shall use the words 'name', 'port' and 'channel' interchangeably) and P, Q, R, T to range over the class of processes. The class Pr_π of the π-calculus processes is built using the operators of prefixing, sum, parallel composition, restriction, matching and constant application, with constants represented by D in the grammar below:

$$P \;::\; \textstyle\sum_{i\in I} \alpha_i.P_i \;\mid\; P_1 \mid P_2 \;\mid\; \boldsymbol{\nu}\, x\, P \;\mid\; [x=y]P \;\mid\; D\langle\tilde{x}\rangle$$

α is called *prefix* and can be either an input or an output:

$$\alpha \;::\; x(\tilde{y}) \;\mid\; \overline{x}\langle\tilde{y}\rangle$$

When a constant application $D\langle\tilde{x}\rangle$ is used, D must have been defined. The defining equations for constants constitute a set and are of the form $D \stackrel{def}{=} (\tilde{x})P$, where the parameters $\tilde{x}$ collect all names which may occur free in P. The latter requirement on $\tilde{x}$ is useful to have simple inductive definitions of substitution and of the free names of an agent. However, for simplicity, in some cases we shall put in the parameters $\tilde{x}$ only those names of P which are supposed to be instantiated (for instance, we might simply write $D \stackrel{def}{=} (x)\overline{a}\langle x\rangle.\mathbf{0}$, that is omitting a in the parameters, if we intend to maintain a in all our uses of D).

There are a few constraints in the above expressions: Firstly, in an input prefix $x(\tilde{y})$ and in a constant definition $D \stackrel{def}{=} (\tilde{x})P$, the tuples $\tilde{x}$ and $\tilde{y}$ are made of all distinct elements (this because they represent binders, as better precised below). Secondly, the brackets $()$ and $\langle\rangle$ are omitted when the name tuple inside is empty. Thirdly, the tuple $\tilde{y}$ in input and output prefixes is finite. By contrast, no limitation is imposed on the tuple $\tilde{x}$ of constant definitions and applications which, therefore, may be infinite. This allows us to have constants which use an infinite

number of names, for instance a counter which at each step is able to emit a signal at a different port. Finally, we suppose that it is always possible to α-convert the names used in an expression to "fresh" names and to augment the set of defining equations — under the condition that existing equations are not overwritten.[1] A way of ensuring this is to require that the classes of names and constant symbols are uncountable.

An input-prefixed process $x(\tilde{y}).P$ waits for a tuple $\tilde{z}$ to be transmitted along x; then the continuation is the process P with the tuple $\tilde{y}$ instantiated by the tuple $\tilde{z}$. An output-prefixed process $\overline{x}\langle\tilde{y}\rangle.P$ sends the tuple $\tilde{y}$ along x and then behaves like P. The matching $[x = y]P$ is used to test for the equality of the names x and y. The restriction construct $\boldsymbol{\nu}\, x\, P$ makes the name x local to P; thus x becomes a new, unique name, distinct from all those external to P. Sum and parallel composition are the same operators as for CCS, the former to express non-determinism, the latter to run two processes in parallel. In the sum, I represents the countable indexing set (I may be infinite in some proofs in Chapter 3). When I is empty, we get the inactive process, written as $\mathbf{0}$. Sometimes we abbreviate $\alpha.\mathbf{0}$ as α. As usually, $+$ is taken to represent binary sum.

Following Milner [Mil91], in the sums we only admit guarded processes, i.e. processes whose outermost operator is prefixing. There are a number of reasons for preferring guarded sums. For the purpose of this thesis, perhaps the most important is that they smooth the comparison between higher-order and first-order processes that we tackle in Chapter 5. Guarded sums also simplify the reduction semantics of Section 2.2.1, and are easier to implement. Furthermore, usually in process algebras guarded sums are necessary to make a number of well-known equivalences, congruences w.r.t. the sum operator. Last but not least, they are justified by practical applications, which show that they give all needed expressive-

[1] To be formal, we could say that an agent comes equipped with a set of defining equations for the constants appearing in it. Then when compositing two agents one should make sure that there is consistency between the two corresponding sets of equations, i.e. a common constant symbol should have the same definition.

ness. We could have been more permissive and allow restrictions and matchings on the top of the outermost prefix of the summands of $\sum_{i\in I}\alpha_i.P_i$; this would have not affected the validity of the results about π-calculus and Higher-Order π-calculus which we shall present in the following chapters. Our choice makes the syntax of the language simpler. (The constrain is not serious for restriction, since it can just be "pulled out"; for instance, instead of $(\boldsymbol{\nu}\, x\, P) + Q$, we can write $(\boldsymbol{\nu}\, x)(P + Q)$, provided x is not free in Q.)

Constants are employed to represent processes with infinite behaviour, for in the definition $(\tilde{x})P$ of a constant D there might be calls to other constants, including D itself. The expression $(\tilde{x})P$ is like a procedure, in which $\tilde{x}$ represents the parameters. Of course, then in an *application* $D\langle\tilde{y}\rangle$, tuple $\tilde{y}$ must be of the same length as $\tilde{x}$; this will be assured by the use of sorts.

If $D \stackrel{def}{=} (\tilde{x})P$ and $\tilde{x}$ is non-empty, then D and $(\tilde{x})P$ are called *abstractions*. Abstractions and processes are *agents*. We use F, G, E to range over abstractions, and A to range over agents. Milner [Mil91] uses abstractions not only in the definition of constants, but also in the construction of processes, for $a(\tilde{x}).P$ is written as $a.(\tilde{x})P$. Moreover by symmetry, $\overline{a}\langle\tilde{x}\rangle.P$ is replaced by $\overline{a}.\langle\tilde{x}\rangle P$, where $\langle\tilde{x}\rangle P$ is called *concretion*. Milner actually goes well beyond this, by allowing for instance the parallel composition of abstractions and concretions and by conceding (in the structural congruence relation) to push restrictions inside them. A definite advantage of his approach is its elegance. Above all, it makes explicit that the binders of abstractions and input prefixes are inherently the same. With our choice the calculus is reduced to its essential parts, and in the semantics, processes are the only kind of agents we have to deal with. This simplifies the treatment of the calculus (we have fewer structural congruence rules) and the reasoning (for example the proof of bisimulations). Moreover, our separation between action and continuation, as distinct moments in the evolution of a process, better agrees with the forms of transition systems and bisimulations we have adopted, and which will be introduced in the following sections.

The operators $a(\tilde{b}).P$, $\boldsymbol{\nu}\,\tilde{b}\,P$ and $(\tilde{b})P$ bind all free occurrences of the names $\tilde{b}$ in P. These binders give rise in the expected way to the definitions of *free* and *bound* names of an agent A, respectively $fn(A)$ and $bn(A)$. Notice in particular that if A is $D\langle\tilde{x}\rangle$, then then $fn(A) = \tilde{x}$ and $bn(A) = \emptyset$. We say that a name x *appears* in A if $x \in fn(A)$ or $x \in bn(A)$. A first-order substitution, or name substitution, is a function from names to names. We write $\{\tilde{y}/\tilde{x}\}$ (where $\tilde{x}$ is a vector of distinct names) for the substitution which maps the x_i-th name in $\tilde{x}$ to the y_i-th name in $\tilde{y}$, and maps all names not in $\tilde{x}$ to themselves. The definitions of substitution and α-conversion on agents are standard, with renaming possibly involved to avoid capture of free names. We only present the definition of substitution. We write $\sigma(a)$ for the name onto which the substitution σ maps a; similarly, $\sigma(\tilde{x})$ and $\sigma(fn(A))$ are the vector and the set of names obtained by applying σ to each element of $\tilde{x}$ and $fn(A)$.

Definition 2.1.1 (first-order substitution) *The effect of the first-order substitution σ on the agent A, written $A\sigma$, is defined inductively on A as below. A substitution does not modify bound names; to avoid that a name free in A become bound in $A\sigma$ we assume that the bound names of A have been previously α-converted to fresh names, so that $bn(A) \cap \sigma(fn(A)) = \emptyset$.*

$$
\begin{aligned}
(D\langle\tilde{x}\rangle)\sigma &= D\langle\sigma(\tilde{x})\rangle \\
((\tilde{x})P)\sigma &= (\tilde{x})(P\sigma) \\
(\textstyle\sum_{i\in I} P_i)\sigma &= \textstyle\sum_{i\in I}(P_i\sigma) \\
(a(\tilde{x}).P)\sigma &= \sigma(a)(\tilde{x}).(P\sigma) \\
(\overline{a}\langle\tilde{x}\rangle.P)\sigma &= \overline{\sigma(a)}\langle\sigma(\tilde{x})\rangle(P\sigma) \\
(P_1 \mid P_2)\sigma &= (P_1\sigma) \mid (P_2\sigma) \\
(\boldsymbol{\nu}\, x\, P)\sigma &= \boldsymbol{\nu}\, x\, (P\sigma) \\
([a = b]P)\sigma &= [\sigma(a) = \sigma(b)](P\sigma)
\end{aligned}
$$

□

A *context* is a term with a single hole $[\cdot]$ in it. We use $C[\cdot]$ to represent a context; then $C[A]$ is the process obtained by replacing $[\cdot]$ with A.

In the following we shall be working modulo α-conversion and we write $A = A'$ if A and A' are α-convertible. We adopt the following precedence among syntactic forms, in decreasing order:

1. application,
2. substitution,
3. restriction, prefixing, matching, replicator (see below),
4. parallel composition,
5. sum,
6. abstraction.

For instance, $\alpha.D\langle\tilde{x}\rangle \,|\, \boldsymbol{\nu}\, y\, P$ means $\big(\alpha.(D\langle\tilde{x}\rangle)\big) \,|\, \boldsymbol{\nu}\, y\, P$. Occasionally, we use $\prod_{i=1}^{n} P_i$ as abbreviation for $P_1 \,|\, \ldots \,|\, P_n$. With reference to the previous scale, symbol $\prod$ has precedence (3); thus $\prod_{i=1}^{n} P_i \,|\, Q$ means $(\prod_{i=1}^{n} P_i) \,|\, Q$.

The *monadic* π-calculus is obtained from the polyadic one when only tuples of length one are allowed. We can see a superiority of the polyadic over the monadic. First, when sorts are employed the former is more expressive than the latter [Mil90b] (at least in the present formulation of sorts, as in Section 2.1.2). Secondly, even if without sorts in the monadic calculus a translation from the polyadic to the monadic is given in [Mil91, Section 3.1], it remains unclear whether this truly respects the equivalences in the two versions. Thirdly, the use of tuples arises naturally in many applications. Finally, it seems that the theory for the monadic is always generalisable straightaway to the polyadic.

Other process expressions

There are two other forms of process expressions, semantically derivable in our syntax, which we shall sometimes use. The *replication* $!P$, whose syntax deliberately recalls the "of course" connective of linear logic [Gir87], intuitively represents

$P \mid P \ldots$, i.e. an unbounded number of copies of P in parallel. Replication sometimes is included in the π-calculus in place of constants [Mil91]. In fact, it is easy to code it up using constants. And if the definition $(\widetilde{x})P$ of each constant has the parameter $(\widetilde{x})$ finite and uses a finite number of constants (i.e. there is no infinite chain $D_1, \ldots, D_n, \ldots$ s.t. P invokes D_1 and the definition of each D_i invokes a D_j with $j > i$), then also replication can encode constants (see [Mil91, Section 3]). We preferred to take recursive definition as primitive, because we need constants defined in terms of an infinity of constants in some proofs on barbed bisimulation. Moreover, constants seem more fundamental, for their simulation using replication may not be possible in a true concurrency setting [Sanb].

The *silent* prefixing $\tau.P$ represents a process which can evolve to P without requiring communication with the environment. It can be written as $\boldsymbol{\nu}\, a\left(\overline{a} \mid a.P\right)$, where a is not free in P. We anticipate here the rules describing the formal behaviour of these operators, which accompany those for the other operators which will be given in Table 2–1:

$$\text{REP: } \frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'} \qquad \text{TAU: } \tau.P \xrightarrow{\tau} P$$

2.1.2 Sorting

All realistic systems which have been described with the π-calculus seem to obey some discipline in the use of names. As Milner says in [Mil91], it is much as for the λ-calculus, which is hardly ever used freely, i.e. without an implicit or explicit type discipline. The introduction of sorts and sortings into the π-calculus intends to make this name discipline explicit. In the polyadic π-calculus, sorts are also essential to avoid disagreement in the arities of tuples carried by a given name, or to be used by a given constant. Let us briefly review now the definition of sorts and sortings.

Names are partitioned into a collection of *subject sorts* — possibly infinite many of them — with the condition that that each term of the language leaves unused un infinite number of names for each sort, to allow α-conversion. We write

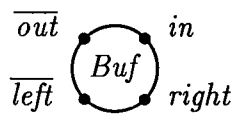
$x : s$ to mean that x belongs to the subject sort s. We also write $x : y$ to mean that names x and y have the same sort. These notations are extended to tuples componentwise. Then *object sorts* are (possibly infinite) sequences over subject sorts, such as $(s_1, \ldots, s_n)$ or (s); () denotes the empty object sort. We use S to range over object sorts. Finally, a *sorting* is a function Ob mapping each subject sort to a finite object sort. Typically, we write $s \mapsto (\tilde{s}) \in Ob$, (or simply $s \mapsto (\tilde{s})$ if Ob is clear), to mean that Ob assigns the object sort $(\tilde{s})$ to s. We also use the compact notation $a : s' \mapsto (\tilde{s})$ to mean that $a : s'$ and $s' \mapsto (\tilde{s})$. Intuitively a sorting just describes the sort of the name-vectors which a subject sort can carry. For instance, by assigning the object sort (s_1, s_2) to the subject sort s, one forces the object part of any name in s to be a pair whose first component is a name of s_1 and whose second component is a name of s_2. Thus CCS and the monadic unsorted π-calculus can be derived by imposing the sortings $\{\text{NAME} \mapsto ()\}$ and $\{\text{NAME} \mapsto (\text{NAME})\}$ respectively, in which all names belong to the same subject sort NAME.

We say that an agent A *respects* a sorting Ob (or is *well-sorted* for Ob) if all prefixes, matchings and applications in A obey the discipline given by Ob, in the following sense. A prefix $a(\tilde{x}).P$ with $a : s' \mapsto (\tilde{s})$ respects Ob if $\tilde{x} : \tilde{s}$. Similarly, a matching $[x = y]P$ respects Ob if $x : y$. Finally, to say when an application respects Ob, we first assign an object sort to agents: Processes take the sort (), whereas if $D \stackrel{def}{=} (\tilde{x})P$ and $\tilde{x} : \tilde{s}$, then D, and $(\tilde{x})P$ take the sort $(\tilde{s})$. Now, the requirement is that in an application $D\langle\tilde{y}\rangle$, if $\tilde{y} : \tilde{s}$ then it must be that $D : (\tilde{s})$. Throughout the thesis we suppose that all agents considered respect some sorting Ob.

Similarly to the notation for names, we write $A : A'$ to mean that A and A' are agents with the same sort.

Example 2.1.2 (from [Mil90b]: A cut-out buffer) We want to represent a chain of identical buffers in which any empty buffer can cut itself out by providing its left neighbour with access to its right neighbour. For simplicity we require

that in this case the left neighbour must be empty too. Pictorially, a buffer is represented as follows:



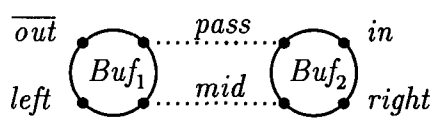
Buf receives a single value at *in* and retransmits it at *out*; ports *right* and *left* are used for the cut-out operations.

$$Buf \stackrel{def}{=}$$
$$(in,\ out,\ left,\ right)\ \big(in\,(x).\overline{out}\langle x\rangle.Buf\langle in,\ out,\ left,\ right\rangle\ +\ \overline{left}\langle in,\ right\rangle\ +$$
$$right\,(newin,\ newright).Buf\langle newin,\ out,\ left,\ newright\rangle\big)$$

The abstraction on the communication ports allows an elegant description of the chaining of two buffers Buf_1 and Buf_2:

$$Buf_1 \frown Buf_2 \stackrel{def}{=}$$
$$(in,out,left,right)(\boldsymbol{\nu}\ pass,mid)\big(Buf_1\langle pass,out,left,mid\rangle \,|\, Buf_2\langle in,pass,mid,right\rangle\big)$$

which can be visualised as



where the dotted lines indicate a restriction on the corresponding ports. The cut-out of Buf_2 is obtained via an interaction at *mid* and yields a process which is structurally congruent (as defined in the next section) to $Buf_1\langle in,out,left,right\rangle$.

In this example, the names involved are given quite different tasks. This is reflected in the sorting Ob which can be assigned. Denoting with S the object sort of the name x (we do not have information for S), we have:

$$Ob = \begin{cases} x : s_1 \ \mapsto\ S \\ in,out,\ pass : s_2 \ \mapsto\ (s_1) \\ left,right,mid : s_3 \ \mapsto\ (s_2,s_3) \end{cases}$$

The sort of the agent identifier *Buf* is (s_2, s_2, s_3, s_3). □

2.2 Operational Semantics of π-calculus

Traditionally, the operational semantics of a process algebra is given in terms of a *labelled transition system* describing the possible evolutions of a process. This contrasts with what happens in *term rewriting systems*, as based on an *unlabelled reduction system*. In the λ-calculus, probably the best known term-rewriting system, what makes a reduction system possible is that two terms having to interact are naturally in contiguous positions. This is not the case in process calculi, where interaction is not dependent upon physical contiguity. To put this another way, a *redex* of a λ-term is a subterm, while a "redex" in a process calculus is distributed over the term.

A guideline for the definition of reduction systems in process algebras is offered by Milner [Mil90a,Mil91], inspired by Berry and Boudol's Chemical Abstract Machine [BB92]. In this technique, axioms for a structural congruence relation are introduced prior to the reduction system, in order to break a rigid, geometrical vision of concurrency; then reduction rules can easily be presented in which redexes are indeed subterms again.

The interpretation of the operators of the language comes out neatly with reduction semantics, due to the compelling naturalness of each structural congruence and reduction rule. This is not quite the case in the labelled semantics, at least for process algebras expressing mobility: The manipulation of names and the side conditions in the rules are non-trivial and this makes delicate the justification of the choices made. However, if the reduction system is available, the correctness of the labelled transition system can be shown by proving the correspondence between the two systems.

On the other hand, the advantages of labelled semantics appear later, when reasoning with agents. In reduction semantics, the behaviour of a process is understood relatively to a context in which it is contained and with which it interacts. Instead, with labelled semantics every possible communication of a process can be determined in a direct way. This allows us to get simple characterisations of be-

havioural equivalences. Moreover with labelled semantics the proofs benefit from the possibility of reasoning in a purely structural way.

The respective advantages of the two semantics become even more noticeable in a higher-order calculus, like HOπ. The conclusion is that both semantics are useful and that they integrate and support each other.

2.2.1 Reduction semantics

Structural congruence, written $\equiv$, is the smallest congruence over the class of π-calculus processes which satisfies the following rules:

1. $P \equiv Q$ if P is α-convertible to Q;
2. $\sum_{i\in I} \alpha_i P_i \equiv \sum_{j\in J} \alpha_j P_j$ if J is a permutation of I;
3. abelian monoid laws for $|$: $P \,|\, Q \equiv Q \,|\, P$; $P \,|\, (Q \,|\, R) \equiv (P \,|\, Q) \,|\, R$; $P \,|\, \mathbf{0} \equiv P$;
4. $\boldsymbol{\nu}\, x\, \mathbf{0} \equiv \mathbf{0}$; $\boldsymbol{\nu}\, x \boldsymbol{\nu}\, y\, P \equiv \boldsymbol{\nu}\, y \boldsymbol{\nu}\, x\, P$; $(\boldsymbol{\nu}\, x\, P) \,|\, Q \equiv \boldsymbol{\nu}\, x\, (P \,|\, Q)$; if $x \notin fn(Q)$;
5. $[x = x]P \equiv P$;
6. if $D \stackrel{def}{=} (\tilde{x})P$, and $\tilde{x} : \tilde{y}$, then $D\langle\tilde{y}\rangle \equiv P\{\tilde{y}/\tilde{x}\}$.

We do not have a precise definition of what should determine the set of rules for $\equiv$. We can however try to indicate some criteria (or general principles) for this:

1. The rules should be valid in any reasonable behavioural equivalence;
2. they should provide some intuition for the operators;
3. they should allow a simple statement of the reduction rules and should be needed when proving the correspondence between reduction and labelled semantics;
4. there should be only a few of them.

On some specific rules however, these principles might not provide a univocal guidance. For instance the rule $\boldsymbol{\nu}\, x\, \mathbf{0} \equiv \mathbf{0}$ is not necessary to state the reduction rules and prove correspondence between reduction and labelled semantics and hence might be canceled; yet we have preferred to keep it because it seems to be strongly suggested by criteria (1) and (2) (a similar dispute arises when considering whether to add the rule $\sum_{i\in I} P \equiv P$). The system that we have given follows Milner's in [Mil91] and we think it achieves a good compromise among the above listed principles.

Now the *reduction rules*, expressing the notion of interaction:

$$\text{COM: } (\cdots + x(\tilde{y}).P) \mid (\cdots + \overline{x}\langle\tilde{z}\rangle.Q) \longrightarrow P\{\tilde{z}/\tilde{y}\} \mid Q$$

$$\text{PAR: } \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \text{RES: } \frac{P \longrightarrow P'}{\boldsymbol{\nu}\, x\, P \longrightarrow \boldsymbol{\nu}\, x\, P'}$$

$$\text{STRUCT: } \frac{Q \equiv P \;\; P \longrightarrow P' \;\; P' \equiv Q'}{Q \longrightarrow Q'}$$

Example 2.2.1 Suppose $x \notin fn(P, Q)$. Then

$$\begin{aligned} \boldsymbol{\nu}\, x\, (\overline{a}\langle x\rangle.P) \mid R \mid a(y).Q &\equiv \\ \boldsymbol{\nu}\, x\, (\overline{a}\langle x\rangle.P \mid a(y).Q) \mid R &\longrightarrow \\ \boldsymbol{\nu}\, x\, (P \mid Q\{x/y\}) \mid R &\equiv \\ P \mid R \mid \boldsymbol{\nu}\, x\, (Q\{x/y\}) & \end{aligned}$$

□

Note that reduction is not allowed underneath a prefix. Thus $\overline{a}\langle x\rangle.P|a(y).Q \longrightarrow P \mid Q\{x/y\}$ but $\alpha.\big(\overline{a}\langle x\rangle.P \mid a(y).Q\big) \not\longrightarrow \alpha.\big(P \mid Q\{x/y\}\big)$. Prefixing is the construct introduced to represent sequentialisation. Hence nothing underneath a prefix should occur before this has fired.

2.2.2 Labelled transition semantics

In the labelled semantics we present, the bound names of an input are instantiated as soon as possible, namely in the rule for input. It is therefore an *early* transition system, as opposed to a *late* transition system – for instance the one used

in [MPW92] – where such an instantiation is done later, in the rule for communication. We chose the former because the bisimulation it supports, called early bisimulation, has some attractive peculiarities, discussed in Section 2.6, and it is the bisimulation which we shall utilise throughout the thesis.

There are three possible forms for an action. Besides the silent-action τ representing interaction, we have:

$P \xrightarrow{x(\tilde{y})} P'$	input action; x is the name at which it occurs, $\tilde{y}$ is the tuple of names which are received
$P \xrightarrow{(\nu\, \tilde{y'})\overline{x}\langle\tilde{y}\rangle} P'$	output action; it is the output of the names $\tilde{y}$ at x. It always holds that $\tilde{y'} \subseteq \tilde{y} - x$; in fact $\tilde{y'}$ represents private names which are emitted from P, i.e. carried out from their current scope (*scope extrusion*)

In both cases, x is the *subject* and $\tilde{y}$ is the *object* part of the action. In the input the subject is *positive*, whereas in the output it is *negative*. Note the different brackets in input prefixes and input actions (round in the former, angled in the latter). This is to recall that in input prefix $x(\tilde{y})$, the names $\tilde{y}$ are *binders* (waiting to be instantiated) whereas in an input action $x\langle\tilde{y}\rangle$ they represent *values* (with which the binders have been instantiated) — in the same way as the names $\tilde{y}$ are values in an output $\overline{x}\langle\tilde{y}\rangle$.

We use μ to represent the label of a generic action (not to be confused with α, which ranges over prefixes). Given an action μ, the bound and free names of μ, respectively written $bn(\mu)$ and $fn(\mu)$, are defined as follows:

μ	$fn(\mu)$	$bn(\mu)$
$x\langle\tilde{y}\rangle$	$x, \tilde{y}$	$\emptyset$
$(\nu\, \tilde{y'})\overline{x}\langle\tilde{y}\rangle$	$x, \tilde{y} - \tilde{y'}$	$\tilde{y'}$
τ	$\emptyset$	$\emptyset$

The names of μ, briefly $n(\mu)$, are $bn(\mu) \cup fn(\mu)$.

The transition system is presented in Table 2–1. We have omitted the symmet-

$$\text{ALP:}\ \frac{P' \xrightarrow{\mu} Q \qquad P \text{ and } P' \text{ are } \alpha\text{-convertible}}{P \xrightarrow{\mu} Q}$$

$$\text{OUT:}\ \overline{x}\langle\widetilde{y}\rangle.P \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} P \qquad \text{INP:}\ x(\widetilde{y}).P \xrightarrow{x(\widetilde{z})} P\{\widetilde{z}/\widetilde{y}\},\ \text{if } \widetilde{z} : \widetilde{y}$$

$$\text{SUM:}\ \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \text{PAR:}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ bn(\mu) \cap fn(Q) = \emptyset$$

$$\text{COM:}\ \frac{P \xrightarrow{(\nu\,\widetilde{y'})\overline{x}\langle\widetilde{y}\rangle} P' \quad Q \xrightarrow{x(\widetilde{y})} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu}\,\widetilde{y'}\,(P' \mid Q')}\ \widetilde{y'} \cap fn(Q) = \emptyset$$

$$\text{MATCH:}\ \frac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'} \qquad \text{CONST:}\ \frac{P\{\widetilde{y}/\widetilde{x}\} \xrightarrow{\mu} P'}{D\langle\widetilde{y}\rangle \xrightarrow{\mu} P'}\ \text{if } D \stackrel{def}{=} (\widetilde{x})P$$

$$\text{RES:}\ \frac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}\,x\,P \xrightarrow{\mu} \boldsymbol{\nu}\,x\,P'}\ x \notin n(\mu) \qquad \text{OPEN:}\ \frac{P \xrightarrow{(\nu\,\widetilde{y'})\overline{x}\langle\widetilde{y}\rangle} P'}{\boldsymbol{\nu}\,x\,P \xrightarrow{(\nu\,x,\widetilde{y'})\overline{x}\langle\widetilde{y}\rangle} P'}\ x \neq z, x \in \widetilde{y} - \widetilde{y'}$$

Table 2–1: π-calculus's labelled transition system

ric versions of rules SUM, PAR, and COM. Note the presence of the rule ALP: This allows us to define the other transitions up-to redenomination of the bound names; it also allows us to avoid some tedious side conditions when using the transition system. The treatment of restriction is delicate and deserves some explanation.

- In the rule RES the side condition prevents transitions like

$$(\boldsymbol{\nu}\,b)a(x).P \xrightarrow{a\langle b\rangle} \boldsymbol{\nu}\,b\,P\{b/x\}$$

which would violate the static binding assumed for restriction. This does not mean however that name b cannot be received. To allow this, we have just to use some α-conversion:

$$(\boldsymbol{\nu}\,b)a(x).P = (\boldsymbol{\nu}\,c)a(x).P\{c/b\} \xrightarrow{a\langle b\rangle} \boldsymbol{\nu}\,c\,P\{b/x\}$$

- It would not be enough for the side condition in the RES rule to say $x \notin fn(\mu)$; for, if so, then one could derive the transition

$$\boldsymbol{\nu}\, y \left((\boldsymbol{\nu}\, y)\overline{x}\langle y \rangle.P \right) \xrightarrow{(\boldsymbol{\nu}\, y)\overline{x}\langle y \rangle} \boldsymbol{\nu}\, y\, P$$

  in which free occurrences of y in P are captured by the wrong binder.

- The OPEN rule is the one which implements scope extrusion.

- The side condition in the rule COM prevents interactions like

$$\boldsymbol{\nu}\, y\, (\overline{a}\langle y \rangle.P) \mid \left(a(x).Q \mid \overline{y}.\mathbf{0} \right) \xrightarrow{\tau} \boldsymbol{\nu}\, y \left(P \mid (Q\{y/x\} \mid \overline{y}.\mathbf{0}) \right)$$

  in which the free occurrence of y in $\overline{y}.\mathbf{0}$ has become bound after the interaction.

2.2.3 Correspondence between the two semantics

It can be shown that modulo structural congruence, the relation $\longrightarrow$ of reduction semantics is exactly the relation $\xrightarrow{\tau}$ of labelled semantics. Moreover, it is also possible to establish what the observable actions of the latter corresponds to in the former. Roughly speaking, they are represented by those prefixes which can be pulled out (nearly) to the top by means of $\equiv$. For instance, if $P \xrightarrow{\overline{x}\langle \widetilde{y} \rangle} P'$ then one can show that Q, T, R exist s.t. $P \equiv \boldsymbol{\nu}\, \widetilde{z} \left((\overline{x}\langle \widetilde{y} \rangle.Q + T) \mid R \right)$, with $\widetilde{z} \cap \{x, \widetilde{y}\} = \emptyset$ and $P' \equiv \boldsymbol{\nu}\, \widetilde{z}\, (Q \mid R)$, and with the subterm $+T$ possibly missing. Precise results by Milner and Turner on the topic are in [Mil90a,Tur91], for the monadic case.

Because of this agreement, in the following chapters — where we always work with the labelled transition system — we shall often abbreviate $\xrightarrow{\tau}$ as $\longrightarrow$.

Higher Order π-calculus

2.3 Syntax of HOπ

In the π-calculus only object sorts of the form $(\widetilde{s})$ are allowed. The sortings so obtained are first-order, as indicated by the level of bracket nesting, which is limited to one. The *Higher-Order π-calculus* is — essentially — derived by dropping this limitation. Thus one may enforce processes to be communicated along a name x by declaring $x : s \mapsto (())$. Then an "executor", which receives a process at x and executes it, can be written as $x(X).X$; when put in parallel with $\overline{x}\langle P\rangle.Q$, it gives rise to the interaction

$$\overline{x}\langle P\rangle.Q \mid x(X).X \longrightarrow Q \mid P$$

Let us see more interesting examples, involving also higher-order abstractions.

Example 2.3.1 In [Mil91], Milner shows how to encode numbers in the π-calculus. If $\overline{y}^n$ is taken to mean $\underbrace{\overline{y}.\cdots.\overline{y}}_{n\ times}$, and $y, z : s \mapsto ()$, then the natural number n is encoded as follows:

$$[n] \stackrel{def}{=} (y,z)\overline{y}^n.\overline{z} : (s,s)$$

We want now to write an agent *Plus* capable of performing the sum of two numbers. Consider the process $\boldsymbol{\nu}\, x\left([n]\langle y,x\rangle \mid x.[m]\langle y,z\rangle\right)$: If we abstract from possible τ-actions, this behaves exactly like $[n+m]\langle y,z\rangle$. Accordingly, if X and Y are variables of the same sort as numerals, we can define

$$\mathit{Plus} \stackrel{def}{=} (X,Y)(y,z)\left(\boldsymbol{\nu}\, x\,(X\langle y,x\rangle \mid x.Y\langle y,z\rangle)\right) : ((s,s),(s,s),s,s)$$

Plus is a higher-order abstraction (it cannot be written in π-calculus), because it abstracts from X and Y which are agent-variables; this is also indicated by the bracket nesting in the definition of *Plus*, which is greater than one. The machinery can be iterated, for instance by defining abstractions on variables of the same sort as *Plus* and so forth, progressively increasing the order of the resulting agents.

Now an adder which recursively takes two integers at ports a_1, a_2 and outputs their sum at a_3 can be represented as:

$$Add \stackrel{def}{=} a_1(X).a_2(Y).\overline{a}_3\Big\langle Plus\,\langle X,Y\rangle\Big\rangle.Add \qquad \square$$

Example 2.3.2 In Example 2.1.2, a chaining operator $\frown$ connects two buffers with sort $(\widetilde{s})$, for $\widetilde{s} = s_2, s_2, s_3, s_3$. In HO$\pi$, if $X, Y : (\widetilde{s})$, then we can set

$$\frown \stackrel{def}{=}$$

$$(X,Y)(in,out,left,right\,)(\boldsymbol{\nu}\ pass,mid\,)(X\langle pass,out,left,mid\,\rangle \mid Y\langle in,pass,mid,right\,\rangle)$$

which thus is an agent of sort $((\widetilde{s}),(\widetilde{s}),\widetilde{s})$. $\square$

At this point, the formal modifications of π-calculus syntax and semantics required to obtain those for HOπ should start becoming clear. We consider this below. The notation already introduced for the π-calculus will not be repeated.

The language

Let *Var* be a set of agent-variables, ranged over by X, Y. To simplify the notation, we use K to stand for an agent or a name and U to stand for a variable or a name. There are only two modifications to bring into the syntax of the π-calculus. First, tuples over K and U must replace pure name tuples in prefixing, abstractions and applications. Secondly, variable application should be allowed too, so that an abstraction received as input can be provided with the appropriate arguments. In this thesis, we shall only be interested in well-sorted expressions. We shall defined well-sorted expressions inductively, using a set of formation rules, after presenting the sorting discipline. For ease of reference, we first give the grammar for (unsorted) processes and agents. The grammar for processes is:

$$\begin{array}{lcl} P & :: & \sum_{i\in I}\alpha_i.P_i \;\mid\; P_1 \mid P_2 \;\mid\; \boldsymbol{\nu}\, x\, P \;\mid [x=y]P \;\mid\; D\langle\widetilde{K}\rangle \;\mid\; X\langle\widetilde{K}\rangle \\ \alpha & :: & \overline{x}\langle\widetilde{K}\rangle \;\mid\; x(\widetilde{U}) \end{array}$$

where defining equations for constants are of the form $D \stackrel{def}{=} (\widetilde{U})P$. Remember that K may be an agent; hence it may be a process, but also an abstraction of

arbitrary high order. The grammar for agents is:

$$A \;::\; (\tilde{U})P \;\mid\; (\tilde{U})X\langle\widetilde{K}\rangle \;\mid\; (\tilde{U})D\langle\widetilde{K}\rangle$$

(Note also that a variable X and a constant D are agent, corresponding to the cases in which $\tilde{U}$ and $\widetilde{K}$ are empty).[1] The same assumptions as for π-calculus apply: Thus, the tuple of binders U is made of distinct elements; and the brackets $()$ and $\langle\rangle$ are intended to be omitted when the tuple inside is empty. All tuples are required to be finite, except those being the parameters of the constants, which may contain an infinite number of names so to express agents which use an infinity of different names (but the number of agent-variables and of agents in these tuples remain finite). A variable X which is not underneath some input prefix $x(\tilde{U})$ or an abstraction $(\tilde{U})$ with $X \in \tilde{U}$, is *free*. A agent possibly containing free variables is *open*. We denote by $fv(A)$ the set of free variables of the agent A. We use $HO\pi^\circ$ to denote the class of open agents and $HO\pi$ for the class of *closed* agents. Note that every subterm of a closed agent has a finite number of free variables. Indeed, an agent obtained from the given grammar may have infinite free variables, due to the use of infinite sums; but then it could not be made closed because the input and abstractions binders can only bind a finite number of variables.

Well-sorted expressions

In the HOπ the need for sorts is even more compelling than in π-calculus: Besides the arity question, we also have to avoid any confusion between instantiation to names and to agents as well as instantiation to agents of different order.

W.r.t. the π-calculus case of Section 2.1.2, the difference in the syntax for sorts is that the sequences representing object sorts do not have to be made only of subject sorts; rather, object sorts themselves can appear too. We call subject

[1] There is a difference in the use of applications, $X\langle\widetilde{K}\rangle$ and $D\langle\widetilde{K}\rangle$, in the grammar for processes and for agents. In the former case, $\widetilde{K}$ is supposed to represent the tuple of *all* arguments for X and D; in the latter case, $\widetilde{K}$ represents an *initial* segment of these arguments. This difference should become clear from the formation rules presented later.

sorts and object sorts *element sorts*; we use El to range over element sorts.

$$El \;::\; s \;\mid\; S$$
$$S \;::\; (\widetilde{El})$$

For each object sort S we suppose the existence of an infinite number of variables of sort S, denoted by *Var(S)*. Notice that the special case of second-order sorting $\{\text{NAME} \mapsto (())\}$ corresponds to Thomsen's Plain CHOCS.

The set of formation rules for well-sorted agents is presented in Table 2–2 (we remind that $a : s \mapsto (\widetilde{El})$ means that the name a belongs to the subject sort s and that the object sort of s is $(\widetilde{El})$). An agent A is *well-sorted* for Ob, if we can infer $A : S$, for some S, from the rules in the table. Moreover, if $S = (\widetilde{El})$, for $\widetilde{El}$ non-empty, then A is an *abstraction*, whereas if $\widetilde{El}$ is empty, then A is a *process*. The well-sortness condition restricts the grammar for agents to those expressions in which all prefixings, matchings and applications conform to the given sorting discipline, in the sense that we explained for π-calculus — the difference being that here we have higher-order sorts. As an aside, let us mention an alternative notation for sorts which seems fairly effective in HOπ. Consider the abstraction $G \stackrel{def}{=} (X)F$, for $X : S'$, $F : S$. It really represents a function which takes an argument of sort S' and gives back an argument of sort S. From a function-theoretic point of view, G has type $S' \longrightarrow S$. Following such intuition, we could explicitly introduce the arrow-sort and say that $G : S' \longrightarrow S$. Thus a sort $(El', \widetilde{El})$ is equivalent to $El' \longrightarrow (\widetilde{El})$. For instance, for the agent *Plus* in Example 2.3.1, we would have, for $S = s \longrightarrow s \longrightarrow ()$:

$$Plus : S \longrightarrow S \longrightarrow s \longrightarrow s \longrightarrow ()$$

or also, $Plus : S \times S \longrightarrow s \times s \longrightarrow ()$, using some "uncurrying". In the sequel however, we stick to the original notation.

More on the syntax for well-sorted expressions

Well-sortness allows us to have, in the syntax, only applications in which the term on the left is a constant or a variable: Every expression $A\langle\widetilde{K}\rangle$, if well-sorted,

$$\frac{X \in Var(S)}{X : S} \qquad \frac{\tilde{U} : \widetilde{El} \quad A : ()}{D : (\widetilde{El})} \text{ if } D \stackrel{def}{=} (\tilde{U})A$$

$$\frac{\tilde{U} : \widetilde{El'} \quad A : (\widetilde{El})}{(\tilde{U})A : (\widetilde{El'}, \widetilde{El})} \qquad \frac{\widetilde{K} : \widetilde{El} \quad A : () \quad a : s \mapsto (\widetilde{El})}{\overline{a}\langle\widetilde{K}\rangle.A : ()}$$

$$\frac{\tilde{U} : \widetilde{El} \quad A : () \quad a : s \mapsto (\widetilde{El})}{a(\tilde{U}).A : ()} \qquad \frac{\forall i \;\; A_i : ()}{\sum_{i \in I} A_i : ()}$$

$$\frac{A_1 : () \quad A_2 : ()}{A_1 \mid A_2 : ()} \qquad \frac{A : ()}{\boldsymbol{\nu}\, x\, A : ()}$$

$$\frac{x : y \quad A : ()}{[x = y]A : ()} \qquad \frac{X : (\widetilde{El'}, \widetilde{El}) \quad \widetilde{K} : \widetilde{El'}}{X\langle\widetilde{K}\rangle : (\widetilde{El})}$$

$$\frac{D : (\widetilde{El'}, \widetilde{El}) \quad \widetilde{K} : \widetilde{El'}}{D\langle\widetilde{K}\rangle : (\widetilde{El})}$$

Table 2–2: Formation rules for HOπ

---

can be rewritten as an expression in our syntax by "executing" the applications it contains; for instance from $\big((X)Y\langle X\rangle\big)\langle P\rangle$, we get $Y\langle P\rangle$. This will be made clearer in Lemma 2.3.4, showing the well-definiteness of agent substitutions. The restriction to only constant and variable applications makes the definition of substitution more elaborated but facilitates the proofs in the calculus. However in the thesis we shall often use $A\langle\widetilde{K}\rangle$ as metanotation: For instance, if $F \stackrel{def}{=} (U)P$, then $F\langle K\rangle$ abbreviates $P\{F/U\}$.

We only give the definition of higher-order substitutions, i.e. substitutions among agents: A substitution can always be decomposed into a first-order substitution, i.e. a substitution among names, and into a higher-order substitution; and first-order substitution is defined similarly as for π-calculus. A higher-order substitution $B\{\tilde{A}/\widetilde{X}\}$ represents the simultaneous replacement in B of the X_i's with the A_i's. Since every subterm of a closed HOπ agent has a finite number of free variables, it is enough to consider substitutions in which the tuples $\tilde{A}$ and $\widetilde{X}$ are finite. Such substitutions can be performed sequentially, i.e. replacing one

argument at a time, if previously some α-conversion is used to guarantee that the X_i's do not appear in $\widetilde{A}$. Therefore we only give the definition of unary substitutions. In Definition 2.3.3 below we use the possibility of η-converting the substituting agent A into the form $(\widetilde{U})P$, that is, an abstraction on a process. The transformation $\rhd_\eta$ which does so has the expected definition:

- If $A = (\widetilde{U})P$ then $A \rhd_\eta A$.
- If $A = (\widetilde{U'})X\langle\widetilde{K}\rangle$ with $X\langle\widetilde{K}\rangle : (\widetilde{El})$ and $\widetilde{U} : \widetilde{El}$ and no U_i appears in $\widetilde{K}$, then $A \rhd_\eta (\widetilde{U'}, \widetilde{U})X\langle\widetilde{K}, \widetilde{U}\rangle$.
- Similarly, if $A = (\widetilde{U'})D\langle\widetilde{K}\rangle$ with $D\langle\widetilde{K}\rangle : (\widetilde{El})$ and $\widetilde{U} : \widetilde{El}$ and no U_i appears in $\widetilde{K}$, then $A \rhd_\eta (\widetilde{U'}, \widetilde{U})D\langle\widetilde{K}, \widetilde{U}\rangle$.

We write $\widetilde{K}\sigma$ for the tuple obtained from $\widetilde{K}$ by replacing its i-th element K_i with $K_i\sigma$; obviously, $K_i\sigma = K_i$ if K_i is a name and σ a higher-order substitution.

Definition 2.3.3 (higher-order substitution) *Let B and A be well-sorted agents and $\sigma = \{A/X\}$, for $A : X$. Then the effect of the substitution σ on the agent B, written $B\sigma$, is defined inductively on B as below. We suppose that the names and variables which are bound in B do not appear in A and are different from X.*

$$
\begin{aligned}
(Y\langle\widetilde{K}\rangle)\sigma &= \begin{cases} P\{\widetilde{K}\sigma/\widetilde{U}\} & \text{if } Y = X \text{ and } A \rhd_\eta (\widetilde{U})P \\ Y\langle\widetilde{K}\sigma\rangle & \text{otherwise} \end{cases} \\
(D\langle\widetilde{K}\rangle)\sigma &= D\langle\widetilde{K}\sigma\rangle \\
\big((\widetilde{U})B'\big)\sigma &= (\widetilde{U})(B'\sigma) \\
(\textstyle\sum_i P_i)\sigma &= \textstyle\sum_i (P_i\sigma) \\
\big(a(\widetilde{U}).P\big)\sigma &= a(\widetilde{U}).(P\sigma) \\
\big(\overline{a}\langle\widetilde{K}\rangle.P\big)\sigma &= \overline{a}\langle\widetilde{K}\sigma\rangle.(P\sigma) \\
(P_1 \mid P_2)\sigma &= (P_1\sigma) \mid (P_2\sigma) \\
(\boldsymbol{\nu}\, x\, P)\sigma &= \boldsymbol{\nu}\, x\, (P\sigma) \\
([a = b]P)\sigma &= [a = b](P\sigma)
\end{aligned}
$$

□

Lemma 2.3.4 *Higher-order substitution, as given in Definition 2.3.3, is well-defined and terminating.*

PROOF: We proceed by a nested induction. The external induction is on the depth of the sort $(\widetilde{El})$ of X and A, where the depth of a sort is the maximum level of bracket nesting in the definition of the sort and intuitively, it says how "high-order" the sort is (the depth is finite because the agents of our language may only abstract on a finite number of variables). The internal induction is on the structure of the agent B to which the substitution is applied. The delicate clause is the one for application $Y\langle\widetilde{K}\rangle$, when $Y = X$. We have to ensure that $\tilde{U}$ and $\widetilde{K}$ have the same sort and that the recursive call of substitutions does not degenerate into an infinite loop. The former is immediate: By hypothesis, X and A have the same sort $(\widetilde{El})$; therefore by definition of well-sorted agents, we have $\tilde{U} : \widetilde{El}$ and $\widetilde{K} : \widetilde{El}$. For the latter, we distinguish the case in which the depth of A is one or greater than one. If the depth is one, then $\widetilde{K}$ and $\tilde{U}$ are all names and therefore $P\{\widetilde{K}/\tilde{U}\}$ terminates in one step. Suppose that the depth of $(\widetilde{El})$ is $n+1$. Then each K_i is either a name or a subagent of B whose sort El_i has depth less or equal to n. Therefore, using induction on the structure of B, $\widetilde{K}\sigma$ is terminating, and then, using induction on the depth of the sort, $P\{\widetilde{K}\sigma/\tilde{U}\}$ is terminating. □

2.4 Operational semantics of HOπ

Reduction semantics

The structural congruence rules and the reduction rules for HOπ are the same as for π-calculus. We only have to generalise the structural rule (6) and the rule COM, so that the tuples involved may contain agents; these become:

6. If $D \stackrel{def}{=} (\tilde{U})P$ and $\tilde{U} : \widetilde{K}$, then $D\langle\widetilde{K}\rangle \equiv P\{\widetilde{K}/\tilde{U}\}$.

and

$$\text{ALP: } \frac{P' \xrightarrow{\mu} Q \quad P \text{ and } P' \text{ are } \alpha\text{-convertible}}{P \xrightarrow{\mu} Q}$$

$$\text{OUT: } \overline{x}\langle\widetilde{K}\rangle.P \xrightarrow{\overline{x}\langle\widetilde{K}\rangle} P \qquad \text{INP: } x(\widetilde{U}).P \xrightarrow{x\langle\widetilde{K}\rangle} P\{\widetilde{K}/\widetilde{U}\},\ \text{if } \widetilde{K} : \widetilde{U}$$

$$\text{SUM: } \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \text{PAR: } \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ bn(\mu) \cap fn(Q) = \emptyset$$

$$\text{COM: } \frac{P \xrightarrow{(\nu\,\widetilde{y})\overline{x}\langle\widetilde{K}\rangle} P' \quad Q \xrightarrow{x\langle\widetilde{K}\rangle} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu}\,\widetilde{y}\,(P' \mid Q')}\ \widetilde{y} \cap fn(Q) = \emptyset$$

$$\text{MATCH: } \frac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'} \qquad \text{CONST: } \frac{P\{\widetilde{K}/\widetilde{U}\} \xrightarrow{\mu} P'}{D\langle\widetilde{K}\rangle \xrightarrow{\mu} P'},\ \text{if } D \stackrel{def}{=} (\widetilde{U})P$$

$$\text{RES: } \frac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}\,x\,P \xrightarrow{\mu} \boldsymbol{\nu}\,x\,P'}\ x \notin n(\mu) \qquad \text{OPEN: } \frac{P \xrightarrow{(\nu\,\widetilde{y})\overline{z}\langle\widetilde{K}\rangle} P'}{\boldsymbol{\nu}\,x\,P \xrightarrow{(\nu\,x,\widetilde{y})\overline{z}\langle\widetilde{K}\rangle} P'}\ x \neq z, x \in fn(\widetilde{K}) - \widetilde{y}$$

Table 2–3: HOπ's labelled transition system

$$\text{COM: } (\cdots + x(\widetilde{U}).P) \mid (\cdots + \overline{x}\langle\widetilde{K}\rangle) \longrightarrow P\{\widetilde{K}/\widetilde{U}\} \mid Q$$

Labelled semantics

The generalisation of the labelled transition system requires a little more thought than for the reduction system. The system is represented in Table 2–3. Visible actions become of the form $x\langle\widetilde{K}\rangle$ (input action) or $(\boldsymbol{\nu}\,\widetilde{y})\overline{x}\langle\widetilde{K}\rangle$ (output action). In the latter, it holds that $\widetilde{y} \subseteq fn(\widetilde{K}) - x$; for instance, if $y \in fn(P)$, we have

$$(\boldsymbol{\nu}\,y)\overline{x}\langle P\rangle.Q \xrightarrow{(\nu\,y)\overline{x}\langle P\rangle} Q$$

Here the free occurrences of y in P force a name extrusion, to respect the static binding on restriction.

The same correspondence between the two semantics mentioned for π-calculus holds for HOπ too.

CONVENTION. When performing algebraic manipulations, sometimes we will omit obvious side conditions. For instance, we might write $\boldsymbol{\nu}\, a\, P \mid Q \equiv \boldsymbol{\nu}\, a\, (P \mid Q)$ without recalling that $a \notin fn(Q)$. □

2.5 Some preliminaries about bisimulations

This section is devoted to presenting some well-established theory and some conventions related to the notion of bisimulation, which will be used throughout the thesis.

2.5.1 Strong and weak bisimulations

For all bisimulations we consider there is a *strong* and a *weak* version. In the strong case all actions are treated uniformly. In the weak case, one abstracts away from silent actions, because they represent internal behaviour of processes. For this, the *weak transitions* have to be introduced: First the relation $\Longrightarrow$, the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$; then $\stackrel{\mu}{\Longrightarrow}$ as $\Longrightarrow\stackrel{\mu}{\longrightarrow}\Longrightarrow$; finally $\stackrel{\widehat{\mu}}{\Longrightarrow}$ as $\stackrel{\mu}{\Longrightarrow}$ if $\mu \neq \tau$, and as $\Longrightarrow$ if $\mu = \tau$.

In the thesis, we only define the strong version of a bisimulation; the weak one can be obtained from the former in a completely standard way, which we are going to describe. Let $<>$ be the symbol chosen for the strong bisimulation. The definition of $<>$-bisimulation will be, approximately, of the following form:

> A relation $\mathcal{R}$ is a $<>$-*simulation* if whenever $(P, Q) \in \mathcal{R}$ and (*)
> $P \stackrel{\mu}{\longrightarrow} P'$, then Q', μ' exist s.t. $Q \stackrel{\mu'}{\longrightarrow} Q'$, $\mathcal{B}(\mu, \mu')$ and $\forall C \in \mathfrak{C}$,
> if $P'' = f(C, \mu, P')$ and $Q'' = f(C, \mu', Q')$, then $P''\ \mathcal{R}\ Q''$.
> $\mathcal{R}$ is a $<>$-*bisimulation* if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are $<>$-simulation.

Here, $\mathfrak{C}$, $\mathcal{B}$ and f are special predefined class, predicate and function, respectively. Often, the quantification on $\mathfrak{C}$ is missing and hence the definitions of the processes

P'' and Q'' only depend upon the actions μ and μ' and the processes P' and Q'. When $\mathfrak{C}$ is present, it will represent a class of contexts. Then we define

$$<> = \bigcup\{ \mathcal{R} \ : \ \mathcal{R} \ \text{ is a } <>\text{-bisimulation}\}$$

Indeed $<>$ will always represent the *largest* $<>$-bisimulation. Now the weak version $\ll\gg$ of $<>$ is obtained from $(*)$ by simply replacing the symbol $<>$ with $\ll\gg$ and the transition $Q \xrightarrow{\mu'} Q'$ with $Q \stackrel{\widehat{\mu'}}{\Longrightarrow} Q'$. (In addition to this one might also replace the transition $P \xrightarrow{\mu} P'$ with $P \stackrel{\mu}{\Longrightarrow} P'$. As usual in bisimulations for process algebras, this would not affect the $\ll\gg$-bisimulation derived, but the definition would be more difficult to verify.)

2.5.2 Congruences

Most of the bisimulations we shall consider are preserved by all operators except prefixing when it involves the input of names. To obtain the full congruence it is enough to require bisimilarity over all first-order substitutions (as done also in [MPW92]). Thus, the congruence $<>^{\mathrm{c}}$ of the bisimulation $<>$ is defined as follows:

$$A<>^{\mathrm{c}}A' \text{ if for all } \tilde{x},\tilde{y} \text{ with } \tilde{x} : \tilde{y},\ A\{\tilde{x}/\tilde{y}\}<>A'\{\tilde{x}/\tilde{y}\}.$$

In our symbology for the behavioural equivalences, the appearance of a superscript "c" means that a relation is a congruence over all operators.

2.5.3 Bisimulations up-to

Definition $(*)$ in Section 2.5.1 requires that $P'' \mathcal{R} Q''$. If we want more flexibility, so to reduce the size of the relations to exhibit for proving a $<>$-bisimilarity, we might try to exploit a *bisimulation up-to* technique ([Mil89,MPW92,SM92]). Obviously, before using any of these techniques, its soundness must be guaranteed, by proving that the relations it defines are contained in $<>$.

In the most common up-to techniques, the closure of $\mathcal{R}$ is achieved modulo some supporting privileged relation $\bowtie$. The definition of strong <>-bisimulation up-to $\bowtie$ is obtained from $(*)$ by replacing $P'' \mathcal{R} Q''$ with $P'' \bowtie \mathcal{R} \cdot \bowtie Q''$. Since <> is a strong relation, $\bowtie$ must be strong too; often it is taken to be <> itself. When adapting this technique to the weak case, some care is necessary. If $\bowtie$ is a "weak" relation, it is not sound in general simply to replace the transition $Q \xrightarrow{\mu'} Q'$ with $Q \stackrel{\widehat{\mu'}}{\Longrightarrow} Q'$ ([SM92]); in addition to this, at least one of the two following requirements is necessary:

(a) use $\bowtie$ only on the right of $\mathcal{R}$, i.e. ask $P'' \mathcal{R} \bowtie Q''$,

(b) replace also the transition $P \xrightarrow{\mu} P'$ with $P \stackrel{\mu}{\Longrightarrow} P'$.

Another up-to technique, first proposed in [MPW92], is called *bisimulation up-to restriction.* The idea is to achieve the closure of $\mathcal{R}$ modulo possible cancellation of the outermost restrictions. To define <>-bisimulation up-to restriction, the modifications of definition $(*)$ involves only the clause for silent actions, which now becomes:

- whenever $P \xrightarrow{\tau} P'$, then Q' exists s.t. $Q \xrightarrow{\tau} Q'$ and for some $P'', Q'', \tilde{x}$, it holds that $P' = \boldsymbol{\nu}\, \tilde{x}\, P''$, $Q' = \boldsymbol{\nu}\, \tilde{x}\, Q''$ and $P'' \mathcal{R} Q''$.

For bisimulation up-to restriction the extension to the weak case is smooth: the replacement of the "strong" arrow $Q \xrightarrow{\mu'} Q'$ with the "weak" arrow $Q \stackrel{\widehat{\mu'}}{\Longrightarrow} Q'$ is enough.

The two up-to techniques considered above can also be mixed together, yielding a *bisimulation up-to $\bowtie$ and up-to restriction.* In this technique, the clauses for visible actions are the same as for bisimulations up-to $\bowtie$; the clause for τ-actions is the same as for bisimulation up-to restriction, but with $P'' \bowtie \mathcal{R} \bowtie Q''$ (possibly only $P'' \mathcal{R} \bowtie Q''$ in the weak case) instead of $P'' \mathcal{R} Q''$.

2.6 Bisimulation and congruence for π-calculus

We conclude the chapter with the definition of bisimulation and congruence for π-calculus. Previous work on π-calculus has individuated two different ways of presenting bisimulation, distinguished as *late* and *early*. Intuitively, the discrepancy between them arises in the instantiation of the formal parameter of an input: In the early case the instantiation occurs at the moment of inferring the input action, whereas in the late case at the moment of inferring a communication. In other words, in the early case the receipt of an object on a port is viewed as one atomic event, whereas in the late case it is viewed as the composition of of two more atomic events, namely first the commitment to a port and then the acquirement of the object. The separation of these two events yields a stronger equivalence than the corresponding late one [MPW92,MPW91].

To see the difference between the two with an example, consider the processes (from [MPW92])

$$P \stackrel{def}{=} a(x).R + a(x).\mathbf{0} \qquad Q \stackrel{def}{=} P + a(x).[x = b]R$$

where R is any non-null process. These processes are early bisimilar, since depending on whether the value y received on a is equal to b or not, Q's last summand is equal to P's first summand or to P's second summand. But P and Q are distinguished in the late bisimulation, where the choice of the value y with which to instantiate the bound name x is done "later" than the choice of the input prefix to consume. Therefore, Q's commitment to $a(x).[x = b]R$ cannot be matched by P, since different instantiations of x distinguish $[x = b]R$ from R and $\mathbf{0}$.

In the thesis, we shall stick to early bisimulation. There are various reasons for this. Firstly, exploiting the early transition system presented in Section 2.2.2, early bisimulation can be given a simpler definition, in which the clause of bisimilarity is unique for all actions. By contrast, late bisimulation (even using a late transition system) has to distinguish the clause for input actions from the clause for output actions. Secondly, early bisimulation is the one which is recovered with barbed

bisimulation, as shown in the next section. Thirdly, we think that the intuition behind early bisimulation — the atomicity requirement on input actions — is more convincingly.

Definition 2.6.1 (early bisimulation) *A relation $\mathcal{R}$ on π-calculus processes is an* early simulation *if whenever $P\ \mathcal{R}\ Q$ and $P \xrightarrow{\mu} P'$ with $bn(\mu) \cap fn(P,Q) = \emptyset$, then Q' exists s.t. $Q \xrightarrow{\mu} Q'$ and $P'\ \mathcal{R}\ Q'$; $\mathcal{R}$ is an* early bisimulation, *briefly $\sim_e$-bisimulation, if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are early simulations. Two process P and Q are* early bisimilar, *briefly $P \sim_e Q$, if $P\ \mathcal{R}\ Q$ for some early bisimulation $\mathcal{R}$.* □

Early bisimulation is a congruence over all operators but input prefixing (the same is true for late bisimulation).

Example 2.6.2 *Let $x : y$ and $P \stackrel{def}{=} x|\overline{y}$, $Q \stackrel{def}{=} x.\overline{y} + \overline{y}.x$. Then $P \sim_e Q$. However, it holds that $a(x).P \not\sim_e a(x).Q$, since $P \xrightarrow{a\langle y\rangle} \xrightarrow{\tau} \mathbf{0}$, which $a(x).Q$ cannot match.* □

The full congruence is called *early congruence* and denoted by $\sim_e^c$. It is defined in the standard way, as described in Section 2.5.2; that is, $P \sim_e^c Q$ if for all first-order substitutions σ, it holds that $P\sigma \sim_e Q\sigma$. The corresponding of $\sim_e$ and $\sim_e^c$ in the weak case are called *weak early bisimulation* and *weak early congruence*, and denoted by $\approx_e$ and $\approx_e^c$, respectively.

Chapter 3

Barbed Bisimulation

3.1 Motivations

We have discussed in Section 1.3 the problems for the definition of a natural bisimulation in a higher-order process calculus and the importance of finding a notion of bisimulation which can be used uniformly in different calculi.

The idea is to try to achieve this by equipping a global observer with a *minimal* ability to observe actions and/or process states. We then obtain an equivalence, namely indistinguishability under global observations. This in turn induces a congruence over agents, namely equivalence in all contexts. The question is: what minimal power of observation is needed so that the congruences induced in this way coincide with the familiar bisimulation congruences?

It is reasonable to examine first the case in which there are *no* observables. Unfortunately the *reduction congruence* which is so obtained is in general not discriminating enough. Therefore it is necessary to add the capability of observing some properties of the states. It seems natural in concurrency that the extra power provided is in terms of action observability; our choice has been to give the external observer visibility of the channel at which an action occurs. We call the resulting bisimulation relation *barbed bisimulation*, because somehow it adds "barbs" to the reduction congruence (as such resembles what A. Pnueli does in [Pnu85] w.r.t.

trace semantics). The purpose of this chapter is to show that barbed bisimulation plus parametrisation over contexts is enough to give the desired discriminating power.

An important feature of barbed bisimulation is that it can be successfully employed in the setting of reduction semantics. It allows us to recover from such a formulation the well-known bisimulation-based equivalences which are defined the labelled transition system, an important requirement for the adoption of reduction semantics.

Viewing it differently, the "labeled" equivalences, i.e. the ordinary equivalences of the labelled transition systems, since they do not involve quantification over contexts, can be seen as direct characterisations of the "barbed" equivalences. Indeed, the barbed equivalences may act as support for the labeled equivalences (similarly as reduction semantics does for labelled transition semantics). For instance, a barbed equivalence may guide us to the definition of the labeled equivalence and ensure us of its sensefulness. These benefits may be important when tackling the analysis of a new calculus, for which barbed bisimulation immediately suggests a natural congruence. Furthermore, it becomes an excellent test for the calculus and its operators to see whether they can express a direct characterisation of this congruence. All this is exemplified in the study of HOπ carried out in Chapter 4.

In the π-calculus the use of barbed bisimulation has other interesting outcomes. Firstly: We have seen in Section 2.6 there are two major philosophies for defining bisimulations in the π-calculus, referred as *early* and *late* and the studies on π-calculus so far appeared in the literature have not clarified yet which one should be preferred. Then the question is: Which of the two – if any – is it captured using the barbed bisimulation machinery? We shall see that the answer is the early version — a good point, we think, in its favour.

Secondly: It is generally recognised that a natural equivalence should be *observational*. That is, two systems are equivalent whenever by interacting with them from the outside world, no difference can be observed. But then one might

argue that the ordinary bisimulation for the π-calculus — in the late or early version — is not quite observational: One is supposed to be able to distinguish whether a name received in a communication is bound by a restriction or not, and properly, this is something which cannot be considered as *visible*. These doubts are cleansed by appealing to the characterisation in terms of barbed bisimulation, since the predicates employed in the definition of the latter have an irrefutable "truly observational" mark.

As an aside let us just point out that the possibility of parametrising barbed bisimulation over a particular class of contexts seems to have other interesting applications. In Chapter 6 it will be used to obtain a fully abstract model for lazy λ-calculus from its encoding into π-calculus and HOπ. We would also like to apply this parametrisation in the framework of action refinement. Intuitively, since the refinement of an action modifies the communication protocol of a process, only contexts which "respect" such a protocol should be considered when verifying the equivalence between two refined processes.

Although barbed bisimulation will be examined here in the setting of process algebra, the idea is more general and, in fact, can be used in any term rewriting structure $\{Pr, \longrightarrow, \{\downarrow_a\}_a\}$, where Pr is the set of terms, $\longrightarrow$ is the reduction relation and $\{\downarrow_a\}_a$ is a certain set of observation predicates (those producing the barbs). Thus, reduction congruence would correspond to the case in which this set of predicates is empty. Moreover, in λ-calculus, using the observation predicates to detect the possibility of convergence of a term, barbed bisimulation induces Abramsky's applicative bisimulation [Abr89] (see Section 6.3.1).

Structure of the chapter: We start Section 3.2 by introducing reduction bisimulation and congruence; we show that in general these are over-generous; to remedy, we move to barbed bisimulation. In Section 3.3 we prove that in CCS and π-calculus, the congruences induced by barbed bisimulation coincides with the ordinary bisimulation-based equivalences.

3.2 From reduction bisimulation to barbed bisimulation and congruence

In this section, although the examples are in CCS or π-calculus, in the definitions we are not restricted to any specific process calculus. We only require that

- the calculus possesses a reduction relation (which is written as $P \longrightarrow P'$);
- processes use channels for communications, and for each channel a there is an *observation predicate* $\downarrow_a$ detecting the possibility of performing an action along a.

We use Pr to denote the class of processes of the calculus and P, Q to range over Pr. As usual, $\Longrightarrow$ is the reflexive and transitive closure of $\longrightarrow$; moreover we use $P \Downarrow_a$ to mean that $P \Longrightarrow P' \downarrow_a$, for some P'.

Example 3.2.1 Let P be the π-calculus process $\overline{a}\langle y\rangle + b(x) + \tau.c$. We have $\{z : P \downarrow_z\} = \{a, b\}$. Notice that $P \not\downarrow_c$, since P cannot perform immediately an action at c. Moreover we have $\{z : P \Downarrow_z\} = \{a, b, c\}$. □

Reduction bisimulation

From a process calculus point of view, reduction bisimulation represents an attempt to recover familiar bisimulation-based equivalences — at least in the strong case — by focusing only on reduction, the simplest form of action.

Definition 3.2.2 Reduction bisimulation *is the largest symmetric relation* $\dot{\sim}_{\text{red}} \subseteq Pr \times Pr$ *s.t.* $P \dot{\sim}_{\text{red}} Q$ *and* $P \longrightarrow P'$ *imply that some* Q' *exists with* $Q \longrightarrow Q'$ *and* $P' \dot{\sim}_{\text{red}} Q'$. □

By itself, $\dot{\sim}_{\text{red}}$ is not very interesting. It is rather weak; in general it is even not preserved by parallel composition, as the following example shows:

Example 3.2.3 Let $P \stackrel{def}{=} a.\mathbf{0}$ and $Q \stackrel{def}{=} \mathbf{0}$. Then it holds that $P \dot{\sim}_{\text{red}} Q$, but $P \mid \overline{a} \not\dot{\sim}_{\text{red}} Q \mid \overline{a}$.

It is natural then to consider the congruence induced by $\dot{\sim}_{\text{red}}$.

Definition 3.2.4 *Two processes P and Q are* reduction congruent, *briefly $P \sim^{c}_{\text{red}} Q$, if for each context $C[\cdot]$, it holds that $C[P] \dot{\sim}_{\text{red}} C[Q]$.* □

In [MS92], $\sim^{c}_{\text{red}}$ is compared with CCS's strong bisimulation. It is shown that the latter implies the former but that in general the converse fails. To see this, let $P \stackrel{def}{=} \tau.P$ and $Q \stackrel{def}{=} \tau.Q + a.P$. They are not strongly bisimilar. However they are reduction congruent (this example is due to Gerard Boudol). Intuitively, since any state that P and Q can reach is *divergent*, i.e. can evolve through an unbounded number of τ-actions, no CCS context can make the distinction between these processes as long as only τ-actions are taken into account. Indeed, in CCS divergency is exactly what makes the two relations different. They coincide on the class of divergence-free processes [MS92].

To conclude, since reduction congruence is not discriminating enough (furthermore in the weak case it corresponds to the universal relation), the power of the observer has to be increased. We do this by adding "barbs" to reduction bisimulation.

Barbed bisimulation

Definition 3.2.5 *A relation $\mathcal{R} \subseteq Pr \times Pr$ is a* barbed simulation *if $(P,Q) \in \mathcal{R}$ implies:*

1. *whenever $P \longrightarrow P'$ then $Q \longrightarrow Q'$ and $(P',Q') \in \mathcal{R}$;*
2. *for each channel a, if $P \downarrow_a$ then also $Q \downarrow_a$.*

A relation $\mathcal{R}$ is a barbed bisimulation *if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are barbed simulations. Two processes P and Q are* barbed-bisimilar, *briefly $P \dot{\sim} Q$, if $(P,Q) \in \mathcal{R}$, for some barbed bisimulation $\mathcal{R}$*

Barbed bisimulation describes a game between two machines which can challenge one another on the reductions they can perform, under the condition that the states reached must have the same observation sets. To obtain *weak barbed bisimulation*, briefly $\dot{\approx}$, just replace in the above definition the transition $Q \longrightarrow Q'$ with $Q \Longrightarrow Q'$ and the convergence predicate $Q \downarrow_a$ with $Q \Downarrow_a$.

As for reduction bisimulation, by itself barbed bisimulation is too weak; we use contexts to get a finer relation. If we use the class of all contexts, we obtain the relation called *barbed congruence*.

Definition 3.2.6 *Two processes P and Q are* barbed-congruent, *written $P \sim^c Q$, if for each context $C[\cdot]$, it holds that $C[P] \dot{\sim} C[Q]$.* □

The most studied bisimulations in the weak case usually are not preserved by *dynamic* operators, i.e. operators like prefixing or sum which can be discharged when an action is produced. To recover these equivalences, we have to parametrise $\dot{\approx}$ over a *subclass* of all possible contexts. The obvious representative is the subclass $\mathcal{SC}$ of *static contexts*, that is contexts which are built by composing $[\cdot]$ and processes by means of only static operators. An operator $\otimes$ is called *static* if the rules describing its behaviour have the conclusion of the form $\otimes(P_1 \ldots P_n) \xrightarrow{\alpha} \otimes(P'_1 \ldots P'_n)$. Formally, the class $\mathcal{SC}$ is defined by the grammar:

$$\mathcal{SC} := [\cdot] \mid P \mid \otimes(\mathcal{SC}_1, \ldots, \mathcal{SC}_{r(\otimes)})$$

where P is a process, $\otimes$ is a static operator and $r(\otimes)$ is its arity and with the restriction that the hole $[\cdot]$ occurs at most once in an expression.

Definition 3.2.7 *Two processes P and Q are* barbed equivalent, *written $P \sim Q$ if for each static context $C[\cdot]$, it holds that $C[P] \dot{\sim} C[Q]$.*

We shall denote by $\approx^c$, $\approx$, the corresponding of $\sim^c$ and $\sim$ in the weak case. The extensions of these relations to abstractions and open agents will be considered in Section 4.7.2. We said in Section 2.5.2 that in the thesis we follow the convention

that a symbol indexed by a superscript "c" denotes a relation which is a full congruence; similarly undecorated symbols will be used to denote relations which are congruences over static contexts.

A brief discussion regarding the observation predicates employed in the definition of barbed bisimulation is pertinent, for these are not the only predicates one might think of using. The simplest possible predicate is the one adopted in [MS92], which just detects whether *some* observable action can be performed. It seems that in the strong case this indeed suffices to induce the desired congruences; but the answer is still unknown in the weak case.

Another possibility is to partition observable actions into subsets and then only to allow the external observer to discriminate between actions from different subsets. Notice that the predicates in Definition 3.2.5 represent an instance of this, in which the subset is determined by the subject of the action. Some preliminaries studies that we have conducted in this direction suggests that a partion into two subsets, for instance the set of input and the set of output actions, might be enough to recover the CCS ordinary bisimulations.

We prefer to leave for future work a detailed analysis of what is the best balance between power of observer and ease of manipulation. But at least, the choice that we have made here appears to be a reasonable one, and it works perfectly well in all cases we have considered.

3.3 Discriminating power induced by barbed bisimulation

In this section, as a test for barbed bisimulation, we show that in CCS and in the π-calculus it allows us to recover the familiar bisimulations and their congruences. The schema of the proof in all these results is the same. We only give the details for CCS, since it is a simpler case, whereas we have relegated those for the π-calculus

in Appendix A.

3.3.1 The CCS case

As mentioned in Section 2.1.2, CCS is derived from the polyadic π-calculus by imposing the sorting $\{Name \mapsto ()\}$. Also the ordinary bisimulation of CCS coincide then with π-calculus's early bisimulation (Definition 2.6.1); we recall the former below for commodity.

Definition 3.3.1 (bisimulation for CCS) *A relation $\mathcal{R}$ on CCS processes is a* simulation *if whenever $P\ \mathcal{R}\ Q$ and $P \xrightarrow{\mu} P'$, then Q' exists s.t. $Q \xrightarrow{\mu} Q'$ and $P'\ \mathcal{R}\ Q'$; $\mathcal{R}$ is a* bisimulation *if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations. Two process P and Q are* bisimilar, *written $P \sim_{\text{ccs}} Q$ if $P\ \mathcal{R}\ Q$ for some bisimulation $\mathcal{R}$.* □

The corresponding of $\sim_{\text{ccs}}$ in the weak case, called *weak bisimulation*, is denoted by $\approx_{\text{ccs}}$. Both $\sim_{\text{ccs}}$ and $\approx_{\text{ccs}}$ are also congruences. For the latter, $\approx_{\text{ccs}}$ is preserved by $+$ because we only allow *guarded* sums (however Theorem 3.3.2 remains valid if this restriction on sum is dropped, since the proof only uses static contexts).

We maintain from the π-calculus the hypothesis that each process leaves an infinity of unused names. This will allow us to define, for each given pair of processes P and Q, contexts which have names which do not occur in P or Q. The definition of these contexts will employ a countable infinite number of constants. This prevents us from extending the proof to a language where constants have been replaced by replication, since, as mentioned in Section 2.1.1, replication can only encode a finite number of constants. Moreover, if the number of free names in P or Q is infinite, our contexts will also incorporate infinite sums.

Theorem 3.3.2

1. *$\sim$ and $\sim_{\text{ccs}}$ coincide;*
2. *$\approx$ and $\approx_{\text{ccs}}$ coincide.*

We only show the proof for the weak case because it is the hardest and because in the thesis we shall be working in the weak case. The proof in the strong case is substantially simpler, due to the tighter control we have on the production of silent actions from the processes (see also for this the proof of Theorem 1 in [MS92]).

The inclusion $\approx_{ccs} \subseteq \approx$ is immediate: Clearly $\approx_{ccs} \subseteq \dot{\approx}$ (every weak bisimulation is a weak barbed bisimulation); since $\approx_{ccs}$ is preserved by the static operators, $P \approx_{ccs} Q$ implies $C[P] \approx_{ccs} C[Q]$, which in turn, implies $C[P] \dot{\approx} C[Q]$, for every static context $C[\cdot]$.

The remainder of the section is devoted to the proof of the inclusion in the opposite direction. The basic idea is the same as the related proofs in [MS92]: Build a class of contexts $\mathcal{C}xt$ powerful enough to guarantee that barbed bisimulation on these contexts implies bisimulation, i.e. prove that

$$\mathcal{R} = \{(P,Q) \ : \ \text{for some } C[\cdot] \in \mathcal{C}xt \ \ C[P] \dot{\approx} C[Q]\}$$

is a weak bisimulation. The key property of the contexts in $\mathcal{C}xt$ is the following. Suppose that $P \xrightarrow{\mu} P'$ and that $P \ \mathcal{R} \ Q$ because $C[P] \dot{\approx} C[Q]$. Then we can find a context $C'[\cdot] \in \mathcal{C}xt$ s.t. $C[P] \Longrightarrow C'[P']$. Since $C[P] \dot{\approx} C[Q]$, the process $C[Q]$ must be able to match this move. But the only way it can do so is by employing a μ-derivative Q' of Q (i.e. $Q \stackrel{\mu}{\Longrightarrow} Q'$) and evolve to $C'[Q'] \dot{\approx} C'[P']$. This closes up the bisimulation.

The definition of the contexts in $\mathcal{C}xt$ depends upon the set of free names in the processes P, Q under consideration; more precisely, such contexts must have "more names" than P and Q. Before defining them, let us introduce some notation.

Let H be a set of pairs of names. We denote by H_i the projection of H on the i-th element of the pairs, for $i = 1, 2$. For instance, if $H = \{(a, a'), (b, b')\}$, then $H_1 = \{a, b\}$ and $H_2 = \{a', b'\}$. We require that $H_1 \cap H_2 = \emptyset$ and that H_2 is made of all distinct names (so for instance, in the previous example for H, it must be that $a' \neq b'$). Moreover, we assume that c, in, ou and d_n, for n positive integer, are names which do not appear in H.

We follow our convention on the use of constants of omitting from the parameters of a constant those names which are supposed not to be instantiated. We make an exception for the constant V, which is given H as parameter although H does not change in the recursive calls of V. This is for uniformity with the corresponding proof for π-calculus (Theorem 3.3.3), in which H does change. For notational simplicity, we use H, which is a set of *pairs* of names, as parameter for V, rather than the tuple of names occurring in H, as the syntax of the language would require. The definition of V depends upon the cardinality of H; therefore, it is supposed that there is a different constant for each cardinality (including the case in which the cardinality is ω).

The term $V\langle H\rangle$ is the central process of a context in $\mathcal{C}xt$, which is of the form

$$C_n^H[\cdot] \stackrel{def}{=} [\cdot] \mid V\langle H\rangle \mid Count_n$$

where

$$Count_n \stackrel{def}{=} d_n + c.Count_{n+1} \qquad n > 1$$

$$\begin{aligned} V \stackrel{def}{=} (H)\Big(& \quad \textstyle\sum_{(a,a')\in H} \overline{a}.\overline{c}.\overline{c}.\left(\tau.(a' + in) + \tau.V\langle H\rangle\right) && (a1)\\ & + \textstyle\sum_{(a,a')\in H} a.\overline{c}.\overline{c}.\left(\tau.(a' + ou) + \tau.V\langle H\rangle\right) && (a2)\\ & + \tau.d_0 + \tau.d_1 \quad \Big) && (a3) \end{aligned}$$

The formal definition of $\mathcal{R}$ is

$$\mathcal{R} = \{(P,Q) \;:\; n, H \text{ exist s.t. } \; fn(P,Q) \subseteq H_1 \text{ and } \; C_n^H[P] \dot{\approx} C_n^H[Q]\}$$

For simplicity, in the following the parameter H will be omitted, and we simply write V in place of $V\langle H\rangle$. Let us now describe the rôles played by $Count_n$ and V to guarantee that $\mathcal{R}$ is weak bisimulation. The task of $Count_n$ is to control and discipline the production of visible actions by P and Q. The process $Count_n$ acts as a counter. In $C_n[P]$ the name d_n is observable, but no name $d_{n'}$ with $n' < n$ is. Now, if P can perform a communication, there is a reduction $C_n[P] \Longrightarrow R$ in which the counter $Count_n$ has evolved to $Count_{n+2}$, and hence the observability

threshold for the d_i's increases to d_{n+2}. In consequence, a reduction $C_n[Q] \Longrightarrow T$ can match the move of $C_n[P]$ only if in so doing, Q has effected exactly one communication — otherwise, different sets of names d_i would be observable in R and T.

Now V. Summand $(a1)$ in V is used to verify the bisimilarity of P, Q on inputs, whereas $(a2)$ is used for outputs (the two summands have the same structure). Names in and ou are used to signal whether the summand $(a1)$ or $(a2)$ has been selected and therefore to know the kind of action — input or output — which P and Q have produced. Further, the label of P and Q's actions can be deduced from the name a'. In the definition of V, the presence of the two consecutive outputs $\overline{c}.\overline{c}$ is not strictly necessary, for one only output would have been enough; having two of them makes the proofs below clearer.

The recursive definition of V (and $Count_n$) allows us to iterate the above analysis on the derivatives of P and Q. Finally, the summand $(a3)$ of V is needed for technical reasons which will become clear later.

As it emerges from the explanation above, the proof that $\mathcal{R}$ is a weak bisimulation is based on the check of observability of the names in $K = \{in, ou\} \cup H_2 \cup \{d_n\}_n$. Let us write then $R \Downarrow_{K'}$, for $K' \subseteq K$, to mean that K' is the maximum subset of names from K which are observable in R. We shall abbreviate $R \Longrightarrow R_1 \Downarrow_{K'}$, for some R_1, as $R \Longrightarrow\Downarrow_{K'}$. Note the difference between this notation and the notation $R \Downarrow_{d_n}$, in which the subscript is a name rather than a set: The latter means that d_n can be observed in R, but without excluding other names from K from being observable too.

We can now embark in the proof that $\mathcal{R}$ is a weak bisimulation. We have to show that whenever $P \xrightarrow{\mu} P'$, some Q' exists s.t.

$$Q \stackrel{\mu}{\Longrightarrow} Q' \quad \text{and} \quad P' \,\mathcal{R}\, Q'. \tag{3.1}$$

We consider only the case when μ is an input, say $\mu = a$. The cases when μ is an output can be worked out in a quite similar way; and the case when μ is

a silent action is simpler. From the summand $(a1)$ of V, if V_1 is its derivative $\tau.(a' + in) + \tau.V$, we get

$$\begin{aligned} C_n[P] &\longrightarrow\!\!\!\!\longrightarrow P' \mid \overline{c}.V_1 \mid Count_{n+1} \stackrel{def}{=} R_1 \\ &\longrightarrow P' \mid V_1 \mid Count_{n+2} \stackrel{def}{=} R_2 \\ &\longrightarrow P' \mid V \mid Count_{n+2} \stackrel{def}{=} R_3 \end{aligned}$$

Let us analyse how $C_n[Q]$ can match each of these steps. We need processes T_1, T_2, T_3, s.t.

$$\begin{aligned} C_n[Q] &\Longrightarrow T_1 \dot{\approx} R_1 \\ T_1 &\Longrightarrow T_2 \dot{\approx} R_2 \\ T_2 &\Longrightarrow T_3 \dot{\approx} R_3 \end{aligned}$$

We shall show that T_i's structure, $i = 1, 2, 3$, has strictly to mirror R_i's.

First step. $(C_n[P] \longrightarrow\!\!\!\!\longrightarrow R_1.)$

Process $C_n[Q]$ has to perform some move, because $C_n[P] \Downarrow_{d_n}$, but $R_1 \not\Downarrow_{d_n}$. However, since $R_1 \Downarrow_{d_{n+1}}$, process $Count_n$ cannot participate in more than one interaction. Therefore T_1 must be of the form $Q_1 \mid \overline{c}.V_1' \mid Count_{n+1}$ for some Q_1 and V_1' s.t. $Q \stackrel{\mu'}{\Longrightarrow} Q_1$ and $V \stackrel{\overline{\mu'}}{\longrightarrow}\!\!\!\!\longrightarrow \overline{c}.V_1'$. But $R_1 \Longrightarrow\!\!\Downarrow_{\{a', in, d_{n+2}\}}$ and T_1 must be able to do the same. We deduce that $\mu' = a$ and hence also $V_1' = V_1$. The conclusion is that $T_1 = Q_1 \mid \overline{c}.V_1 \mid Count_{n+1}$, for some Q_1 s.t. $Q \stackrel{a}{\Longrightarrow} Q_1$.

Second step $(R_1 \longrightarrow R_2.)$

We reason similarly as in the first step. Since $R_2 \not\Downarrow_{d_{n+1}}$ but $R_2 \Downarrow_{d_{n+2}}$, in $T_1 \Longrightarrow T_2$ there must be exactly one interaction between $c.V_1$ and $Count_{n+1}$. Moreover, because of the actions $V_1 \longrightarrow a' + in$, and $V_1 \longrightarrow\!\!\!\!\longrightarrow d_0$, we have $R_2 \Longrightarrow\!\!\Downarrow_{\{a', in, d_{n+2}\}}$ and $R_2 \Longrightarrow\!\!\Downarrow_{\{d_0, d_{n+2}\}}$. One can easily verify that in $T_1 \Longrightarrow T_2$, this precludes $\overline{c}.V_1$, once evolved to V_1, from performing other transitions. Thus $T_2 = Q_2 | V_1 | Count_{n+2}$, for some Q_2 s.t. $Q_2 \Longrightarrow Q_1$.

Third step $(R_2 \longrightarrow R_3.)$

First we make some consideration on R_2 and R_3. The process $\tau.V$ is a summand of V_1 and the transition $R_2 \longrightarrow R_3$ must have been caused by $V_1 \longrightarrow V$. Therefore:

1. Because of the transition $V_1 \longrightarrow a' + in$, we have $R_2 \Longrightarrow\Downarrow_{\{a', in, d_{n+2}\}}$, whereas R_3 cannot.

2. Because of the transitions $V \longrightarrow d_0$ and $V \longrightarrow d_1$, we have $R_3 \Longrightarrow\Downarrow_{\{d_0, d_{n+2}\}}$ and $R_3 \Longrightarrow\Downarrow_{\{d_1, d_{n+2}\}}$.

Now, let us turn to $T_2 \Longrightarrow T_3$. In this reduction:

- To match (1), process V_1 has to perform *some* action.
- To match (2), process V_1 cannot interact with Q_2 or $Count_{n+2}$, nor can it evolve to some derivative of $\tau.(a' + in)$ or $\tau.d_0$ or $\tau.d_1$. (The use of $\tau.d_1$, in addition to $\tau.d_0$, prevents V_1 from evolving to the derivative d_0 of $\tau.d_0$, in the case in which the subcomponent P' of R_3 does not have any observable behaviour).

Hence V_1 can only perform $V_1 \longrightarrow V$. We conclude that $T_3 = Q_3 \mid V \mid Count_{n+2}$, for some Q_3 s.t. $Q_2 \Longrightarrow Q_3$.

Combining what we have obtained in the three steps above, we have $Q \stackrel{a}{\Longrightarrow} Q_3$. Moreover, from $C_{n+2}[P'] = R_3 \dot{\approx} T_3 = C_{n+2}[Q_3]$, we get $P' \mathcal{R} Q_3$. Hence, for $Q' \stackrel{def}{=} Q_3$ in (3.1), this concludes the analysis.

3.3.2 The π-calculus case

The main theorem to prove is

Theorem 3.3.3

1. $\sim$ *and* $\sim_e$ *coincide;*

2. $\approx$ *and* $\approx_e$ *coincide.*

PROOF: Again, we only look at the weak case because it is much harder. The proof is quite similar to that of Theorem 3.3.2. The main difference is in the

definition of the process V, whose summands have to be made more sophisticated since the π-calculus actions are more complex than the CCS actions. The details can be found in Appendix A. □

From Theorem 3.3.3 we can infer similar equalities for the corresponding congruences:

Corollary 3.3.4

1. $\sim^c$ *and* $\sim_e^c$ *coincide;*

2. $\approx^c$ *and* $\approx_e^c$ *coincide.*

Chapter 4

Bisimulation in HOπ

In Section 1.3 we pointed out some counterintuitive equalities on higher-order processes determined by higher-order bisimulation. The adoption of barbed bisimulation allows us to overcome these problems. In this chapter we deepen the analysis of higher-order bisimulation in HOπ. As for π-calculus, we focus on the congruence induced by (weak) barbed bisimulation on static contexts and on all contexts, respectively called barbed equivalence ($\approx$) and barbed congruence ($\approx^c$). The target is to derive direct characterisations for $\approx$ and $\approx^c$, as simple as possible, somehow repeating what we did in the previous chapter for CCS and π-calculus.

We had to face some serious obstacle. In a higher-order process calculus, the possibility of transmitting an agent not only makes it difficult to understand which conditions a bisimulation should require on higher-order actions. Even more, it makes potentially very burdensome the verification of the congruence properties of the resulting relation (above all, the congruence over parallel composition) and the construction and proof of bisimulations. However, we think that our study does show that a simple treatment of the higher-order constructs is feasible. The mathematical tools we provide (factorisation theorem, triggered and normal bisimulation) are derived through non-trivial proofs, but once we have them, the reasoning on the calculus becomes manageable.

We start from *context bisimulation*, probably the most intuitive adjustment of

higher-order bisimulation for eliminating its drawbacks discussed in Section 1.3. Context bisimulation is however heavy to handle, due to the appearance of universal quantifications in its definition. But we need it as a supporting relation to derive simpler characterisations. To this end, a central rôle is played by a special kind of agents called *triggers*. They are elementary agents whose only functionality is to activate a copy of another agent, providing it with the possible arguments. Using triggers, any subagent of a given agent can be factorised out: This is the content of the *factorisation theorem*. Building on this, we introduce the subclass of *triggered agents*, in which every agent emitted in an output or "expected" in an input is a trigger. Thus higher-order communications have become homogeneous and have lost all their potential richness and variety. This greatly simplifies the reasoning over agents. In particular, on triggered agents context bisimulation coincides with *triggered bisimulation*. The advantage of the latter is that the clause for higher-order outputs just requires *identity* (modulo α-conversion) on the labels of two matching actions and the clause for higher-order inputs only contemplates the input of a fresh trigger. We then define a mapping $\mathcal{T}$ which transforms every agent into a triggered agent. By exploiting $\mathcal{T}$, first we are able to prove a simple characterisation of context bisimulation on the whole class of HOπ agents (not necessarily in triggered form), called *normal bisimulation*. This is not as simple as triggered bisimulation, but at least all universal quantifications in the definition of context bisimulation have disappeared. Secondly, we show (for the weak case) that all these bisimulations and their congruences coincide with barbed equivalence and congruence. We shall use $\mathcal{T}$ also in Chapter 5, to define the encoding from HOπ to π-calculus, and in Chapter 7, to study the effect of modifications of the sorting on the equivalence of processes.

To sum up, the equivalences we shall consider on *HOπ* agents and which we shall prove to represent the same relation, are barbed equivalence, context bisimulation, and normal bisimulation (and for triggered agents, also triggered bisimulation). The same relationship holds on the corresponding congruences.

Some of the proofs we shall deal with are rather involved. In a few cases, in order to make our ideas more intelligible, we have chosen to present the proof of a simplified version of the result, under the condition that the extension to the full result is entirely smooth. The most important example is the limitation to unary arities for the sorts of the HOπ agents. The only exception is the finiteness requirement on sums and on the number of constant with which an agent is defined: These simplifications, discussed in the coming section, are meaningful because they are used in a non-trivial way in some proofs.

4.1 The reduced HOπ

We present our result on the subclass of the HOπ agents which satisfy the following two restrictions:

- All arities are unary, that is:
  - Only *unary* abstractions are permitted;
  - *One* only value can be exchanged in a communication. The value can be a name or a (unary) abstraction.
- Each agent A is *finitely* describable. That is, each sum in A is finite and the set $\mathcal{D}_A$ of constants from which the definition of A depends, is finite. This in turns implies finiteness on the number of free names of A and on the number of parameters of the constants in $\mathcal{D}_A$. These assumptions allow us to have replication in place of constants since, as mentioned in Section 2.1.1, a finite number of constants which have finite parameters can be coded up with replication — and vice versa.

It is important to stress the different significance of the two simplifications: The former is purely to make our results more readable — the generalisation to the calculus with arbitrary arities does not introduce semantic complications (see also

Section 4.7.3). By contrast, the latter does intend to exclude some agents, namely those defined with an infinite number of constants or with infinite summation. In particular, the possibility of adopting replication is useful because it is precisely the recursive operator which is needed in the definition of our encodings and because it facilitates the reasoning by structural induction in the proofs. We think that our results do generalise to the case of infinite constants and sums; but we shall leave the issue for future investigations (we refer to Section 2.1.1 for more discussion on the choice between constants and replication).

Thus, the subclass of agents we shall be working with contains the well-sorted agents which are described by the following grammar:

$$\begin{array}{rcl} P & :: & \sum_{i\in I}\alpha_i.P_i \;\mid\; P_1 \mid P_2 \;\mid\; \boldsymbol{\nu}\, x\, P \;\mid\; [x=y]P \;\mid\; !\,P \;\mid \\ & & X\langle F\rangle \;\mid\; X\langle x\rangle \end{array}$$

where I is a finite indexing set and prefixes and abstractions are of the form

$$\begin{array}{rcl} \alpha & :: & \overline{x}\langle F\rangle \;\mid\; \overline{x}\langle y\rangle \;\mid\; x(X) \;\mid\; x(y) \\ F & :: & (X)P \;\mid\; (x)P \;\mid\; X \end{array}$$

In the remainder of the chapter, anything which has to do with HOπ should be referred to this subclass of agents. We keep however '$\stackrel{def}{=}$' as abbreviation mechanism, to assign a name to expressions to which we want to refer later. We remind the reader that given an abstraction $G = (X)P$, we use $G\langle F\rangle$ as an abbreviation for $P\{F/X\}$.

Remark 4.1.1 In this reduced HOπ perhaps replication could have been avoided. Thomsen [Tho90] has shown that in a higher-order process calculus, the effect of recursion and replication in some cases can be simulated using restriction and parallel composition (similarly to the way in which one defines the fixpoint operator Y in the λ-calculus). It remains to see whether a general result of this form is possible. Anyway, it is convenient for us to have replication explicitly; moreover its presence does not complicate our theory. □

We call $(x)P$ a *first-order abstraction*, and $(X)P$ a *higher-order abstraction*. Similarly, we distinguish *first-order applications* $X\langle x\rangle$, from *higher-order applica-*

tions $X\langle F\rangle$; and *first-order prefixings/actions* in which the object part is a name, like in $\overline{a}\langle b\rangle.P$ and $Q \xrightarrow{\overline{a}\langle b\rangle} P$, from *higher-order prefixings/actions* where the object part represents an agent, like in $\overline{a}\langle F\rangle.P$ and $Q \xrightarrow{\overline{a}\langle F\rangle} P$.

The simplifications in the syntax of HOπ also induce simplifications in the language of sorts. As object sort, all we need is the sort () for processes and the unary object sorts El_u, which are described by the grammar

$$El_u :: s \mid (El_u) \quad \text{for } s \in \text{subject sorts}$$

We shall omit the sorting information unless necessary or important. For instance, when writing $X\langle F\rangle$, it is intended that for some S, it holds that $X : (S)$ and $F : S$, and similarly in

"for all F, ... $G\{F/X\}$..."

it is intended that

"for every F of the same sort as X, ... $G\{F/X\}$..."

Because of the clear separation between first-order and higher-order actions that we have in this reduced HOπ, when defining bisimulations both of them must be considered. Since the clauses for first-order actions will be always the same, it is convenient to factorise them out. They are the ordinary clauses of early bisimulation for π-calculus:

Definition 4.1.2 (first-order simulation) *A relation $\mathcal{R}$ on HOπ processes is a* strong first-order simulation *if $P\ \mathcal{R}\ Q$ implies that whenever $P \xrightarrow{\mu} P'$ and μ is a first-order action with $bn(\mu) \cap fn(P,Q) = \emptyset$, then $Q \xrightarrow{\mu} Q'$ and $P'\ \mathcal{R}\ Q'$.* □

Its corresponding in the weak case is called *weak first-order simulation.* The following is a useful lemma to reason on the input actions of HOπ processes. We state it only for higher-order actions because this is all we shall need but in fact the result holds for first-order actions too.

Lemma 4.1.3 *Suppose* $P \xrightarrow{a\langle F\rangle} P'$*; then there exists* G *s.t.* $P' = G\langle F\rangle$ *and for every* E*, it holds that* $P \xrightarrow{a\langle E\rangle} G\langle E\rangle$.

PROOF: Induction on the length of the inference of $P \xrightarrow{a\langle F\rangle} P'$.

For instance, if $P = a(X).(\overline{b} \mid X\langle c\rangle)$, we have $P \xrightarrow{a\langle F\rangle} \overline{b} \mid F\langle c\rangle$ and then the G of the lemma is $(X)\big(\overline{b} \mid X\langle c\rangle\big)$.

4.2 Strong context bisimulation

The non-naturalness problems of higher-order bisimulation, examined in Section 1.3, were due to the fact that in a higher-order output the object part and the continuation are examined separately, thus preventing a satisfactory treatment of the channels private to the two. This resembles the problems of *distributed bisimulation* [CH89] in presence of restriction. We might then try to follow the solution for the latter proposed by Boudol et al. in [BCHK91] and based on the introduction of *locations*. The idea is to keep the two components to analyse together but assign them different locations, which can be detected when an observable action is produced. Thus the observable actions of the two components can be distinguished and yet, there can still be private names and communications between them. The inconvenient of this approach is that it forces an extension of the syntax of the language. Instead, we have chosen to avoid the separation between object part and continuation by explicitly taking into account the *context* in which the emitted agent is supposed to be used. We have called the resulting bisimulation *context bisimulation*. We shall see later that its weak version and the corresponding congruence coincide with barbed equivalence and congruence.

4.2.1 Definition and preliminaries

Definition 4.2.1 (context bisimulation) *A relation* $\mathcal{R}$ *over HOπ processes is a* strong context simulation *if* $\mathcal{R}$ *is a strong first-order simulation and* $P\ \mathcal{R}\ Q$

implies, for every F,

1. *whenever* $P \xrightarrow{a\langle F\rangle} P'$, *there exists* Q' *s.t.* $Q \xrightarrow{a\langle F\rangle} Q'$ *and* $P' \,\mathcal{R}\, Q'$,
2. *whenever* $P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle F\rangle} P'$ *there exist* $Q', E, \widetilde{c}$, *s.t.* $Q \xrightarrow{(\nu\,\widetilde{c})\overline{a}\langle E\rangle} Q'$
   and for all G *with* $fn(G) \cap (\widetilde{b} \cup \widetilde{c}) = \emptyset$,

$$\boldsymbol{\nu}\,\widetilde{b}(G\langle F\rangle \mid P') \;\; \mathcal{R} \;\; \boldsymbol{\nu}\,\widetilde{c}(G\langle E\rangle \mid Q').$$

A relation $\mathcal{R}$ *is a* strong context bisimulation, *briefly* $\simeq_{\mathrm{Ct}}$*-bisimulation, if* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are strong context simulations. We say that* P, Q *are* strong context bisimilar, *briefly* $P \simeq_{\mathrm{Ct}} Q$, *if* $P \,\mathcal{R}\, Q$, *for some* $\simeq_{\mathrm{Ct}}$*-bisimulation* $\mathcal{R}$. □

Remark 4.2.2 There is an appealing formulation for context bisimulation following Milner's idea of *concretion* in [Mil91]. A higher-order concretion would be an expression of the form $\boldsymbol{\nu}\,\widetilde{b}\,\langle F, Q\rangle$, using which, a transition $P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle F\rangle}$ can be rewritten as $P \xrightarrow{\overline{a}} \boldsymbol{\nu}\,\widetilde{b}\,\langle F, Q\rangle$. Also Milner's definition of pseudoapplication between concretion and abstraction (of the same arity) can be extended by setting

$$G \bullet \boldsymbol{\nu}\,\widetilde{b}\,\langle F, Q\rangle \stackrel{def}{=} \boldsymbol{\nu}\,\widetilde{b}\big(G\langle F\rangle \mid Q\big)$$

(with possible renaming of names in $\boldsymbol{\nu}\,\widetilde{b}$ to avoid the capture of names in G). Now, using C, D to denote concretions, clause (2) in the definition of context bisimulation can be rephrased as follows:

> whenever $P \xrightarrow{\overline{a}} C$ there exist D, s.t. $Q \xrightarrow{\overline{a}} D$ and for all G, it holds that $G \bullet C \,\mathcal{R}\, G \bullet D$.

As mentioned in Section 2.1.1, we chose to avoid concretions in order not to burden the proofs in the calculus. □

It is easy to convince ourselves that $\simeq_{\mathrm{Ct}}$ is an equivalence relation and that it includes the structural congruence relation; that is

Proposition 4.2.3 *It holds that:*

1. $\simeq_{\mathrm{Ct}}$ *is an equivalence relation;*
2. $P \equiv Q$ *implies* $P \simeq_{\mathrm{Ct}} Q$.

The following proposition shows the soundness of some bisimulation up-to techniques discussed in Section 2.5.3, which will be useful when reasoning with context bisimulation.

Proposition 4.2.4 *Suppose* $\mathcal{R}$ *is one of the following:*

- $a \simeq_{\mathrm{Ct}}$*-bisimulation up-to restriction,*
- $a \simeq_{\mathrm{Ct}}$*-bisimulation up-to* $\simeq_{\mathrm{Ct}}$*,*
- $a \simeq_{\mathrm{Ct}}$*-bisimulation up-to* $\simeq_{\mathrm{Ct}}$ *and restriction.*

Then $\mathcal{R} \subseteq \simeq_{\mathrm{Ct}}$.

PROOF: Standard proof-techniques for bisimulations up-to ([Mil89,MPW92]). The higher-order setting in which we are working does not introduce any additional complication. □

The relation $\simeq_{\mathrm{Ct}}$ is generalised to abstractions and open agents in the obvious way. In the next definition, K stands for a name if F_1 and F_2 are first-order abstractions, for an agent otherwise.

Definition 4.2.5

1. ***($\simeq_{\mathrm{Ct}}$ on abstractions)*** *Let* F_1 *and* F_2 *in* $HO\pi$. *We set* $F_1 \simeq_{\mathrm{Ct}} F_2$ *when for all* K *it holds that* $F_1\langle K\rangle \simeq_{\mathrm{Ct}} F_2\langle K\rangle$.
2. ***($\simeq_{\mathrm{Ct}}$ on open agents)*** *Let* A_1 *and* A_2 *in* $HO\pi^o$, *with* $fv(A_1, A_2) \subseteq \{\widetilde{X}\}$; *we set* $A_1 \simeq_{\mathrm{Ct}} A_2$ *if for all* $\widetilde{F} : \widetilde{X}$, *it holds that* $A_1\{\widetilde{F}/\widetilde{X}\} \simeq_{\mathrm{Ct}} A_2\{\widetilde{F}/\widetilde{X}\}$.

4.2.2 Congruence properties of context bisimulation

It is common in calculi which can describe mobile processes that the strong bisimilarity relations are preserved by all operators but first-order input prefixing. The same is true for $\simeq_{\mathrm{Ct}}$ (use the same counterexample 2.6.2 as for π-calculus's early bisimulation).

The proof of congruence over the remaining operators is presented in Theorem 4.2.7. The most difficult results are, by far, congruence for object constructor and for application on the right (Theorem 4.2.7(6-7)). They are exclusive of the higher-order setting to which HOπ belongs. To derive them, we prove congruence for $\simeq_{\mathrm{Ct}}$ over higher-order substitution. This is stated in Proposition 4.2.6; we refer to appendix B for the proof, which is rather delicate.

Proposition 4.2.6 *Let $F_1, F_2, A \in HO\pi^\circ$; then $F_1 \simeq_{\mathrm{Ct}} F_2$ implies $A\{F_1/X\} \simeq_{\mathrm{Ct}} A\{F_2/X\}$.*

PROOF: See Appendix B.

Theorem 4.2.7 (congruence properties of $\simeq_{\mathrm{Ct}}$) *Let $P, Q, F_1, F_2, R \in HO\pi^\circ$; if $P \simeq_{\mathrm{Ct}} Q$ and $F_1 \simeq_{\mathrm{Ct}} F_2$ then*

1. $\boldsymbol{\nu}\, a\, P \simeq_{\mathrm{Ct}} \boldsymbol{\nu}\, a\, Q$;
2. $P \mid R \simeq_{\mathrm{Ct}} Q \mid R$;
3. $!P \simeq_{\mathrm{Ct}} !Q$;
4. $[a = b]P \simeq_{\mathrm{Ct}} [a = b]Q$;
5. $\alpha.P + R \simeq_{\mathrm{Ct}} \alpha.Q + R$, *if α is not a first-order input*;
6. $\overline{a}\langle F_1\rangle.P + R \simeq_{\mathrm{Ct}} \overline{a}\langle F_2\rangle.P + R$;
7. $X\langle F_1\rangle \simeq_{\mathrm{Ct}} X\langle F_2\rangle$;

where, in clauses (5) and (6), the subterm $+R$ can also be missing from the left and right members.

PROOF: Clauses (2), (3), (4), (6), (7) and (5) when α is an output can be easily derived using Proposition 4.2.6. This does not hold for clauses (1), and (5) when α is a higher-order input, since a and α may bind some free occurrences of names or variables in P and Q. For these cases, one has to give the bisimulation explicitly. We omit the details, which are simple. □

From Proposition 4.2.6 we can also immediately infer the congruence of $\simeq_{\mathrm{Ct}}$ w.r.t. the metanotation $G\langle F\rangle$; that is, if $F_1 \simeq_{\mathrm{Ct}} F_2$ then

$$G\langle F_1\rangle \simeq_{\mathrm{Ct}} G\langle F_2\rangle \quad \text{and} \quad F_1\langle E\rangle \simeq_{\mathrm{Ct}} F_2\langle E\rangle$$

4.3 Strong context congruence

Let $\simeq^{\mathrm{c}}_{\mathrm{Ct}}$ be the congruence defined from $\simeq_{\mathrm{Ct}}$ in the standard way by means of first-order substitutions (see Section 2.5.2).

Theorem 4.3.1 *$\simeq^{\mathrm{c}}_{\mathrm{Ct}}$ is a congruence over all the operators.*

PROOF: It is straightforward to check that $\simeq^{\mathrm{c}}_{\mathrm{Ct}}$ is a congruence over first-order input prefixing. For the remaining operators, the result follows from the corresponding one for $\simeq_{\mathrm{Ct}}$ (Theorem 4.2.7). □

Similarly, using Proposition 4.2.6 we can infer that $\simeq^{\mathrm{c}}_{\mathrm{Ct}}$ is preserved by higher-order substitutions and by the metanotation $G\langle F\rangle$. That is, on open agents, $F_1 \simeq^{\mathrm{c}}_{\mathrm{Ct}} F_2$ implies:

- $A\{F_1/X\} \simeq^{\mathrm{c}}_{\mathrm{Ct}} A\{F_2/X\}$;
- $G\langle F_1\rangle \simeq^{\mathrm{c}}_{\mathrm{Ct}} G\langle F_2\rangle$;
- $F_1\langle E\rangle \simeq^{\mathrm{c}}_{\mathrm{Ct}} F_2\langle E\rangle$.

4.3.1 Distributivity properties for replication

NOTATION. We shall write $m \star F$ as an abbreviation for

$$m(x).F\langle x\rangle \qquad \text{or} \qquad m(X).F\langle X\rangle$$

depending upon the sorts of the expressions involved, and $P\{m := F\}$ as abbreviation for

$$\boldsymbol{\nu}\, m\,(P \mid !\, m \star F)$$

For notational convenience, we define $\{m := F\}$ on abstractions as well. By η-conversion we can assume that the abstraction is of the form $(X)P$ or$(x)P$, and we define $\big((X)P\big)\{m := F\}$ as $(X)\big(P\{m := F\}\big)$, and $\big((a)P\big)\{m := F\}$ as $(a)\big(P\{m := F\}\big)$, for $a \neq m$. We let $\{m := F\}$ have the same precedence as substitution.

Remember that m is restricted, and hence *not free*, in $P\{m := F\}$, in the same way as x is not free in the λ-expression $M\{y/x\}$. Indeed, we chose curly brackets for the above abbreviation because, under a certain condition on the use of m in P and F, $\{m := F\}$ behaves just like a substitution in $P\{m := F\}$, roughly the substitution of '$F\langle E\rangle \mid Q$' (resp. '$F\langle b\rangle \mid Q$') for the prefixes '$\overline{m}\langle E\rangle.Q$' (resp. '$\overline{m}\langle b\rangle.Q$'). All results in the present and following section can be interpreted as formalisation of this fact. Intuitively, the condition on the use of m in P and F prevents m from being exported; hence, since m remains restricted, the only possible effect of $\overline{m}\langle E\rangle.Q$ is to trigger a copy of F with argument E. For example, a little thinking should convince the reader that if m is not free in E, E', P, F, then the visible behaviour of $\big(\overline{m}\langle E\rangle.\overline{m}\langle E'\rangle.P\big)\{m := F\}$ is the same as $F\langle E\rangle|F\langle E'\rangle|P$.

CONVENTION. To simplify the uses of the abbreviation $\{m := F\}$, it is convenient to allow a free use of expressions of the form

$$\big(\overline{a}\langle Tr_m\rangle.P\big)\{m := F\} \qquad (*)$$

Formally, since only guarded sums are admitted in the language, it is not legal to use such an expression in a context like $[\cdot] + Q$. Instead, we should write

$$\left(\overline{a}\langle Tr_m\rangle.P + Q\right)\{m := F\} \qquad (**)$$

provided m is not free in Q. However there are simple and obvious transformations which convert any misuse of $(*)$ into a process of the form $(**)$ and vice versa, which isolate from $(**)$ a subexpression of the form $(*)$. Thus we leave these transformations implicit and we work with $(*)$. □

We first show with Theorem 4.3.3 that $\{m := F\}$ distributes over all operators. For this, Proposition 4.3.2 below is crucial to guarantee that $\{m := F\}$ distributes over higher-order substitutions. What happens here is therefore similar to what we did in Section 4.2.2 to prove the congruences of $\simeq_{\mathrm{Ct}}$. In both cases the results for higher-order substitution, namely Propositions 4.2.6 and 4.3.2, are very delicate; but because the technicalities on which they are based will not be needed later, their proofs have been placed in an appendix.

Proposition 4.3.2 *Let $A, E, F \in HO\pi^{\circ}$. Suppose $m \notin fn(F)$, and that m appears free only in negative subject position in A, E. Then*

$$A\{E/X\}\{m := F\} \simeq^{c}_{\mathrm{Ct}} A\{m := F\}\{E\{m := F\}/X\}.$$

PROOF: See Appendix C. □

Theorem 4.3.3 (distributivity of $\{m := F\}$)) *Suppose that $m \notin fn(F)$ and that m appears free only in negative subject position in α, P_i, P, Q, E. Then the following results on open agents hold:*

1. $\left(\sum_{i\in I} P_i\right)\{m := F\} \simeq^{c}_{\mathrm{Ct}} \sum_{i\in I}\left(P_i\{m := F\}\right)$.
2. $\left(\boldsymbol{\nu}\, a\, P\right)\{m := F\} \simeq^{c}_{\mathrm{Ct}} \boldsymbol{\nu}\, a\left(P\{m := F\}\right)$, *if* $a \notin fn(F) \cup m$.
3. $\left(P \mid Q\right)\{m := F\} \simeq^{c}_{\mathrm{Ct}} P\{m := F\} \mid Q\{m := F\}$.

4. $(!\,P)\,\{m := F\} \simeq^{c}_{Ct} !(P\,\{m := F\})$.

5. $([a = b]P)\,\{m := F\} \simeq^{c}_{Ct} [a = b](P\,\{m := F\})$, *if* $a, b \neq m$.

6. $X\langle E\rangle\,\{m := F\} \simeq^{c}_{Ct} X\langle E\,\{m := F\}\rangle$.

7. *Suppose* α *does not bind* m *or free names or variables of* F. *Then*

$$\big(\alpha.P\big)\{m := F\} \simeq^{c}_{Ct} \begin{cases} \big(\alpha.(P\,\{m := F\})\big)\,\{m := F\} & \text{if } m \text{ is free in } \alpha \\ \alpha.(P\,\{m := F\}) & \text{if } m \text{ is not free in } \alpha \end{cases}$$

8. $(\overline{a}\langle E\rangle.P)\,\{m := F\} \simeq^{c}_{Ct} \big(\overline{a}\langle E\,\{m := F\}\rangle.P\big)\,\{m := F\}$.

PROOF: Each case either can be easily derived using Proposition 4.3.2, or is straightforward on its own. □

4.4 Weak context bisimulation and congruence

The corresponding of $\simeq_{Ct}$ and $\simeq^{c}_{Ct}$ in the weak case are called *weak context bisimulation* and *weak context congruence*. They are denoted by $\approxeq_{Ct}$ and $\approxeq^{c}_{Ct}$, respectively. All congruence results shown for $\simeq_{Ct}$ and $\simeq^{c}_{Ct}$ can be extended to $\approxeq_{Ct}$ and $\approxeq^{c}_{Ct}$, with a completely analogous proof. In particular, we have:

Theorem 4.4.1

1. $\approxeq_{Ct}$ *is a congruence over all operators but first-order prefixing.*

2. $\approxeq^{c}_{Ct}$ *is a congruence over all operators.* □

A simple and useful lemma is the following:

Lemma 4.4.2 *For each* $P \in HO\pi^{\circ}$, *it holds that* $\tau.P \approxeq^{c}_{Ct} P$. □

Notice that if the language for HOπ did not require only *guarded* sums, then it would not be true that $\approxeq_{Ct}$ is a congruence over the sum operator; also, in Lemma 4.4.2, the congruence $\approxeq^{c}_{Ct}$ should then be replaced by the bisimulation $\approxeq_{Ct}$. In fact, in languages where sum can be used freely, the congruence between $\tau.P$ and P breaks down in contexts of the form $[\cdot] + Q$.

4.4.1 Triggers

A *trigger* is an agent whose only functionality is to activate (to trigger) a copy of another agent, providing it with its arguments. In the reduced HOπ we are dealing with, a trigger has only the form (depending upon the sort)

$$(x)\overline{m}\langle x\rangle.\mathbf{0}, \qquad \text{or} \qquad (X)\overline{m}\langle X\rangle.\mathbf{0}.$$

We write Tr_m to represent a trigger whose free name is m. We show now that using triggers, any subagent can be factorised out from a given agent. This is extremely useful for reasoning with higher-order processes. For instance, when comparing the behaviour of two processes it allows us to "isolate" subcomponents which might cause differences, so that the analysis can be concentrated on them.

Example 4.4.3 Consider the process

$$P \stackrel{def}{=} \left(\overline{a}\langle b\rangle.F\langle b\rangle + \alpha.R\right) \mid F\langle b'\rangle$$

We can factorise F out using triggers, thus obtaining

$$\begin{aligned} Q &\stackrel{def}{=} \left(\left(\overline{a}\langle b\rangle.Tr_m\langle b\rangle + \alpha.R\right) \mid Tr_m\langle b'\rangle\right)\{m := F\} \\ &= \left(\left(\overline{a}\langle b\rangle.\overline{m}\langle b\rangle. + \alpha.R\right) \mid \overline{m}\langle b'\rangle\right)\{m := F\} \end{aligned}$$

Theorem 4.4.7 will ensure that $P \approx^{c}_{\mathrm{Ct}} Q$. The difference in Q is that F is not activated immediately upon availability of its argument, which instead, is received via m. We can think of m as acting as a *pointer* which adds a level of indirection in the path leading to F. □

For the proof of the factorisation theorem we need some auxiliary lemmas.

Lemma 4.4.4 *In HOπ the following equalities hold:*

1. $\boldsymbol{\nu}\, a\,(\overline{a}\langle b\rangle.P \mid a(x).Q) \simeq^{c}_{\mathrm{Ct}} \tau.\boldsymbol{\nu}\, a\,(P \mid Q\{b/x\})$ *and, similarly,* $\boldsymbol{\nu}\, a\,(\overline{a}\langle E\rangle.P \mid a(X).Q) \simeq^{c}_{\mathrm{Ct}} \tau.\boldsymbol{\nu}\, a\,(P \mid Q\{E/X\})$.
2. $P\,\{m := F\} \simeq^{c}_{\mathrm{Ct}} P$ *if* $m \notin fn(P)$. □

Lemma 4.4.5 *Let $F, E \in HO\pi^\circ$ and $m \notin fn(F, E)$. Then*
$(\overline{m}\langle b\rangle.\mathbf{0})\,\{m := F\} \simeq^{c}_{\mathrm{Ct}} \tau.F\langle b\rangle$ *and* $(\overline{m}\langle E\rangle.\mathbf{0})\,\{m := F\} \simeq^{c}_{\mathrm{Ct}} \tau.F\langle E\rangle$

PROOF: Immediate using Lemma 4.4.4. □

In the version of the factorisation theorem which we present, only abstractions can be factorised out; it is however straightforward to generalise it to any agents. We derive the factorisation theorem from Lemma 4.4.6 which shows us precisely that the effect of using a trigger is to add a τ-action on the head of the replaced expression. We use $\tau \star F$ as abbreviation for $(a)\tau.F\langle a\rangle$ or $(E)\tau.F\langle E\rangle$, depending on whether F is a first-order or a higher-order abstraction.

Lemma 4.4.6 *For every $A, F \in HO\pi^\circ$ with $m \notin fn(A, F)$, it holds that*

$$A\{\tau \star F/X\} \simeq^{c}_{\mathrm{Ct}} A\{Tr_m/X\}\,\{m := F\}$$

PROOF: By induction on the structure of A. The basic case is when $A = Y$. If $Y = X$ use Lemma 4.4.5; otherwise it is immediate. Now the inductive cases.

Case (a) $A = P_1 \mid P_2$

We have:

$$\begin{aligned}
(P_1 \mid P_2)\{\tau \star F/X\} &= \\
P_1\{\tau \star F/X\} \mid P_2\{\tau \star F/X\} &\simeq^{c}_{\mathrm{Ct}} \quad \text{(induction twice)} \\
P_1\{Tr_m/X\}\,\{m := F\} \mid P_2\{Tr_m/X\}\,\{m := F\} &\simeq^{c}_{\mathrm{Ct}} \quad \text{(Theorem 4.3.3(3))} \\
\big(P_1\{Tr_m/X\} \mid P_2\{Tr_m/X\}\big)\,\{m := F\} &= \\
(P_1 \mid P_2)\{Tr_m/X\}\,\{m := F\}
\end{aligned}$$

Case (b) $A = \sum_{i\in I} P_i$, $A = \boldsymbol{\nu}\, a\, P$, $A = [a = b]P$, or $A = !\,P$

Similar to above.

Case (c) $A = (a)P_1$ or $A = (Y)P_1$

Easy.

Case (d) $A = \overline{a}\langle E\rangle.P$

We have:

$$\begin{aligned}
(\overline{a}\langle E\rangle.P)\{\tau \star F/X\} &= \\
\overline{a}\big\langle E\{\tau \star F/X\}\big\rangle.\big(P\{\tau \star F/X\}\big) &\simeq^{c}_{\mathrm{Ct}} \quad \text{(induction twice)} \\
\overline{a}\big\langle E\{Tr_m/X\}\ \{m := F\}\big\rangle.\big(P\{Tr_m/X\}\ \{m := F\}\big) &\simeq^{c}_{\mathrm{Ct}} \quad \text{(Theorem 4.3.3(7))} \\
\big(\overline{a}\big\langle E\{Tr_m/X\}\ \{m := F\}\big\rangle.P\{Tr_m/X\}\big)\ \{m := F\} &\simeq^{c}_{\mathrm{Ct}} \quad \text{(Theorem 4.3.3(8))} \\
\big(\overline{a}\big\langle E\{Tr_m/X\}\big\rangle.P\{Tr_m/X\}\big)\ \{m := F\} &= \\
(\overline{a}\langle E\rangle.P)\{Tr_m/X\}\ \{m := F\}
\end{aligned}$$

Case (e) $A = \alpha.P$ and α is not a higher-order output.

Use the induction hypothesis and Theorem 4.3.3(7).

Case (f) $A = Y\langle E\rangle$

Let us suppose $X = Y$ (the case $X \neq Y$ is simpler and requires Theorem 4.3.3(6)). We have:

$$\begin{aligned}
\big(X\langle E\rangle\big)\{\tau \star F/X\} &= \\
X\{\tau \star F/X\}\big\langle E\{\tau \star F/X\}\big\rangle &\simeq^{c}_{\mathrm{Ct}} \quad \text{(induction twice)} \\
\big(X\{Tr_m/X\}\ \{m := F\}\big)\big\langle E\{Tr_m/X\}\ \{m := F\}\big\rangle &= \\
\big(Tr_m\ \{m := F\}\big)\big\langle E\{Tr_m/X\}\ \{m := F\}\big\rangle &= \\
\big(\overline{m}\big\langle E\{Tr_m/X\}\ \{m := F\}\big\rangle.\mathbf{0}\big)\ \{m := F\} &\simeq^{c}_{\mathrm{Ct}} \quad \text{Theorem 4.3.3(8)} \\
\big(\overline{m}\big\langle E\{Tr_m/X\}\big\rangle.\mathbf{0}\big)\ \{m := F\} &= \\
Tr_m\langle E\{Tr_m/X\}\rangle\ \{m := F\} &= \\
X\langle E\rangle\{Tr_m/X\}\ \{m := F\}
\end{aligned}$$

□

Theorem 4.4.7 (factorisation theorem) *For every $A, F \in HO\pi^{\circ}$ with $m \notin fn(A, F)$, it holds that*

$$A\{F/X\} \mathrel{\approx\!\!\!\!\!\!\approx}^{c}_{\mathrm{Ct}} A\{Tr_m/X\}\ \{m := F\}.$$

PROOF: From Lemma 4.4.6, $A\{Tr_m/X\}\ \{m := F\} \simeq^c_{Ct} A\{\tau \star F/X\}$. From Lemma 4.4.2, we infer $\tau \star F \cong^c_{Ct} F$, hence $A\{\tau \star F/X\} \cong^c_{Ct} A\{F/X\}$. Since $\simeq^c_{Ct} \subseteq \cong^c_{Ct}$, this proves the theorem. □

Before leaving this section, let us present a useful corollary of the factorisation theorem, which relates higher-order input actions and "triggered" input actions (i.e. inputs of triggers). We shall need this result when proving a simpler alternative characterisation of context bisimulation, based on the use of triggers.

Corollary 4.4.8 *If $m \notin fn(P, F)$,*

1. *whenever $P \xrightarrow{a\langle F\rangle} P'$, there exists P'' s.t.*
   $P \xrightarrow{a\langle Tr_m\rangle} P''$ *and* $P' \cong^c_{Ct} P''\{m := F\}$;
2. *whenever $P \xrightarrow{a\langle Tr_m\rangle} P''$ there exists P' s.t. $P \xrightarrow{a\langle F\rangle} P'$ and*
   $P' \cong^c_{Ct} P''\{m := F\}$.

PROOF: Use the factorisation theorem and Lemma 4.1.3. □

4.5 Triggered agents

We present in this section the subclass of *triggered agents* and the mapping $\mathcal{T}$ from agents to triggered agents s.t. $A \cong^c_{Ct} \mathcal{T}[\![A]\!]$.

Preliminaries

In the following we suppose that Ob is a *downward-closed* sorting, that is if $s \mapsto (S) \in Ob$, then for some s', we have $s' \mapsto S \in Ob$. If Ob is not already downward-closed, then it can easily be extended to make it so.

The downward-closure property is necessary to be able to use triggers in the object part of higher-order actions. For instance, if $a : s \mapsto (S)$, then for the expression $\overline{a}\langle Tr_m\rangle.P$ to be legal, it must be $m : s' \mapsto S$, for some s'.

To define triggered agents, we first have to choose the subsorting SOb from which to select the free names of the triggers used. We introduce SOb to guarantee that if two triggers Tr_m and $Tr_{m'}$ have the same sort, then the names m, and m' belong to the same subject sort. This condition will be very useful in Chapter 5, to ensure the well-sortness of the encoding from triggered to π-calculus agents. It is also necessary for the definition of triggered bisimulation in Section 4.6, where it allows us to consider Tr_m and $Tr_{m'}$ α-convertible if m and m' are bound. We simply require that in SOb there is one and only subject sort for each object sort which appears in Ob; to put it formally, SOb must satisfy the two following conditions:

1. An object sort must appear at most once in SOb; i.e. if $s_1 \mapsto S_1 \in SOb$ and $s_2 \mapsto S_2 \in SOb$, then $S_1 \neq S_2$.

2. Each object sort which appears in Ob must be represented in SOb; i.e. if $s \mapsto S \in Ob$, then there exists s' s.t. $s' \mapsto S \in SOb$.

We denote the class of closed triggered agents by $THO\pi$ and the class of open triggered agents by $THO\pi^\circ$. $THO\pi$ is parametric on SOb, since the choice of SOb in general is not unique and different choices give rise to different classes of triggered agents. We omit the reference to SOb for notational simplicity. In the following we assume that m, m' range over names with sort in SOb (i.e. SOb is defined on the sorts of m and m').

4.5.1 Definition of the class

The technique for the treatment of higher-order communications in triggered agents is the following. Instead of communicating an agent F explicitly, like in $P_1 = \overline{a}\langle F\rangle.Q$, what is transmitted is a trigger to F, that is the capability of activating an arbitrary number of copies of F, like in $P_2 = (\overline{a}\langle Tr_m\rangle.Q)\{m := F\}$, for $m \notin fn(Q, F)$. It is important to understand that from the point of view of an agent willing to perform an input at a, it does not make any difference whether it

interacts with P_1 or P_2, although in one case the agent received is F, in the other case is Tr_m. To see this, consider what happens when the input is used with, say, argument E. In the first case the process $F\langle E\rangle$ is activated, in the second one $\overline{m}\langle E\rangle.\mathbf{0}$; but since m is restricted, the only effect of the latter is to produce an interaction with P_2 which activates $F\langle E\rangle$. Therefore, the only difference w.r.t. the case of communication with P_1 is an additional internal interaction, which is not observable.

We could define a process P *triggered* if the following condition holds:

- If only triggers are received as inputs, then whenever a descendant of P can perform an output, this must have been caused by a subprocess of the form

$$\left(\overline{a}\langle Tr_m\rangle.P\right)\{m := F\}. \qquad (\dagger)$$

Similarly, we could say that abstractions and open agents are triggered if they are transformed into triggered processes by the instantiation of their variables with triggers.

Now, this is a *semantic* condition, and as such, in general difficult to verify. Let us see if we can translate it into a *syntactic* condition. We certainly want all higher-order outputs to be in the above triggered form; but is this enough? Consider the process

$$P \stackrel{def}{=} a(X).X\langle F\rangle$$

When P receives a trigger Tr_m at a, it becomes $\overline{m}\langle F\rangle.\mathbf{0}$, which is not of the form $(\dagger)$. This shows that we also need a condition on applications; they must be in triggered form too. If we adjust P to conform to this, we get

$$P' \stackrel{def}{=} a(X).\left(X\langle Tr_m\rangle\{m := F\}\right)$$

These two conditions, on outputs and applications, only fail for variables representing higher-order abstractions. Suppose X is one of such variables and let us instantiate it with the trigger $Tr_m = (Y)\overline{m}\langle Y\rangle.\mathbf{0}$; if we now use the trigger $Tr_{m'}$ as

actual parameter for Y, we get $\overline{m}\langle Tr_{m'}\rangle.\mathbf{0}$, which is not of the form (†). Another way of showing the problem is by applying an η rule to X: We get $(Y)X\langle Y\rangle$, in which the application $X\langle Y\rangle$ is not triggered. This guides us also to a solution for the problem: we require that each non-binding occurrence of a variable is followed by its argument, so to make explicit all potential applications. For uniformity (and also to make smoother the mapping in Chapter 5 from triggered to first-order agents), we impose this condition also on variables representing first-order abstractions.

Summarising, an agent is triggered if each higher-order output and applications is in triggered form, and each non-binding occurrence of a variable — except those inside a trigger — is followed by its arguments. The formal definition of the class is given below. For this definition, the reader should keep in his/her mind the reduce syntax for HOπ agents, described in Section 4.1, which we are using in this chapter, and the convention on the use of the abbreviation $\{m := F\}$ in presence of sums, given in Section 4.3.1.

Definition 4.5.1 (triggered agents) *The class of* triggered agents *is the subclass of HOπ agents derived from the following grammar*

$$\begin{array}{rcl} P & :: & \sum_{i\in I} N_i \quad | \quad P_1 \,|\, P_2 \quad | \quad \boldsymbol{\nu}\, x\, P \quad | \quad [x=y]P \quad | \quad !\,P \quad | \\ & & X\langle Tr_m\rangle\,\{m := F\} \quad | \quad X\langle x\rangle \end{array}$$

where I is a finite indexing set and N and F are of the form

$$\begin{array}{rcl} N & :: & (\overline{x}\langle Tr_m\rangle.P)\,\{m := F\} \quad | \quad \overline{x}\langle y\rangle.P \quad | \quad x(X).P \quad | \quad x(y).P \\ F & :: & (X)P \quad | \quad (x)P \end{array}$$

□

The class $THO\pi^{\circ}$ is closed under instantiation of free variables with triggers and under the metanotation $\{m := F\}$.

Lemma 4.5.2 *If $A \in THO\pi^{\circ}$, then for every Tr_m we have $A\{Tr_m/X\} \in THO\pi^{\circ}$.*

PROOF: By induction on the structure of A. We only look at the case in which $A = X\langle Tr_{m'}\rangle\ \{m' := F\}$; all the others are trivial. The trigger Tr_m must be of the form $(Y)\overline{m}\langle Y\rangle.\mathbf{0}$ and then we have

$$A\{Tr_m/X\} = Tr_m\langle Tr_{m'}\rangle\ \{m' := F\} = \left(\overline{m}\langle Tr_{m'}\rangle.\mathbf{0}\right)\ \{m' := F\} \in THO\pi^{\circ}. \quad \square$$

Lemma 4.5.3 *If $P, F \in THO\pi^{\circ}$, then $P\ \{m := F\} \in THO\pi^{\circ}$.*

PROOF: Suppose F is of the form $(X)Q$, for $Q \in THO\pi^{\circ}$; the case $F = (x)P$ is analogous. Then $P\ \{m := F\}$ abbreviates

$$\boldsymbol{\nu}\, m\left(P \mid !\, m(Y).F\langle Y\rangle\right) = \boldsymbol{\nu}\, m\left(P \mid !\, m(Y).Q\{Y/X\}\right)$$

where by hypothesis, $P \in THO\pi^{\circ}$ and, since $Q \in THO\pi^{\circ}$, also $Q\{Y/X\} \in THO\pi^{\circ}$. Therefore, from the grammar of Definition 4.5.1, we get that the process $\boldsymbol{\nu}\, m\left(P \mid !\, m(Y).Q\{Y/X\}\right)$ is in $THO\pi^{\circ}$. $\square$

4.5.2 The transformation to triggered agents

The transformation $\mathcal{T}$ transforms every well-sorted agent into triggered form. It is formally defined in Table 4–1. It is assumed that m is fresh, i.e. it does not occur in the source agent. The rules for higher-order application and output are the core of $\mathcal{T}$, and are motivated by the discussion in the introduction of the previous section. In the rule for variable, an η-rule is employed; elsewhere $\mathcal{T}$ acts as an homomorphism. In the rule for variable, when X is a higher-order abstraction, a new variable is introduced. The termination of $\mathcal{T}$ is guaranteed by the fact that the sort of Y is smaller than the one of X (in the sense that if the sort of X is (S), then the sort of Y is S). Note that this introduces a dependency from the sorting in the encoding.

Remark 4.5.4 Consider the rule for higher-order application $\mathcal{T}[\![X\langle F\rangle]\!]$. If $X : (S)$ and S is not among the object sorts appearing in the sorting Ob, then its translation $X\langle Tr_m\rangle\ \{m := \mathcal{T}[\![F]\!]\}$ does not make sense because m has to belong

$$\mathcal{T}[\![X]\!] \stackrel{def}{=} \begin{cases} (Y)\mathcal{T}[\![X\langle Y\rangle]\!] & \text{if } X \text{ is a higher-order abstraction} \\ (b)X\langle b\rangle & \text{otherwise} \end{cases}$$

$$\mathcal{T}[\![\alpha.P]\!] \stackrel{def}{=} \begin{cases} (\overline{a}\langle Tr_m\rangle.\mathcal{T}[\![P]\!])\,\{m := \mathcal{T}[\![F]\!]\} & \text{if } \alpha = \overline{a}\langle F\rangle \\ \alpha.\mathcal{T}[\![P]\!] & \text{otherwise} \end{cases}$$

$$\begin{array}{llll} \mathcal{T}[\![X\langle F\rangle]\!] \stackrel{def}{=} X\langle Tr_m\rangle\,\{m := \mathcal{T}[\![F]\!]\} & & & \\ \mathcal{T}[\![X\langle b\rangle]\!] \stackrel{def}{=} X\langle b\rangle & & \mathcal{T}[\![!\,P]\!] \stackrel{def}{=} & !\mathcal{T}[\![P]\!] \\ \mathcal{T}[\![P \mid Q]\!] \stackrel{def}{=} \mathcal{T}[\![P]\!] \mid \mathcal{T}[\![Q]\!] & & \mathcal{T}[\![\sum_{i\in I} P_i]\!] \stackrel{def}{=} & \sum_{i\in I}\mathcal{T}[\![P_i]\!] \\ \mathcal{T}[\![\boldsymbol{\nu}\, a\, P]\!] \stackrel{def}{=} \boldsymbol{\nu}\, a\, \mathcal{T}[\![P]\!] & & \mathcal{T}[\![[a=b]P]\!] \stackrel{def}{=} & [a=b]\mathcal{T}[\![P]\!] \\ \mathcal{T}[\![(X)P]\!] \stackrel{def}{=} (X)\mathcal{T}[\![P]\!] & & \mathcal{T}[\![(a)P]\!] \stackrel{def}{=} & (a)\mathcal{T}[\![P]\!] \end{array}$$

Table 4–1: The transformation $\mathcal{T}$.

to a sort s with $s \mapsto S$. The problem is not serious. One could just extend the sorting to include a sort $s \mapsto S$. Alternatively, one could exclude agents using such variables from the domain of $\mathcal{T}$. In fact, these are rather uninteresting agents; in particular there is no closed process (in a closed process the only way to introduce a variable $X : (S)$ is by an input action $a(X)$; but then, for some s, it must be $a : s \mapsto ((S))$ and by the downward-closure property of Ob, some s' exists s.t. $s' \mapsto S \in Ob$). □

Example 4.5.5 *Let* $F \stackrel{def}{=} (b)\mathbf{0} : S$ *and* $X : (S)$*:*

1. *If* $P_1 = X\langle F\rangle \mid \overline{b}\langle F\rangle$, *then*
$$\mathcal{T}[\![P_1]\!] = X\langle Tr_m\rangle\,\{m := F\} \mid \overline{b}\langle Tr_m\rangle\,\{m := F\}$$

2. *If* $P_2 = a(X).Z\langle X\rangle$, *then*
$$\mathcal{T}[\![P_2]\!] = a(X).\Big(Z\langle Tr_m\rangle\,\Big\{m := (Y)\Big(X\langle Tr_{m'}\rangle\,\{m' := (b)Y\langle b\rangle\}\Big)\Big\}\Big)$$ □

Let us also illustrate how the translation works on reductions. In the two examples we give, we use the algebraic law of Lemma 4.4.4, namely

$$P\,\{m := F\} \simeq^{c}_{\mathrm{Ct}} P \qquad \text{if } m \text{ not free in } P$$

There are two dimensions at which $\mathcal{T}$ expands the number of reductions. One is *horizontal.* If a transmitted agent F is used by its recipient n times, n interactions are required in the triggered agent to activate the copies of F.

Example 4.5.6 *Let* $P \stackrel{def}{=} \overline{a}\langle F\rangle.Q \mid a(Y).(Y\langle b\rangle \mid Y\langle c\rangle)$. *Then*

$$P \longrightarrow P' \stackrel{def}{=} Q \mid F\langle b\rangle \mid F\langle c\rangle.$$

In $\mathcal{T}[\![P]\!]$ *this is simulated using two additional reductions:*

$$\begin{array}{rcl}
\mathcal{T}[\![P]\!] & \stackrel{def}{=} & (\overline{a}\langle Tr_m\rangle.\mathcal{T}[\![Q]\!])\,\{m := \mathcal{T}[\![F]\!]\} \mid a(Y).(Y\langle b\rangle \mid Y\langle c\rangle) \\
& \equiv & \big(\overline{a}\langle Tr_m\rangle.\mathcal{T}[\![Q]\!] \mid a(Y).(Y\langle b\rangle \mid Y\langle c\rangle)\big)\,\{m := \mathcal{T}[\![F]\!]\} \\
& \longrightarrow & \big(\mathcal{T}[\![Q]\!] \mid Tr_m\langle b\rangle \mid Tr_m\langle c\rangle\big)\,\{m := \mathcal{T}[\![F]\!]\} \\
& = & \big(\mathcal{T}[\![Q]\!] \mid \overline{m}\langle b\rangle \mid \overline{m}\langle c\rangle\big)\,\{m := \mathcal{T}[\![F]\!]\} \\
& \longrightarrow\longrightarrow\equiv & \big(\mathcal{T}[\![Q]\!] \mid \mathcal{T}[\![F]\!]\langle b\rangle \mid \mathcal{T}[\![F]\!]\langle c\rangle\big)\,\{m := \mathcal{T}[\![F]\!]\} \\
& \simeq^{c}_{\mathrm{Ct}} & \mathcal{T}[\![Q]\!] \mid \mathcal{T}[\![F]\!]\langle b\rangle \mid \mathcal{T}[\![F]\!]\langle c\rangle \;=\; \mathcal{T}[\![P']\!]
\end{array}$$

The other way to add interactions is *vertical* and takes its significance from the ω-order nature of HOπ. It arises with higher-order abstractions when, after the abstraction itself, one also has to trigger its arguments. This may give rise to interesting chains of activations. Below we show a simple case, in which the abstraction is of order two.

Example 4.5.7 Suppose $F \stackrel{def}{=} (b)\overline{b}\langle d\rangle : (s)$ and $G \stackrel{def}{=} (X)\big(Q \mid X\langle c\rangle\big) : ((s))$. Let $P \stackrel{def}{=} \overline{a}\langle G\rangle \mid a(Y).Y\langle F\rangle$; then $P \longrightarrow G\langle F\rangle = Q \mid F\langle c\rangle = Q \mid \overline{c}\langle d\rangle$. On the other hand, we have

$$\begin{aligned}
\mathcal{T}[\![P]\!] &\stackrel{def}{=} \overline{a}\langle Tr_{m_G}\rangle\,\{m_G := \mathcal{T}[\![G]\!]\} \mid a(Y).\big(Y\langle Tr_{m_F}\rangle\,\{m_F := \mathcal{T}[\![F]\!]\}\big) \\
&\longrightarrow\equiv \big(\overline{m_G}\langle Tr_{m_F}\rangle\,\{m_F := \mathcal{T}[\![F]\!]\}\big)\,\{m_G := \mathcal{T}[\![G]\!]\} \\
&\longrightarrow\equiv \big(\mathcal{T}[\![G]\!]\langle Tr_{m_F}\rangle\,\{m_F := \mathcal{T}[\![F]\!]\}\big)\,\{m_G := \mathcal{T}[\![G]\!]\} \\
&\qquad\qquad \text{(a copy of } G \text{ is activated)} \\
&\simeq^{c}_{Ct} (\mathcal{T}[\![Q]\!] \mid \overline{m_F}\langle c\rangle)\,\{m_F := \mathcal{T}[\![F]\!]\} \\
&\longrightarrow\simeq^{c}_{Ct} \mathcal{T}[\![Q]\!] \mid \overline{c}\langle d\rangle = \mathcal{T}[\![Q \mid \overline{c}\langle d\rangle]\!] \qquad \text{(a copy of } F \text{ is activated)}
\end{aligned}$$

Now we have to prove the correctness of $\mathcal{T}$. For this, we have to show that $\mathcal{T}$ returns a $\approx^{c}_{Ct}$ agent which is triggered and respects Ob.

Theorem 4.5.8 (correctness of $\mathcal{T}$) *For each $A \in HO\pi^{\circ}$ with $A :: Ob$, we have*

1. *$\mathcal{T}[\![A]\!]$ is triggered;*
2. *$\mathcal{T}[\![A]\!] :: Ob$;*
3. *$\mathcal{T}[\![A]\!] \approx^{c}_{Ct} A$.*

PROOF:

(1,2) : Immediate from the definition of $\mathcal{T}$.

(3) Induction on the structure of A. Using some η-conversion we can assume that each non-binding occurrence of a variable is followed by its arguments. Then there are only two cases in which $\mathcal{T}$ does not act as an homomorphism.

- $A = \overline{a}\langle F\rangle.P$

  We have, for $m \notin fn(A)$:

$$\begin{aligned}
\overline{a}\langle F\rangle.P &\approx^{c}_{Ct} \quad \text{(Theorem 4.4.7)} \\
\big(\overline{a}\langle Tr_m\rangle.P\big)\,\{m := F\} &\approx^{c}_{Ct} \quad \text{(induction twice)} \\
\big(\overline{a}\langle Tr_m\rangle.\mathcal{T}[\![P]\!]\big)\,\{m := \mathcal{T}[\![F]\!]\} &= \mathcal{T}[\![A]\!]
\end{aligned}$$

- $A = X\langle F\rangle$

  Similar to the previous case.

It is helpful to see precisely the operational correspondence between P and $\mathcal{T}[\![P]\!]$. This is important to understand better the relationship between them, but even more, because it gives us the intuitions for the definition of a simpler characterisation of $\cong_{Ct}$, by evidencing some minimum test for higher-order actions (we shall develop this further in Section 4.7.1). An action $P \xrightarrow{\tau} P'$ whose derivation uses the rule COM with higher-order actions as premises (i.e., an agent is transmitted) is called a *higher-order reduction*; if the premises in the rule COM are first-order actions, then $P \xrightarrow{\tau} P'$ is called a *first-order reduction*. The result of the Lemma 4.5.9 is organised in the four clauses (a), (b), (c) and (d), which treat respectively, higher-order outputs, higher-order inputs, higher-order reductions and, finally, first-order inputs/outputs and reductions. Actually, for higher-order inputs we only deal with inputs of triggers, because — as we shall see better in Section 4.7.1 — this is the interesting case. In clause (c), F represents the abstraction exchanged in the interaction which has taken place in P. By contrast, in the corresponding triggered agent $\mathcal{T}[\![P]\!]$, the agent transmitted is a trigger Tr_m to $\mathcal{T}[\![F]\!]$. This is the reason for the peculiar structure of the derivative of $\mathcal{T}[\![P]\!]$. First we need a lemma:

Lemma 4.5.9 *Let $A \in HO\pi^\circ$ and $m \notin fn(A)$. Then*
$\mathcal{T}[\![A\{Tr_m/X\}]\!] = \mathcal{T}[\![A]\!]\{Tr_m/X\}$.

PROOF: Induction on the structure of A.

Lemma 4.5.10 (operational correspondence for $\mathcal{T}$ on strong transitions) *Let $m \notin fn(P, \mathcal{T}[\![P]\!])$.*

1. *(a) If $P \xrightarrow{(\nu\,\tilde{b})\overline{a}\langle F\rangle} P'$, then $\mathcal{T}[\![P]\!] \xrightarrow{(\nu\,m)\overline{a}\langle Tr_m\rangle} \equiv \mathcal{T}[\![\boldsymbol{\nu}\,\tilde{b}(P' \mid !m \star F)]\!]$;*

   (b) if $P \xrightarrow{a\langle Tr_m\rangle} P'$, then $\mathcal{T}[\![P]\!] \xrightarrow{a\langle Tr_m\rangle} \mathcal{T}[\![P']\!]$;

   (c) if $P \xrightarrow{\tau} P'$ is a higher-order reduction, then $\tilde{b}, G, F$ exist s.t. $P' \equiv \boldsymbol{\nu}\,\tilde{b}\,G\langle F\rangle$, and $\mathcal{T}[\![P]\!] \xrightarrow{\tau} \equiv \mathcal{T}[\![\boldsymbol{\nu}\,\tilde{b}\left(G\langle Tr_m\rangle\,\{m := F\}\right)]\!]$;

(d) if $P \xrightarrow{\mu} P'$, *and* μ *is a first-order input or output or a first-order reduction, then* $\mathcal{T}[\![P]\!] \xrightarrow{\mu} \mathcal{T}[\![P']\!]$.

2. The converse of (1), i.e. :

(a) If $\mathcal{T}[\![P]\!] \xrightarrow{(\nu\, m)\overline{a}\langle Tr_m\rangle} P''$, *then* P' *exists s.t.*
$P \xrightarrow{(\nu\, \tilde{b})\overline{a}\langle F\rangle} P'$, *and* $P'' \equiv \mathcal{T}[\![\boldsymbol{\nu}\, \tilde{b}\,(P' \mid !m \star F)]\!]$;

(b) if $\mathcal{T}[\![P]\!] \xrightarrow{a\langle Tr_m\rangle} P''$ *then* P' *exists s.t.*
$P \xrightarrow{a\langle Tr_m\rangle} P'$, *and* $P'' = \mathcal{T}[\![P']\!]$;

(c) if $\mathcal{T}[\![P]\!] \xrightarrow{\tau} P''$ *is a higher-order reduction, then* $\tilde{b}, G, F$ *exist s.t.*
$P \xrightarrow{\tau} \equiv \boldsymbol{\nu}\, \tilde{b}\,(G\langle F\rangle)$ *and* $P'' \equiv \mathcal{T}[\![\boldsymbol{\nu}\, \tilde{b}\big(G\langle Tr_m\rangle\ \{m := F\}\big)]\!]$;

(d) if $\mathcal{T}[\![P]\!] \xrightarrow{\mu} P''$ *and* μ *is a first-order input or output or a first-order reduction, then* P' *exists s.t.* $P \xrightarrow{\mu} P'$, *and* $P'' = \mathcal{T}[\![P']\!]$.

PROOF: By induction on the structure of P. We consider the three main cases.

- P is of the form $\overline{a}\langle F\rangle.Q$.

  Then $\mathcal{T}[\![P]\!] = \big(\overline{a}\langle Tr_m\rangle.\mathcal{T}[\![Q]\!]\big)\ \{m := \mathcal{T}[\![F]\!]\}$. We have:

$$P \xrightarrow{\overline{a}\langle F\rangle} Q \quad \text{and}$$
$$\mathcal{T}[\![P]\!] \xrightarrow{(\nu\, m)\overline{a}\langle Tr_m\rangle} \mathcal{T}[\![Q]\!] \mid !m \star \mathcal{T}[\![F]\!] = \mathcal{T}[\![Q \mid !m \star F]\!]$$

- P is of the form $a(X).Q$.

  Then $\mathcal{T}[\![P]\!] = a(X).\mathcal{T}[\![Q]\!]$. We have:

$$P \xrightarrow{a\langle Tr_m\rangle} Q\{Tr_m/X\} \quad \text{and}$$
$$\mathcal{T}[\![P]\!] \xrightarrow{a\langle Tr_m\rangle} \mathcal{T}[\![Q]\!]\{Tr_m/X\} = \mathcal{T}[\![Q\{Tr_m/X\}]\!]$$

where the last equality holds by Lemma 4.5.9.

- P is of the form $P_1 \mid P_2$.

  Then $\mathcal{T}[\![P]\!] = \mathcal{T}[\![P_1]\!] \mid \mathcal{T}[\![P_2]\!]$. The interesting case is (c). We consider only (1.c), as (2.c) is analogous. Suppose that $P_1 \xrightarrow{a\langle F\rangle} P_1'$, and $P_2 \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle F\rangle} P_2'$ with $\widetilde{b} \cap fn(P_1) = \emptyset$. Then from the communication law, we infer $P \xrightarrow{\tau} \boldsymbol{\nu}\,\widetilde{b}(P_1' \mid P_2') \stackrel{def}{=} P'$. By the induction hypothesis, we have

$$\mathcal{T}[\![P_2]\!] \xrightarrow{(\nu\,m)\overline{a}\langle Tr_m\rangle} P_2'' \equiv \mathcal{T}[\![\boldsymbol{\nu}\,\widetilde{b}(P_2' \mid !m \star F)]\!]. \tag{4.1}$$

Let us try to use the inductive hypothesis on P_1 too. By Lemma 4.1.3, G_1 exists s.t. $P_1' = G_1\langle F\rangle$ and $P_1 \xrightarrow{a\langle Tr_m\rangle} G_1\langle Tr_m\rangle$; therefore by induction

$$\mathcal{T}[\![P_1]\!] \xrightarrow{a\langle Tr_m\rangle} \mathcal{T}[\![G_1\langle Tr_m\rangle]\!] \stackrel{def}{=} P_1'' \tag{4.2}$$

From (4.1) and (4.2) we infer

$$\begin{aligned}
\mathcal{T}[\![P]\!] \xrightarrow{\tau} P'' &= \boldsymbol{\nu}\,m\,(P_1'' \mid P_2'') \\
&\equiv \boldsymbol{\nu}\,m\,\big(\mathcal{T}[\![G_1\langle Tr_m\rangle]\!] \mid \mathcal{T}[\![\boldsymbol{\nu}\,\widetilde{b}(P_2' \mid !m \star F)]\!]\big) \\
&= \mathcal{T}[\![\boldsymbol{\nu}\,m\,\big(G_1\langle Tr_m\rangle \mid \boldsymbol{\nu}\,\widetilde{b}(P_2' \mid !m \star F)\big)]\!] \\
&\equiv \mathcal{T}[\![\boldsymbol{\nu}\,\widetilde{b}\,\big((G_1\langle Tr_m\rangle \mid P_2')\,\{m := F\}\big)]\!] \qquad (4.3)
\end{aligned}$$

Finally, we are ready to prove that (1.c) holds. Set $G \stackrel{def}{=} (X)(G_1\langle X\rangle \mid P_2')$: We have $P' = \boldsymbol{\nu}\,\widetilde{b}(P_1' \mid P_2') = \boldsymbol{\nu}\,\widetilde{b}(G_1\langle F\rangle \mid P_2') = \boldsymbol{\nu}\,\widetilde{b}\,G\langle F\rangle$, and by (4.3), $P'' \equiv \mathcal{T}[\![\boldsymbol{\nu}\,\widetilde{b}\,\big(G\langle Tr_m\rangle\,\{m := F\}\big)]\!]$.

4.6 Triggered bisimulation

We show that the class $THO\pi$ of triggered agents is amenable to an analysis in which only first-order actions and higher-order actions in triggered form — that is in the form $a\langle Tr_m\rangle$ or $(\boldsymbol{\nu}\,m\,)\overline{a}\langle Tr_m\rangle$ — are employed. We call these actions *basic actions*.

We start by showing that the class $THO\pi$ is closed with respect to the production of basic actions and that every higher-order output that a process in $THO\pi$ can perform is indeed a basic action.

Lemma 4.6.1 *Let $P \in THO\pi$ and $P \xrightarrow{\mu} P'$. Then it holds that*

1. *If μ is a basic action, then $P' \in THO\pi$;*
2. *If μ is a higher-order output, then $\mu = (\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m\rangle$ and for some $P'', F, \tilde{b}$ with $m \notin fn(P'', F) \cup \tilde{b}$ we have $P' \equiv \boldsymbol{\nu}\, \tilde{b}(P'' \mid !m \star F)$; moreover, by (1) $P' \in THO\pi$.*

PROOF: By induction on the structure of P. We consider the two main cases:

Case (a) $P = a(X).Q$

If $\mu = a\langle Tr_m\rangle$, then $P \xrightarrow{\mu} P' = Q\{Tr_m/X\}$. By Lemma 4.5.2, $P' \in THO\pi$.

Case (b) $P = P_1 \mid P_2$ and $P \xrightarrow{\mu} P'$.

There are four cases to examine, according to the rule which was used to infer the action for P.

Case (b.1) $P_1 \xrightarrow{\mu} P_1'$ and $P' = P_1' \mid P_2$.

Let us look at (2), as (1) is similar (actually simpler). By induction, $\mu = (\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m\rangle$, and $P_1' \equiv \boldsymbol{\nu}\, \tilde{b}(P_1'' \mid !m \star F)$ with $m \notin fn(P_1'', F) \cup \tilde{b}$. Using α-conversion we can assume $(\tilde{b} \cup m) \cap fn(P_2) = \emptyset$. Therefore

$$P' = (\boldsymbol{\nu}\, \tilde{b}(P_1'' \mid !m \star F)) \mid P_2 \equiv \boldsymbol{\nu}\, \tilde{b}((P_1'' \mid P_2) \mid !m \star F).$$

Case (b.2) $P_2 \xrightarrow{\mu} P_2'$ and $P' = P_1 \mid P_2'$.

Similar to (b.1).

Case (b.3) $P_1 \xrightarrow{(\boldsymbol{\nu}\, \tilde{b})\overline{a}\langle F\rangle} P_1'$, $P_2 \xrightarrow{a\langle F\rangle} P_2'$ and $\mu = \tau$, $P' = \boldsymbol{\nu}\, \tilde{b}(P_1' \mid P_2')$.

By induction, $P_1' \in THO\pi$ and the output emitted by P_1 is a trigger, i.e. we have $(\boldsymbol{\nu}\, \tilde{b})\overline{a}\langle F\rangle = (\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m\rangle$. Therefore we can exploit the induction assumption also on P_2 and infer $P_2' \in THO\pi$.

Case (b.4) $P_1 \xrightarrow{a\langle F\rangle} P_1'$, $P_2 \xrightarrow{(\nu \widetilde{b})\overline{a}\langle F\rangle} P_2'$ and $\mu = \tau$, $P' = \boldsymbol{\nu}\, \widetilde{b}(P_1' \mid P_2')$.

Similar to (b.3).

Having proved that $THO\pi$ is closed w.r.t. the production of basic actions, it makes sense to define a bisimulation in which *only* basic actions are involved.

Definition 4.6.2 (triggered bisimulation) *A relation $\mathcal{R}$ on $THO\pi$ processes is a* strong triggered simulation *if $\mathcal{R}$ is a strong first-order simulation and $P\ \mathcal{R}\ Q$ implies, for $m \notin fn(P,Q)$:*

1. *whenever $P \xrightarrow{a\langle Tr_m\rangle} P'$, there exists Q' s.t. $Q \xrightarrow{a\langle Tr_m\rangle} Q'$ and $P'\ \mathcal{R}\ Q'$,*
2. *whenever $P \xrightarrow{(\nu\, m)\overline{a}\langle Tr_m\rangle} P'$, there exists Q' s.t. $Q \xrightarrow{(\nu\, m)\overline{a}\langle Tr_m\rangle} Q'$ and $P'\ \mathcal{R}\ Q'$.*

A relation $\mathcal{R}$ on $THO\pi$ processes is a strong triggered bisimulation, *briefly $\simeq_{\mathrm{Tr}}$-bisimulation, if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong triggered simulations. We say that P and Q are* strong triggered bisimilar, *briefly $P \simeq_{\mathrm{Tr}} Q$, if $P\ \mathcal{R}\ Q$, for some $\simeq_{\mathrm{Tr}}$-bisimulation $\mathcal{R}$.*

Notice that in the clauses (1) and (2) it is enough to take *some* $m \notin fn(P,Q)$. Any other choice of m — with $m \notin fn(P,Q)$ — represents the same kind of test on the processes since, intuitively, the difference between the two choices is expressed by an injective mapping on names. This observation is to evidence the simplicity of the clauses of Definition 4.6.2 compared to those in the definition of context bisimulation: In clause (2) no universal quantification and no check whatsoever on the agents emitted in the output is necessary; similarly, in clause (1) a single fresh trigger is used, whereas in context bisimulation every agent which can possibly be received is taken into account.

In line with Definition 4.6.2, $\simeq_{\mathrm{Tr}}$ is extended to higher-order abstractions and open agents employing "fresh" triggers. Thus, if F, E are higher-order abstractions and $m \notin fn(F,E)$, then $F \simeq_{\mathrm{Tr}} E$ holds if $F\langle Tr_m\rangle \simeq_{\mathrm{Tr}} E\langle Tr_m\rangle$. On open agents A_1, A_2 with $fv(A_1,A_2) = \{X_1,\ldots,X_n\}$, if $\{m_1,\ldots,m_n\}$ is a

set of distinct names not occurring free in A_1, A_2, then $A_1 \simeq_{\mathrm{Tr}} A_2$ holds if $A_1\{Tr_{m_1}, \ldots, Tr_{m_n}/X_1, \ldots, X_n\} \simeq_{\mathrm{Tr}} A_2\{Tr_{m_1}, \ldots, Tr_{m_n}/X_1, \ldots, X_n\}$. The definition of $\simeq_{\mathrm{Tr}}$ on first-order abstractions, the definition of *strong triggered congruence* ($\simeq^{\mathrm{c}}_{\mathrm{Tr}}$), *weak triggered bisimulation* and *congruence* (resp. $\approxeq_{\mathrm{Tr}}$ and $\approxeq^{\mathrm{c}}_{\mathrm{Tr}}$) are standard. Finally, for $\simeq_{\mathrm{Tr}}$ and $\approxeq_{\mathrm{Tr}}$ too, the up-to techniques for defining bisimulation described in Section 2.5.3 work fine.

4.6.1 Weak triggered and context bisimulations coincide

Lemma 4.6.3 *$P \approxeq_{\mathrm{Tr}} Q$ implies:*

1. $\boldsymbol{\nu}\, a\, P \approxeq_{\mathrm{Tr}} \boldsymbol{\nu}\, a\, Q$;
2. $P \mid R \approxeq_{\mathrm{Tr}} Q \mid R$.

PROOF: By Lemma 4.6.1 and definition of $\approxeq_{\mathrm{Tr}}$, the only actions we need to consider are communications of triggers. Then the proofs become similar to the corresponding ones for weak early bisimulation for π-calculus.

We only examine (2). For this, we prove that

$$\mathcal{R} = \{(P_1 \mid R, P_2 \mid R) \,:\, P_1, P_2, R \in THO\pi \text{ and } P_1 \approxeq_{\mathrm{Tr}} P_2\}$$

is a $\approxeq_{\mathrm{Tr}}$-bisimulation up-to restriction. Suppose $(P_1 \mid R, P_2 \mid R) \in \mathcal{R}$ and $P_1 \mid R \xrightarrow{\mu} Q_1$. We proceed by case analysis on the rule used to infer this action. If COM has been used with P_1 performing the output, then for some m, we have

$$P_1 \xrightarrow{(\nu\, m)\overline{a}\langle Tr_m\rangle} P_1',\ R \xrightarrow{a\langle Tr_m\rangle} R',\ \mu = \tau \text{ and } Q_1 = \boldsymbol{\nu}\, m\,(P_1' \mid R').$$

Since $P_1 \approxeq_{\mathrm{Tr}} P_2$, and assuming m not free in P_2, we have

$$P_2 \overset{(\nu\, m)\overline{a}\langle Tr_m\rangle}{\Longrightarrow} P_2' \approxeq_{\mathrm{Tr}} P_1', \text{ and } P_2 \mid R \overset{\tau}{\Longrightarrow} \boldsymbol{\nu}\, m\,(P_2' \mid R').$$

This is enough because $\mathcal{R}$ is a bisimulation up-to restriction and by Lemma 4.6.1, we have $P_1', P_2', R' \in THO\pi$. All remaining cases are similar.

Proposition 4.6.4 *Let $P, Q \in THO\pi$; then $P \cong_{\mathrm{Ct}} Q$ implies $P \cong_{\mathrm{Tr}} Q$.*

PROOF: We proof that

$$\mathcal{R} = \{(P_1, P_2) \ : \ P_1, P_2 \in THO\pi \ and \ P_1 \cong_{\mathrm{Ct}} P_2\}$$

is a $\cong_{\mathrm{Tr}}$-bisimulation. Let $(P_1, P_2) \in \mathcal{R}$ and suppose that $P_1 \xrightarrow{\mu} P_1'$. There are two cases:

Case (a) $\mu = (\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m \rangle$.

Remember that by Lemma 4.6.1, this is the only possible form of higher-order output for P_1 and P_2. This and $P_1 \cong_{\mathrm{Ct}} P_2$ imply that there exists P_2' s.t. $P_2 \stackrel{(\boldsymbol{\nu}\, m)\langle Tr_m \rangle}{\Longrightarrow} P_2'$ and for every G, with $m \notin fn(G)$:

$$\boldsymbol{\nu}\, m\, (G\langle Tr_m \rangle \mid P_1') \cong_{\mathrm{Ct}} \boldsymbol{\nu}\, m\, (G\langle Tr_m \rangle \mid P_2') \tag{4.4}$$

In order to close the bisimulation we have to show that $P_1' \cong_{\mathrm{Ct}} P_2'$. Lemma 4.6.1 tells us that there exist $\tilde{b_1}, \tilde{b_2}, F_1, F_2, P_1'', P_2''$ s.t.

$$P_1' \equiv \boldsymbol{\nu}\, \tilde{b_1}\left(P_1'' \mid !\, m \star F_1\right), \qquad P_2' \equiv \boldsymbol{\nu}\, \tilde{b_2}\left(P_2'' \mid !\, m \star F_2\right)$$

where $m \notin fn(F_1, F_2, P_1'', P_2'') \cup \tilde{b_1} \cup \tilde{b_2}$. Suppose m' is a fresh name. Now, changing m for m' does not affect the equivalence among processes, i.e. we have

$$\boldsymbol{\nu}\, \tilde{b_1}\left(P_1'' \mid !\, m \star F_1\right) \ \cong_{\mathrm{Ct}} \ \boldsymbol{\nu}\, \tilde{b_2}\left(P_2'' \mid !\, m \star F_2\right) \ \text{ iff}$$
$$\boldsymbol{\nu}\, \tilde{b_1}\left(P_1'' \mid !\, m' \star F_1\right) \ \cong_{\mathrm{Ct}} \ \boldsymbol{\nu}\, \tilde{b_2}\left(P_2'' \mid !\, m' \star F_2\right)$$

Therefore if we can prove the latter equivalence we get $P_1' \cong_{\mathrm{Ct}} P_2'$. The advantage with this is that we shall be able to exploit (4.4). Since m is not free in $\boldsymbol{\nu}\, \tilde{b_1}\left(P_1'' \mid !\, m' \star F_1\right)$ and $\boldsymbol{\nu}\, \tilde{b_2}\left(P_2'' \mid !\, m' \star F_2\right)$, and using the factorisation theorem 4.4.7 we have

$$\begin{aligned}
\boldsymbol{\nu}\,\tilde{b_1}\left(P_1'' \mid !m' \star F_1\right) &\approx_{\mathrm{Ct}} \\
\boldsymbol{\nu}\,\tilde{b_1}\left(\left(P_1'' \mid !m' \star Tr_m\right)\{m := F_1\}\right) &\equiv \\
\boldsymbol{\nu}\,m\left(!m' \star Tr_m \mid \boldsymbol{\nu}\,\tilde{b_1}\left(P_1'' \mid !m \star F_1\right)\right) &= \\
\boldsymbol{\nu}\,m\left(!m' \star Tr_m \mid P_1'\right) &\stackrel{def}{=} Q_1
\end{aligned}$$

and similarly,

$$\boldsymbol{\nu}\,\tilde{b_2}\left(P_2'' \mid !m' \star F_2\right) \approx_{\mathrm{Ct}} \boldsymbol{\nu}\,m\left(!m' \star Tr_m \mid P_2'\right) \stackrel{def}{=} Q_2$$

Now $Q_1 \approx_{\mathrm{Ct}} Q_2$ can be derived from (4.4), for $G = (X)\,!m' \star X$.

Case (b) μ is an input or a first-order action.

In this case from $P_1 \approx_{\mathrm{Ct}} P_2$, we derive that there exists P_2' s.t. $P_2 \stackrel{\hat{\mu}}{\Longrightarrow} P_2'$ and $P_1' \approx_{\mathrm{Ct}} P_2'$. Thus $(P_1, P_2) \in \mathcal{R}$. □

Proposition 4.6.5 *Let $P, Q \in THO\pi$; then $P \approx_{\mathrm{Tr}} Q$ implies $P \approx_{\mathrm{Ct}} Q$.*

PROOF: We show that

$$\mathcal{R} = \{(P, Q) \;:\; P \approx_{\mathrm{Tr}} Q\}$$

is a $\approx_{\mathrm{Ct}}$-bisimulation up-to $\approx_{\mathrm{Ct}}$. For $P\,\mathcal{R}\,Q$, we have to examine the higher-order actions of P. The input is more delicate and we consider it first. Suppose that $P \stackrel{a\langle F\rangle}{\Longrightarrow} P'$. We have to find Q', Q'' and P'' s.t.

$$Q \stackrel{a\langle F\rangle}{\Longrightarrow} Q' \text{ and } P' \approx_{\mathrm{Ct}} P''\,\mathcal{R}\,Q'' \approx_{\mathrm{Ct}} Q'. \tag{4.5}$$

If $P \stackrel{a\langle F\rangle}{\Longrightarrow} P'$, then there exist P_1, P_2 s.t.

$$P \Longrightarrow P_1 \stackrel{a\langle F\rangle}{\longrightarrow} P_2 \Longrightarrow P'$$

Using Corollary 4.4.8(1), there exists P_3 s.t. if $m \notin fn(P, Q)$, then

$$P_1 \stackrel{a\langle Tr_m\rangle}{\longrightarrow} P_3 \qquad \text{and} \qquad P_3\,\{m := F\} \approx^{\mathrm{c}}_{\mathrm{Ct}} P_2 \tag{4.6}$$

Moreover, $P_3 \in THO\pi$ by Lemma 4.6.1. In general, F is not triggered. We can "trigger" F using the mapping $\mathcal{T}$. Let $F' = \mathcal{T}[\![F]\!]$: By the correctness of $\mathcal{T}$ (Theorem 4.5.8(3)) we have $F' \approx^{\mathrm{c}}_{\mathrm{Ct}} F$. Hence

$$P_3\{m := F'\} \approx^{c}_{\mathrm{Ct}} P_3\{m := F\}$$

From this and (4.6), by transitivity of $\approx^{c}_{\mathrm{Ct}}$, also $P_3\{m := F'\} \approx^{c}_{\mathrm{Ct}} P_2$. Therefore, since $P_2 \Longrightarrow P'$, for some P_4,

$$P_3\{m := F'\} \Longrightarrow P_4 \approx_{\mathrm{Ct}} P' \tag{4.7}$$

It holds that $P_4 \in THO\pi$ because from $P_3 \in THO\pi$ and Lemma 4.5.3, we have $P_3\{m := F'\} \in THO\pi$ and, by Lemma 4.6.1, $THO\pi$ is closed w.r.t. basic actions. Summarising what we have obtained so far:

$$P \stackrel{a\langle F\rangle}{\Longrightarrow} P' \approx_{\mathrm{Ct}} P_4 \in THO\pi$$

Let $P'' \stackrel{def}{=} P_4$ in (4.5); it remains to find Q' and Q''. From $Q \approx_{\mathrm{Tr}} P$ and $P \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} P_3$ we get that there exists Q_1 s.t.

$$Q \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} Q_1 \quad \text{and} \quad Q_1 \approx_{\mathrm{Tr}} P_3$$

By the congruence properties of $\approx_{\mathrm{Tr}}$, we also have

$$Q_1\{m := F'\} \approx_{\mathrm{Tr}} P_3\{m := F'\} \tag{4.8}$$

Reversing the reasoning used to find P_4 and prove $P' \approx_{\mathrm{Ct}} P_4$ we find Q_2 s.t.

$$Q \stackrel{a\langle F\rangle}{\Longrightarrow} Q_2 \quad \text{and} \quad Q_2 \approx_{\mathrm{Ct}} Q_1\{m := F'\} \tag{4.9}$$

Now, from (4.7), (4.8), (4.9) and the definitions of $\approx_{\mathrm{Tr}}$, $\approx_{\mathrm{Ct}}$, we can complete the following diagram:

$$\begin{array}{ccccc} P_3\{m := F'\} & \approx_{\mathrm{Tr}} & Q_1\{m := F'\} & \approx_{\mathrm{Ct}} & Q_2 \\ \Downarrow & & \Downarrow & & \Downarrow \\ P_4 & \approx_{\mathrm{Tr}} & Q_3 & \approx_{\mathrm{Ct}} & Q_4 \end{array}$$

For $Q'' \stackrel{def}{=} Q_3$, $Q' \stackrel{def}{=} Q_4$ in (4.5), this concludes the analysis.

Now the case of higher-order output. Because $P, Q \in THO\pi$, they can only output triggers (Lemma 4.6.1). Suppose that $P \stackrel{(\nu\, m)\overline{a}\langle Tr_m\rangle}{\Longrightarrow} P'$. We can assume $m \notin fn(P,Q)$ and then, since $P \approx_{\mathrm{Tr}} Q$, for some Q' we have $Q \stackrel{(\nu\, m)\overline{a}\langle Tr_m\rangle}{\Longrightarrow} Q' \approx_{\mathrm{Tr}} P'$. By definition of $\approx_{\mathrm{Ct}}$ and $\mathcal{R}$, we have to check that for every G with $m \notin fn(G)$,

$$\boldsymbol{\nu}\, m\, (G\langle Tr_m\rangle \mid P') \approx_{\mathrm{Ct}} \mathcal{R} \approx_{\mathrm{Ct}} \boldsymbol{\nu}\, m\, (G\langle Tr_m\rangle \mid Q').$$

Appealing to the correctness of $\mathcal{T}$ (Theorem 4.5.8(3)) and the definition of $\mathcal{R}$, this holds if

$$R_1 \stackrel{def}{=} \boldsymbol{\nu}\, m\, (\mathcal{T}[\![G\langle Tr_m\rangle]\!] \mid P') \approx_{\mathrm{Tr}} \boldsymbol{\nu}\, m\, (\mathcal{T}[\![G\langle Tr_m\rangle]\!] \mid Q') \stackrel{def}{=} R_2.$$

Since $R_1, R_2 \in THO\pi$ and $P' \approx_{\mathrm{Tr}} Q'$, we can infer $R_1 \approx_{\mathrm{Tr}} R_2$ from congruence of $\approx_{\mathrm{Tr}}$ over parallel composition and restriction proved in Theorem 4.6.3. □

The previous two propositions give:

Corollary 4.6.6 *For all closed triggered processes P, Q, it holds that $P \approx_{\mathrm{Tr}} Q$ iff $P \approx_{\mathrm{Ct}} Q$.* □

4.7 Uses of triggered agents

Triggered agents are a powerful tool for reasoning with higher-order process calculi. A striking example is presented in the next chapter, where their use is essential to obtain full abstraction for a compilation from higher-order to first-order agents. We believe that triggered agents would also be helpful for the development of an algebraic theory for HOπ. In this section, they are exploited to derive two alternative characterisations of context bisimulation on the whole class of HOπ processes, i.e. not necessarily in triggered form. The first is a more efficient characterisation, that is, easier to verify. The second is a characterisation in terms of barbed equivalence.

4.7.1 Normal bisimulation

The appellation "normal bisimulation" is to indicate that it is obtained by "normalising" the clauses of context bisimulation. Let us first present the definition; then we shall comment on it.

Definition 4.7.1 (normal bisimulation) *A relation $\mathcal{R}$ on HOπ processes is a* strong normal simulation *if $\mathcal{R}$ is a strong first-order simulation and $P\ \mathcal{R}\ Q$ implies, for $m \notin fn(P,Q)$:*

1. *whenever $P \xrightarrow{a\langle Tr_m\rangle} P'$, then Q' exists s.t. $Q \xrightarrow{a\langle Tr_m\rangle} Q'$ and $P'\ \mathcal{R}\ Q'$,*
2. *whenever $P \xrightarrow{(\nu\, b)\overline{a}\langle F\rangle} P'$, then $Q', \tilde{c}, E$ exist s.t. if m is a fresh name, $Q \xrightarrow{(\nu\,\tilde{c})\overline{a}\langle E\rangle} Q'$ and $\boldsymbol{\nu}\,\tilde{b}(P' \mid !m \star F)\ \mathcal{R}\ \boldsymbol{\nu}\,\tilde{c}(Q' \mid !m \star E)$.*

A relation $\mathcal{R}$ on HOπ processes is a strong normal bisimulation, *briefly $\simeq_{\mathrm{Nr}}$-bisimulation, if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong normal simulations. We say that P and Q are* strong normal bisimilar, *briefly $P \simeq_{\mathrm{Nr}} Q$, if $P\ \mathcal{R}\ Q$, for some $\simeq_{\mathrm{Nr}}$-bisimulation $\mathcal{R}$.* □

The extension of $\simeq_{\mathrm{Nr}}$ to abstractions and open agents, its congruence $\simeq^{c}_{\mathrm{Nr}}$, and their corresponding weak versions $\approx_{\mathrm{Nr}}$ and $\approx^{c}_{\mathrm{Nr}}$ are defined similarly as for triggered bisimulation. In particular, the definitions on higher-order abstractions and open agents are obtained employing only fresh triggers. Both the strong and the weak version of normal bisimulation are equivalence and the up-to techniques described in Section 2.5.3 apply to them.

As for trigger bisimulation, we want to stress that in the clauses (1) and (2) of Definition 4.7.1, it is enough to pick *some* fresh name m, since the specific choice of the fresh name does not affect the equivalence of the resulting processes.

The idea of normal bisimulation comes from the results on the discriminanting power of triggers shown in the previous sections. To test the equivalence between $a(X).P$ and $a(X).Q$ we do not have to try every possible instantiation of X in P and Q, but it is enough to verify it with a trigger Tr_m, as long as m is fresh. In fact, by the congruence properties of $\approx_{\mathrm{Ct}}$, for any F,

$$P\{Tr_m/X\} \approx_{\mathrm{Ct}} Q\{Tr_m/X\} \qquad \text{implies}$$
$$P\{Tr_m/X\}\ \{m := F\} \approx_{\mathrm{Ct}} Q\{Tr_m/X\}\ \{m := F\}$$

and then, by Theorem 4.4.7 (the factorisation theorem),

$$P\{Tr_m/X\}\ \{m := F\} \cong_{\mathrm{Ct}} P\{F/X\},$$
$$Q\{Tr_m/X\}\ \{m := F\} \cong_{\mathrm{Ct}} Q\{F/X\}$$

which yields

$$P\{F/X\} \cong_{\mathrm{Ct}} Q\{F/X\}$$

Similar reasoning can be used to justify the clause on higher-order outputs in Definition 4.7.1; but to understand it, it is also enlighting to look back at the operational correspondence between P and $\mathcal{T}[\![P]\!]$ shown in Lemma 4.5.10, and compare clause (1.a) of that lemma with clause (2) in Definition 4.7.1. It may be useful however to see why in the latter the requirement

$$\boldsymbol{\nu}\,\tilde{b}(P' \mid !m \star F)\ \mathcal{R}\ \boldsymbol{\nu}\,\tilde{c}(Q' \mid !m \star E)$$

cannot be made simpler without losing discriminanting power. The counter-examples we give below are in the weak case – the interesting one – and for simplicity, they employ names which can carry either a process or the empty tuple.

First of all, the guard m on F and E is necessary: For, otherwise, if R is a process like $!\,d$, which can perform an unbounded number of identical visible actions, then the processes

$$\overline{a}\langle R\rangle.\mathbf{0} \qquad \text{and} \qquad \overline{a}\langle \mathbf{0}\rangle.R$$

would be made equivalent. But they are not context bisimilar; for instance they can be distinguished when interacting with a process like $a(X).\mathbf{0}$, which discharges what it receives at a.

Now, the use of replication. Suppose we eliminate it from the clause (2) of Definition 4.7.1. Then let $R = b.b.\overline{c} \mid \overline{b}$ and consider

$$P \stackrel{def}{=} (\boldsymbol{\nu}\, b)\overline{a}\langle R\rangle.\mathbf{0} \qquad\qquad Q \stackrel{def}{=} \overline{a}\langle \mathbf{0}\rangle.\mathbf{0}$$

Again, P and Q would become equivalent since $\boldsymbol{\nu}\, b\,(m.R)$ and $m.\mathbf{0}$ are indistinguishable. However P and Q can be differentiated when interacting with a process like $a(X).(X \mid X)$ which makes two copies of the input run in parallel so that the action c in R can be observed. (In fact, only in the *linear* HOπ, where in each expression, a variable may occur free only once, the replication in question can be avoided.)

We now prove formally that $\cong_{\mathrm{Ct}}$ and $\cong_{\mathrm{Nr}}$ coincide. The inclusion $\cong_{\mathrm{Ct}} \subseteq \cong_{\mathrm{Nr}}$ is obvious, since the requirements in the definition of $\cong_{\mathrm{Nr}}$ are a subset of those in the definition of $\cong_{\mathrm{Ct}}$. To prove the opposite inclusion, we use the Lemma 4.7.3 below, which relates weak transitions of P and $\mathcal{T}[\![P]\!]$. Its proof is obtained from the operational correspondence on strong transitions in Lemma 4.5.10 and the following auxiliary lemma:

Lemma 4.7.2 *For any G, F with $m \notin fn(G,F)$, it holds that*

$$\mathcal{T}[\![\boldsymbol{\nu}\, \tilde{b}\, G\langle F\rangle]\!] \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\, \tilde{b}\,\big(G\langle Tr_m\rangle\ \{m := F\}\big)]\!]$$

PROOF: Using Theorem 4.4.7 (factorisation theorem) and Theorem 4.5.8 (correctness of $\mathcal{T}$), we derive that the two processes in the assertion of the lemma are in the relation $\cong_{\mathrm{Ct}}$. Since $\cong_{\mathrm{Ct}}$ and $\cong_{\mathrm{Tr}}$ coincide on triggered agents, the two processes are also in the relation $\cong_{\mathrm{Tr}}$. □

Lemma 4.7.3 *Let $m \notin fn(P, \mathcal{T}[\![P]\!])$:*

1. *(a) If $P \overset{(\nu\, \tilde{b})\overline{a}\langle F\rangle}{\Longrightarrow} P'$, then P'' exists, s.t.*
   $\mathcal{T}[\![P]\!] \overset{(\nu\, m)\overline{a}\langle Tr_m\rangle}{\Longrightarrow} P'' \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\, \tilde{b}\,(P' \mid\, !m \star F)]\!]$

   (b) Suppose μ is a basic input or a silent action and $bn(\mu) \cap fn(P,Q) = \emptyset$;
   if $P \overset{\mu}{\Longrightarrow} P'$, then P'' exists, s.t. $\mathcal{T}[\![P]\!] \overset{\mu}{\Longrightarrow} P'' \cong_{\mathrm{Tr}} \mathcal{T}[\![P']\!]$.

2. *The converse of (1), i.e.*

   (a) If $\mathcal{T}[\![P]\!] \overset{(\nu\, m)\overline{a}\langle Tr_m\rangle}{\Longrightarrow} P''$, then $P', F, \tilde{b}$ exist, s.t.
   $P \overset{(\nu\, \tilde{b})\overline{a}\langle F\rangle}{\Longrightarrow} P'$ and $P'' \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\, \tilde{b}\,(P' \mid\, !m \star F)]\!]$,

(b) Suppose μ is a basic input or a silent action and $bn(\mu) \cap fn(P,Q) = \emptyset$; if $\mathcal{T}[\![P]\!] \stackrel{\mu}{\Longrightarrow} P''$ then P' exists s.t. $P \stackrel{\mu}{\Longrightarrow} P'$ and $P'' \cong_{\mathrm{Tr}} \mathcal{T}[\![P']\!]$.

PROOF: We only look at (1.a), since the other cases are similar, and we proceed by induction on the length of the transition for $P \stackrel{(\nu \widetilde{b})\overline{a}\langle F\rangle}{\Longrightarrow} P'$. If the length is one, then the result follows from Lemma 4.5.10(1.a). If the length is greater than one, then, for some P_1, either

$$P \stackrel{\tau}{\longrightarrow} P_1 \stackrel{(\nu \widetilde{b})\overline{a}\langle F\rangle}{\Longrightarrow} P' \quad \text{or} \quad P \stackrel{(\nu \widetilde{b})\overline{a}\langle F\rangle}{\longrightarrow} P_1 \stackrel{\tau}{\Longrightarrow} P'$$

holds. Again, since the two cases can be handled in a similar way, we only consider the first. By the induction hypothesis, we have

$$\mathcal{T}[\![P_1]\!] \stackrel{(\nu\, m)\overline{a}\langle T r_m\rangle}{\Longrightarrow} P''' \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\, \widetilde{b}\,(P' \mid !m \star F)]\!] \tag{4.10}$$

On the other hand, we have for some P_1',

$$\mathcal{T}[\![P]\!] \stackrel{\tau}{\longrightarrow} P_1' \cong_{\mathrm{Tr}} \mathcal{T}[\![P_1]\!] \tag{4.11}$$

as can be inferred from Lemma 4.5.10(1.d) or from Lemma 4.5.10(1.c) together with Lemma 4.7.2, depending on whether $\mathcal{T}[\![P]\!] \stackrel{\tau}{\longrightarrow} P''$ is a first-order or a higher-order reduction. Now, from (4.11) and (4.10),

$$P_1' \stackrel{(\nu\, m)\overline{a}\langle T r_m\rangle}{\Longrightarrow} P'' \cong_{\mathrm{Tr}} P'''$$

Summarising, we have $\mathcal{T}[\![P]\!] \stackrel{(\nu\, m)\overline{a}\langle T r_m\rangle}{\Longrightarrow} P'' \cong_{\mathrm{Tr}} P''' \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\, b\,(P' \mid !m \star F)]\!]$, as required.

Theorem 4.7.4 *On HOπ processes, $\cong_{\mathrm{Nr}}$ and $\cong_{\mathrm{Ct}}$ coincide.*

PROOF: We have only to prove the inclusion $\cong_{\mathrm{Nr}} \subseteq \cong_{\mathrm{Ct}}$. For this, we show that

$$\mathcal{R} = \{(\mathcal{T}[\![P]\!], \mathcal{T}[\![Q]\!]) \,:\, P \cong_{\mathrm{Nr}} Q\}$$

is a $\cong_{\mathrm{Tr}}$-bisimulation up-to $\cong_{\mathrm{Tr}}$. This is enough, because on triggered processes $\cong_{\mathrm{Tr}}$ and $\cong_{\mathrm{Ct}}$ coincide (Corollary 4.6.6), and $\mathcal{T}$ respects $\cong_{\mathrm{Ct}}$ (Theorem 4.5.8(3)): Hence from $\mathcal{T}[\![P]\!] \cong_{\mathrm{Tr}} \mathcal{T}[\![Q]\!]$, we can infer $P \cong_{\mathrm{Ct}} Q$.

Suppose $\mathcal{T}[\![P]\!] \stackrel{(\nu\, m)\overline{a}\langle T r m\rangle}{\Longrightarrow} P''$, for $m \notin fn(P,Q)$. By Lemma 4.7.3(2.a), P', F, and $\tilde{b}$ exist s.t.

$$P \stackrel{(\nu\tilde{b})\overline{a}\langle F\rangle}{\Longrightarrow} P' \text{ and } P'' \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\,\tilde{b}(P' \mid !m \star F)]\!]$$

Since $P \cong_{\mathrm{Nr}} Q$, there exist Q', E, and $\tilde{c}$, s.t. $Q \stackrel{(\nu\tilde{c})\overline{a}\langle E\rangle}{\Longrightarrow} Q'$ and

$$\boldsymbol{\nu}\,\tilde{b}(P' \mid !m \star F) \cong_{\mathrm{Nr}} \boldsymbol{\nu}\,\tilde{c}(Q' \mid !m \star E)$$

Further, by Lemma 4.7.3(1.a),

$$\mathcal{T}[\![Q]\!] \stackrel{(\nu\, m)\overline{a}\langle T r m\rangle}{\Longrightarrow} Q'' \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\,\tilde{c}(Q' \mid !m \star E)]\!]$$

Summarising, we have

$$\mathcal{T}[\![P]\!] \stackrel{(\nu\, m)\overline{a}\langle T r m\rangle}{\Longrightarrow} \cong_{\mathrm{Tr}} \mathcal{T}[\![\boldsymbol{\nu}\,\tilde{b}(P' \mid !m \star F)]\!]\ \mathcal{R}\ \mathcal{T}[\![\boldsymbol{\nu}\,\tilde{c}(Q' \mid !m \star E)]\!] \cong_{\mathrm{Tr}} \stackrel{(\nu\, m)\overline{a}\langle T r m\rangle}{\Longleftarrow} \mathcal{T}[\![Q]\!]$$

which is enough, because $\mathcal{R}$ is a $\cong_{\mathrm{Tr}}$-bisimulation up-to $\cong_{\mathrm{Tr}}$. □

Remark 4.7.5 Now that we have proved that $\cong_{\mathrm{Ct}}$ and $\cong_{\mathrm{Nr}}$ coincide, one might wonder why we did not introduce $\cong_{\mathrm{Nr}}$ directly. The reason is that we would have needed $\cong_{\mathrm{Ct}}$ to prove the congruence of $\cong_{\mathrm{Nr}}$ over parallel composition (the skeptical reader might want to try the proof himself/herself). Moreover, it is easier to convince ourselves of the naturalness of context bisimulation; we then can accept normal bisimulation as a simpler characterisation of the former. □

4.7.2 Characterisation in terms of barbed bisimulation

We have already seen the correspondence between $\cong_{\mathrm{Ct}}, \cong_{\mathrm{Nr}}$, and $\cong_{\mathrm{Tr}}$. It remains to relate these with $\approx$ (weak barbed equivalence). For this, it is advantageous to work with $\cong_{\mathrm{Nr}}$.

Theorem 4.7.6 *On HOπ processes, $\approx$ and $\cong_{\mathrm{Nr}}$ coincide.* □

PROOF: The proof is obtained with a few adjustments from the proof of Theorem 3.3.3(2), where it is shown that in the π-calculus, $\approx$ coincides weak early bisimulation. See Appendix D for the details.

Corollary 4.7.7 *The relations* $\cong_{\mathrm{Ct}}$, $\cong_{\mathrm{Nr}}$, *and* $\approx$ *coincide on HOπ processes.*

PROOF: Follows from Theorems 4.7.4 and 4.7.6. □

In Chapter 3 we defined barbed bisimulation and its congruences only for closed processes. The most reasonable way to extend it to open agents is to do as for context bisimulation, requiring the instantiation of a variable with each agent of the right sort. Corollary 4.7.7 can then be generalised to open agents. To show that $\cong_{\mathrm{Ct}}$ and $\cong_{\mathrm{Nr}}$ coincide on open agents, we need also the factorisation theorem (since $\cong_{\mathrm{Nr}}$ is defined on open agents by requiring only instantiation of variables with fresh triggers).

Corollary 4.7.8 *The relations* $\cong_{\mathrm{Ct}}$, $\cong_{\mathrm{Nr}}$, $\approx$ *coincide on* $HO\pi^{\circ}$ *agents.* □

The result can then be extended to the congruences:

Corollary 4.7.9 *The relations* $\cong^{\mathsf{c}}_{\mathrm{Ct}}$, $\cong^{\mathsf{c}}_{\mathrm{Nr}}$, $\approx^{\mathsf{c}}$ *coincide on* $HO\pi^{\circ}$ *agents.* □

Remark 4.7.10 We think that Corollaries 4.6.6, 4.7.8 and 4.7.9 hold in the strong case too. We have not studied it though, since we are mainly interested in weak equivalences (and mapping $\mathcal{T}$ only respects these). We leave the problem for future investigations. □

4.7.3 Some comments on the bisimulations considered

We hope not to have confused the reader by considering too many equivalences on HOπ agents, namely context bisimulation, normal bisimulation and barbed equivalence, plus the corresponding congruences. But we feel that this variety was somehow necessary. Having shown the correspondence among them (Corollaries 4.7.8 and 4.7.9) we can now see how each of them has its own rôle in strengthening the importance of the resulting behavioural equivalence:

- the characterisation in terms of normal bisimulation specifies some minimum requirement, or test, for higher-order actions and as such provides us with an efficient mathematical tool to verify agent equivalences;

- the characterisation in terms of context bisimulation gives us a measure of the power of the above tests;
- finally, the characterisation in terms of barbed equivalence demonstrates the naturalness of the equivalence.

In the following, whenever possible we shall try to use barbed equivalence and congruence. But the other relations will often be invoked in the proofs.

EXTENSION TO AGENTS WITH ARBITRARY ARITIES. We have presented the definitions of the bisimulations on agents which may only use unary tuples in communications and abstractions. The extension of the definitions to agents with arbitrary arities is fairly intuitive: It is just a question of combining together the clauses for first-order and higher-order actions. As an example, we look at the output clause of normal bisimulation. Take the actions:

$$P \xrightarrow{(\nu\,\tilde{b})\overline{a}\langle F,d\rangle} P' \qquad \text{and} \qquad Q \xrightarrow{(\nu\,\tilde{c})\overline{a}\langle G,d'\rangle} Q'$$

with $(\tilde{b} \cup \tilde{c}) \cap fn(P,Q) = \emptyset$. When do they "match" each other? First of all, the usual condition on first-order actions demands that the names emitted are the same, thus $d = d'$ (up-to α-conversion). Moreover, if d represents a bound name, it becomes visible at completion of the actions. Therefore, if $\tilde{b}' = \tilde{b} - d$, $\tilde{c}' = \tilde{c} - d$, and m is a fresh name, the other condition to check is

$$(\nu\,\tilde{b}')(!\,m \star F \mid P') \simeq_{\mathrm{Nr}} (\nu\,\tilde{c}')(!\,m \star G \mid Q')$$

Chapter 5

The Representability of HOπ in π-calculus

In this chapter we show how the higher-order constructs of HOπ can be effectively represented in π-calculus. This is obtained via a compilation $\mathcal{C}$ which intuitively, replaces the communication of an agent with the communication of the *access* to that agent. Formally, $\mathcal{C}$ is derived in two steps. The first one is the mapping $\mathcal{T}$ presented in the previous chapter which transforms every agent into a triggered agent; the second one is a mapping $\mathcal{F}$ which takes from triggered agents down to first-order agents (and first-order sortings).

We continue to work in the reduced HOπ presented in Section 4.1, where: One only value – a name or an abstraction — can be exchanged in a communication; only unary abstractions are allowed; constants are replaced by replication and all sums are finite. We remind that only the latter restriction is meaningful — it intentionally limits the applicability of our results to those HOπ agents defined from a finite number of constants and with finite sums. The arity restrictions are adopted merely to make the presentation of the results clearer. We maintain the notation introduced in Section 4.1.

We study $\mathcal{F}$ and $\mathcal{C}$ in Sections 5.1 and 5.2, respectively. In Section 5.3 we analyse Bent Thomsen's work, which first attempted the translation of a higher-order

calculus, namely Plain CHOCS, into the π-calculus. Thomsen's approach employs higher-order bisimulation; we take advantage of the opportunity to compare it with barbed bisimulation.

5.1 The encoding of triggered agents

In this section we present the encoding $\mathcal{F}$ which transforms triggered HOπ agents and their sorting into π-calculus agents and first-order sortings.

In triggered agents every higher-order communication is the communication of an agent, a trigger, which is univocally determined by the unique name free in it. Consequently, it is possible to simulate in a simple way the behaviour of triggered agents using only name-passing: Instead of the trigger Tr_m, one transmits the name m. We call m a *name-trigger*, since has the same functionality as a trigger. This transformation modifies the object sort of the name at which the communication has taken place and, hence, the transformation has to be defined on *sorts* too. Moreover, since its domain and codomain are different, some care is necessary when presenting results on its faithfulness. For this, it is elegant to exploit barbed bisimulation, whose versatility allows a uniform definition in the two calculi.

Definition of the encoding

NOTATION. In the definition of triggered agents, the choice of the subsorting SOb from which the free names of the triggers are drawn is important (see preliminaries of Section 4.5). For an object sort S, we denote by SOb_S the unique subject sort s s.t. $s \mapsto S \in SOb$. As in Chapter 4, whenever possible we leave SOb implicit; for instance, we simply write $A \in THO\pi$ to say that A is a triggered agent, without specifying SOb. We shall always use m, m' as names whose sort is in SOb.

On the sorting, the encoding $\mathcal{F}$ has to replace all uses of a higher-order sort S with SOb_S. Therefore we get

$$\mathcal{F}[\![Ob]\!] = \{s \mapsto (s') \in Ob\} \bigcup \{s \mapsto (SOb_S) \ : \ s \mapsto (S) \in Ob\}.$$

$\mathcal{F}[\![(a)P]\!]$	$\stackrel{def}{=}$	$(a)\mathcal{F}[\![P]\!]$	$\mathcal{F}[\![(X)P]\!]$	$\stackrel{def}{=}$	$(x)\mathcal{F}[\![P]\!]$	$\mathcal{F}[\![[a=b]P]\!]$	$\stackrel{def}{=}$	$[a=b]\mathcal{F}[\![P]\!]$
$\mathcal{F}[\![\boldsymbol{\nu}\, a\, P]\!]$	$\stackrel{def}{=}$	$\boldsymbol{\nu}\, a\, \mathcal{F}[\![P]\!]$	$\mathcal{F}[\![!\, P]\!]$	$\stackrel{def}{=}$	$!\, \mathcal{F}[\![P]\!]$	$\mathcal{F}[\![\sum_{i\in I} P_i]\!]$	$\stackrel{def}{=}$	$\sum_{i\in I} \mathcal{F}[\![P_i]\!]$
$\mathcal{F}[\![X\langle Tr_m\rangle]\!]$	$\stackrel{def}{=}$	$\overline{x}\langle m\rangle.\mathbf{0}$	$\mathcal{F}[\![X\langle b\rangle]\!]$	$\stackrel{def}{=}$	$\overline{x}\langle b\rangle.\mathbf{0}$	$\mathcal{F}[\![P \mid Q]\!]$	$\stackrel{def}{=}$	$\mathcal{F}[\![P]\!] \mid \mathcal{F}[\![Q]\!]$
$\mathcal{F}[\![\alpha.P]\!]$	$\stackrel{def}{=}$	$\mathcal{F}[\![\alpha]\!].\mathcal{F}[\![P]\!]$	where $\mathcal{F}[\![\alpha]\!]$ is the abbreviation:					

$$\mathcal{F}[\![\alpha]\!] \stackrel{def}{=} \begin{cases} \overline{a}\langle m\rangle & \text{if } \alpha = \overline{a}\langle Tr_m\rangle \\ a(x) & \text{if } \alpha = a(X) \\ \alpha & \text{otherwise} \end{cases}$$

Table 5–1: Transformation $\mathcal{F}$

The definition of $\mathcal{F}$ on agents is presented in Table 5–1. In the table, we have assumes that if $X : S$ is a variable, its lower case letter x is an unused name whose sort is SOb_S. We thus ensure that distinct variables are mapped onto distinct and fresh names by mapping each variable X to its corresponding lower case letter x. The encoding $\mathcal{F}$ acts inductively on the structure of an agent. The interesting cases are those for higher-order application and prefixing. $\mathcal{F}$ acts as a homomorphism elsewhere. Since in triggered agents every non-binding occurrence of a variable is followed by its argument, we do not need a rule for variables.

Let us add a comment on the rule for higher-order application $X\langle Tr_m\rangle$. When X is instantiated to a trigger, say $Tr_{m'}$, it becomes $Tr_{m'}\langle Tr_m\rangle = \overline{m'}\langle Tr_m\rangle.\mathbf{0}$. Translating this, we expect to send m instead of Tr_m, that is, to have $\overline{m'}\langle m\rangle.\mathbf{0}$ instead of $\overline{m'}\langle Tr_m\rangle.\mathbf{0}$. With this example in mind, our rule for higher-order application should become clear. Note that from the definitions of $\mathcal{F}$ and of the abbreviation $\{m := F\}$, we have

$$\mathcal{F}[\![P\,\{m := F\}]\!] = \mathcal{F}[\![P]\!]\,\{m := \mathcal{F}[\![F]\!]\}$$

We show how $\mathcal{F}$ acts with two examples. They continue Examples 4.5.5 and 4.5.6; for readability we have abbreviated $\mathcal{F}[\![\mathcal{T}[\![P]\!]]\!]$ as $\mathcal{F}\mathcal{T}[\![P]\!]$.

Example 5.1.1 Let $\mathcal{T}[\![P_i]\!]$, $i = 1, 2$ as in Example 4.5.5, i.e.

$$\begin{aligned}\mathcal{T}[\![P_1]\!] &= X\langle Tr_m\rangle\ \{m := F\} \mid (\overline{b}\langle Tr_m\rangle.\mathbf{0})\ \{m := F\}, \text{ for } F \stackrel{def}{=} (b)\mathbf{0}\\ \mathcal{T}[\![P_2]\!] &= a(X).\Big(Z\langle Tr_m\rangle\ \Big\{m := (Y)\Big(X\langle Tr_{m'}\rangle\ \{m' := (b)Y\langle b\rangle\}\Big)\Big\}\Big)\end{aligned}$$

Then we have:

$$\begin{aligned}\mathcal{FT}[\![P_1]\!] &= (\overline{x}\langle m\rangle.\mathbf{0})\ \{m := F\} \mid (\overline{b}\langle m\rangle.\mathbf{0})\ \{m := F\}\\ \mathcal{FT}[\![P_2]\!] &= a(x).\Big((\overline{z}\langle m\rangle.\mathbf{0})\ \Big\{m := (y)(\overline{x}\langle m'\rangle.\mathbf{0})\ \{m' := (b)\overline{y}\langle b\rangle.\mathbf{0}\}\Big\}\Big)\end{aligned}$$

□

Example 5.1.2 Let us examine how the translation works on reductions, using the processes P and $\mathcal{T}[\![P]\!]$ of Example 4.5.6, namely $P = \overline{a}\langle F\rangle.Q | a(Y).(Y\langle b\rangle | Y\langle c\rangle)$, and $\mathcal{T}[\![P]\!] = (\overline{a}\langle Tr_m\rangle.\mathcal{T}[\![Q]\!])\ \{m := \mathcal{T}[\![F]\!]\} \mid a(Y).(Y\langle b\rangle \mid Y\langle c\rangle)$. We saw that P reduces to $Q \mid F\langle b\rangle \mid F\langle c\rangle$ in one step, and that $\mathcal{T}[\![P]\!]$ needs three steps to to simulate this and reduce to a process which is $\simeq^{c}_{Ct}$ with $\mathcal{T}[\![Q \mid F\langle b\rangle \mid F\langle c\rangle]\!]$. Let us show how closely $\mathcal{FT}[\![P]\!]$ follows the behaviour of $\mathcal{T}[\![P]\!]$.

$$\begin{aligned}\mathcal{FT}[\![P]\!] \quad &= \quad (\overline{a}\langle m\rangle.\mathcal{FT}[\![Q]\!])\ \{m := \mathcal{FT}[\![F]\!]\} \mid a(y).(\overline{y}\langle b\rangle.\mathbf{0} \mid \overline{y}\langle c\rangle.\mathbf{0})\\ &\equiv \quad \Big(\overline{a}\langle m\rangle.\mathcal{FT}[\![Q]\!] \mid a(y).(\overline{y}\langle b\rangle.\mathbf{0} \mid \overline{y}\langle c\rangle.\mathbf{0})\Big)\ \{m := \mathcal{FT}[\![F]\!]\}\\ &\longrightarrow \quad \Big(\mathcal{FT}[\![Q]\!] \mid \overline{m}\langle b\rangle.\mathbf{0} \mid \overline{m}\langle c\rangle.\mathbf{0}\Big)\ \{m := \mathcal{FT}[\![F]\!]\}\\ &\longrightarrow\equiv \quad \Big(\mathcal{FT}[\![Q]\!] \mid \mathcal{FT}[\![F]\!]\langle b\rangle \mid \mathcal{FT}[\![F]\!]\langle c\rangle\Big)\ \{m := \mathcal{FT}[\![F]\!]\}\\ &\sim^{c} \quad \mathcal{FT}[\![Q]\!] \mid \mathcal{FT}[\![F]\!]\langle b\rangle \mid \mathcal{FT}[\![F]\!]\langle c\rangle = \mathcal{FT}[\![Q \mid F\langle b\rangle \mid F\langle c\rangle]\!]\end{aligned}$$

where the last $\sim^{c}$ holds because m is not free in the body. □

Syntactic correctness

Before tackling the question of the semantic correctness of $\mathcal{F}$, we have to check that its definition is syntactically meaningful by ensuring that it returns first-order agents and that there is agreement between the definition of $\mathcal{F}$ on sorts and on agents. The former is immediate. The latter holds too because all new names which are introduced (in the rule for application) or whose object part is modified (in the rule for prefixing), respect the sorting $\mathcal{F}[\![Ob]\!]$. Note that for the well-definiteness of the translation of higher-order outputs and applications, we exploit the property of triggered agents that if two triggers Tr_m and $Tr_{m'}$ have

the same sort S, then the names m and m' belong to the same sort SOb_S

Theorem 5.1.3 *Suppose that $A \in THO\pi$; we have:*

1. *$\mathcal{F}[\![A]\!]$ is first-order agent;*

2. *$\mathcal{F}[\![A]\!] :: \mathcal{F}[\![Ob]\!]$.*

Operational correspondence

The following lemma is analogous to Lemma 4.5.9 for $\mathcal{T}$.

Lemma 5.1.4 *Let $A \in THO\pi^\circ$. Then $\mathcal{F}[\![A\{Tr_m/X\}]\!] = \mathcal{F}[\![A]\!]\{m/x\}$.*

PROOF: By induction on the structure of A. We consider only one case; the remaining ones are easier and can be treated in a similar way. Suppose that $A = X\langle Tr_{m'}\rangle\ \{m' := F\}$. Then Tr_m must be of the form $(Y)\overline{m}\langle Y\rangle.\mathbf{0}$ and, for $F' \stackrel{def}{=} F\{Tr_m/X\}$, we have

$$A\{Tr_m/X\} = Tr_m\langle Tr_{m'}\rangle\ \{m' := F'\} = \left(\overline{m}\langle Tr_{m'}\rangle.\mathbf{0}\right)\ \{m' := F'\}.$$

From here we get

$$\mathcal{F}[\![A\{Tr_m/X\}]\!] = \left(\overline{m}\langle m'\rangle.\mathbf{0}\right)\ \{m := \mathcal{F}[\![F']\!]\}.$$

This is the same expression denoted by $\mathcal{F}[\![A]\!]\{m/x\}$; in fact we have:

$$\begin{aligned}
\mathcal{F}[\![A]\!]\{m/x\} &= \\
\mathcal{F}[\![X\langle Tr_{m'}\rangle\ \{m' := F\}]\!]\{m/x\} &= \\
\left(\overline{m}\langle m'\rangle.\mathbf{0}\right)\ \{m' := \mathcal{F}[\![F]\!]\{m/x\}\} &= \\
\left(\overline{m}\langle m'\rangle.\mathbf{0}\right)\ \{m := \mathcal{F}[\![F']\!]\}
\end{aligned}$$

where the last equality holds because $\mathcal{F}[\![F']\!] = \mathcal{F}[\![F\{Tr_m/X\}]\!] =$ (by induction) $\mathcal{F}[\![F]\!]\{m/x\}$. □

In Table 5–1, we extended $\mathcal{F}$ to prefixes; now we extend it also to actions to better evidence the operational correspondence yielded by $\mathcal{F}$. The definition of

$\mathcal{F}[\![\mu]\!]$ is obvious from the one of $\mathcal{F}[\![\alpha]\!]$: Since actions use free inputs rather than bound inputs as prefixes, we have just to replace in the definition of $\mathcal{F}[\![\alpha]\!]$ the clause for higher-order inputs with the clause $\mathcal{F}[\![a\langle Tr_m\rangle]\!] = a\langle m\rangle$.

Lemma 5.1.5 (operational correspondence for $\mathcal{F}$ on strong transitions) *Let $P \in THO\pi$ and μ a basic action. Then*

1. *$P \xrightarrow{\mu} P'$ implies $\mathcal{F}[\![P]\!] \xrightarrow{\mathcal{F}[\![\mu]\!]} \mathcal{F}[\![P']\!]$.*
2. *the converse, i.e. $\mathcal{F}[\![P]\!] \xrightarrow{\mu'} P''$ implies there exists P', μ s.t. $P \xrightarrow{\mu} P'$ and $P'' = \mathcal{F}[\![P']\!]$, $\mu' = \mathcal{F}[\![\mu]\!]$.*

PROOF: Another easy induction on the structure of P, both for (1) and for (2). Again, we consider the most significant cases.

- $P = a(X).Q$.

  Then $\mathcal{F}[\![P]\!] = a(x).\mathcal{F}[\![Q]\!]$. It holds that $P \xrightarrow{a\langle Tr_m\rangle} Q\{Tr_m/X\}$ and $\mathcal{F}[\![P]\!] \xrightarrow{a\langle m\rangle} \mathcal{F}[\![Q]\!]\{m/x\}$; we have $a\langle m\rangle = \mathcal{F}[\![a\langle Tr_m\rangle]\!]$ and by Lemma 5.1.4, $\mathcal{F}[\![Q\{Tr_m/X\}]\!] = \mathcal{F}[\![Q]\!]\{m/x\}$.

- $P = P_1 \mid P_2$.

  Then $\mathcal{F}[\![P]\!] = \mathcal{F}[\![P_1]\!] \mid \mathcal{F}[\![P_2]\!]$. We consider only (1) in the case that the rule used to infer the action is COM. We have

$$\frac{P_1 \xrightarrow{(\nu\, m)\overline{a}\langle Tr_m\rangle} P_1' \quad P_2 \xrightarrow{a\langle Tr_m\rangle} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} \boldsymbol{\nu}\, m\,(P_1' \mid P_2')}$$

  Using induction twice,

$$\frac{\mathcal{F}[\![P_1]\!] \xrightarrow{(\nu\, m)\overline{a}\langle m\rangle} \mathcal{F}[\![P_1']\!] \quad \mathcal{F}[\![P_2]\!] \xrightarrow{a\langle m\rangle} \mathcal{F}[\![P_2']\!]}{\mathcal{F}[\![P_1]\!] \mid \mathcal{F}[\![P_2]\!] \xrightarrow{\tau} \boldsymbol{\nu}\, m\,(\mathcal{F}[\![P_1']\!] \mid \mathcal{F}[\![P_2']\!]) = \mathcal{F}[\![\boldsymbol{\nu}\, m\,(P_1' \mid P_2')]\!]}$$

The generalisation of Lemma 5.1.5 to the weak case is straightforward.

Corollary 5.1.6 (operational correspondence for $\mathcal{F}$ on weak transitions) *Let $P \in THO\pi$ and μ a basic action. Then*

1. *$P \stackrel{\mu}{\Longrightarrow} P'$ implies $\mathcal{F}[\![P]\!] \stackrel{\mathcal{F}[\![\mu]\!]}{\Longrightarrow} \mathcal{F}[\![P']\!]$.*

2. *the converse, i.e. $\mathcal{F}[\![P]\!] \stackrel{\mu'}{\Longrightarrow} P''$ implies there exists P', μ s.t. $P \stackrel{\mu}{\Longrightarrow} P'$ and $P'' = \mathcal{F}[\![P']\!]$, $\mu' = \mathcal{F}[\![\mu]\!]$.* □

Correctness w.r.t. weak barbed equivalence and congruence

In the proof of the results below we exploit the characterisations of barbed equivalence in terms of weak context bisimulation ($\approx_{\mathrm{Ct}}$) and weak triggered bisimulation ($\approx_{\mathrm{Tr}}$) in HOπ (Corollary 4.7.8), and of weak early bisimulation ($\approx_{\mathrm{e}}$) in π-calculus (Theorem 3.3.3); moreover we only check the actions occurring at names whose object sort has been modified by $\mathcal{F}$. The proof for the remaining actions is a straightforward application of Lemma 5.1.5 and Corollary 5.1.6.

Lemma 5.1.7 *Suppose P and Q are $THO\pi$ processes. Then $P \approx Q$ implies $\mathcal{F}[\![P]\!] \approx \mathcal{F}[\![Q]\!]$.*

PROOF: We prove that

$$\mathcal{R} = \{(\mathcal{F}[\![P]\!], \mathcal{F}[\![Q]\!]) \ : \ P, Q \in THO\pi \text{ and } P \approx_{\mathrm{Tr}} Q\}$$

is a $\approx_{\mathrm{e}}$-bisimulation. Suppose $\mathcal{F}[\![P]\!] \stackrel{a\langle m\rangle}{\longrightarrow} P''$. By Lemma 5.1.5,

$$P \stackrel{a\langle Tr_m\rangle}{\longrightarrow} P' \quad \text{with} \quad P'' = \mathcal{F}[\![P']\!].$$

The definition of $\approx_{\mathrm{Tr}}$ guarantees that there exists Q' s.t. $Q \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} Q' \approx_{\mathrm{Tr}} P'$ only for $m \notin fn(P, Q)$. Since here in general m may occur free in P or Q, we have to exploit the equality between $\approx_{\mathrm{Ct}}$ and $\approx_{\mathrm{Tr}}$ on triggered agents. From $P \approx_{\mathrm{Ct}} Q$ we get

$$Q \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} Q' \approx_{\mathrm{Ct}} P'$$

Since $Q' \in THO\pi$ by Lemma 4.6.1, also $Q' \cong_{\mathrm{Tr}} P'$; moreover by Corollary 5.1.6,

$$\mathcal{F}[\![Q]\!] \stackrel{a\langle m\rangle}{\Longrightarrow} \mathcal{F}[\![Q']\!]$$

yielding $P'' \; \mathcal{R} \; \mathcal{F}[\![Q']\!]$.

Let us consider now outputs. By Lemma 5.1.5 every output performed by $\mathcal{F}[\![P]\!]$ is the image, under $\mathcal{F}$, of an action performed by P. By Lemma 4.6.1, every higher-order output performed by P is of the form $(\boldsymbol{\nu}\, m)\overline{a}\langle Tr_m\rangle$. Hence the corresponding output of $\mathcal{F}[\![P]\!]$ are of the form $\mathcal{F}[\![P]\!] \stackrel{(\nu\, m)\overline{a}\langle m\rangle}{\longrightarrow} P''$. Now, proceeding as in the input case above — but this time we do not need to employ $\cong_{\mathrm{Ct}}$ — we find P' with $P'' = \mathcal{F}[\![P']\!]$, and $Q' \cong_{\mathrm{Tr}} P'$ s.t. $\mathcal{F}[\![Q]\!] \stackrel{(\nu\, m)\overline{a}\langle m\rangle}{\longrightarrow} \mathcal{F}[\![Q']\!]$. □

Lemma 5.1.8 *For processes $P, Q \in THO\pi$, $\mathcal{F}[\![P]\!] \approx \mathcal{F}[\![Q]\!]$ implies $P \approx Q$.*

PROOF: We prove that

$$\mathcal{R} = \{(P, Q) \; : \; \mathcal{F}[\![P]\!] \approx_{\mathrm{e}} \mathcal{F}[\![Q]\!]\}$$

is a $\cong_{\mathrm{Tr}}$-bisimulation. Suppose $P \stackrel{a\langle Tr_m\rangle}{\longrightarrow} P'$, and $m \notin fn(P, Q)$. Then, by Lemma 5.1.5, $\mathcal{F}[\![P]\!] \stackrel{a\langle m\rangle}{\longrightarrow} \mathcal{F}[\![P']\!]$. From $\mathcal{F}[\![P]\!] \approx_{\mathrm{e}} \mathcal{F}[\![Q]\!]$, we get $\mathcal{F}[\![Q]\!] \stackrel{a\langle m\rangle}{\Longrightarrow} Q''$ and by Corollary 5.1.6, $Q \stackrel{a\langle Tr_m\rangle}{\Longrightarrow} Q'$ with $\mathcal{F}[\![Q']\!] = Q''$. Therefore, $Q' \; \mathcal{R} \; P'$.

Now the case of an output action. Using Lemma 4.6.1 the transition for P is of the form $P \stackrel{(\nu\, m)\overline{a}\langle Tr_m\rangle}{\longrightarrow} P'$ and everything follows similarly to the input case. □

We now show that the faithfulness of $\mathcal{F}$ extends to abstractions too. Then we shall be able to express the final results on *open* agents.

Lemma 5.1.9 *For abstractions $F, E \in THO\pi$, we have*

$$F \approx E \;\; \textit{iff} \;\; \mathcal{F}[\![F]\!] \approx \mathcal{F}[\![E]\!].$$

PROOF: This assertion is easy to prove for first-order abstractions using the two previous lemmas, so we only consider higher-order abstractions. Suppose $F = (X)P$ and $E = (X)Q$ with $F \approx E$. Using the characterisation of $\approx$ in

terms of $\cong_{\mathrm{Ct}}$, we infer that for every m, $P\{Tr_m/X\} \approx Q\{Tr_m/X\}$. Hence by Lemma 5.1.7, $\mathcal{F}[\![P\{Tr_m/X\}]\!] \approx \mathcal{F}[\![Q\{Tr_m/X\}]\!]$, from which, using Lemma 5.1.4, also $\mathcal{F}[\![P]\!]\{m/x\} \approx \mathcal{F}[\![Q]\!]\{m/x\}$. But this means $(x)\mathcal{F}[\![P]\!] \approx (x)\mathcal{F}[\![Q]\!]$ which, by definition of the encoding, is the same as $\mathcal{F}[\![(X)P]\!] \approx \mathcal{F}[\![(X)Q]\!]$.

The implication in the opposite direction can be worked out similarly, but this time using the characterisation of $\approx$ in terms of $\cong_{\mathrm{Tr}}$ instead of $\cong_{\mathrm{Ct}}$. □

Theorem 5.1.10 ($\mathcal{F}$ **respects** $\approx$) *For agents $A_1, A_2 \in THO\pi^{\circ}$, we have*

$$A_1 \approx A_2 \text{ iff } \mathcal{F}[\![A_1]\!] \approx \mathcal{F}[\![A_2]\!]$$

PROOF: Lemmas 5.1.7 and 5.1.8 prove the result for closed processes, and Lemma 5.1.9 for closed abstractions. Therefore, it remains to consider the case in which A_1 or A_2 contain free variables. The proof for this case is similar to the proof of Lemma 5.1.9 for higher-order abstractions. (Actually, it is simpler: In the implication from left to right we do not need to employ $\cong_{\mathrm{Ct}}$, since $\cong_{\mathrm{Tr}}$ is enough. The reason is that a variable which is free in A_1, A_2 becomes a free name in $\mathcal{F}[\![A_1]\!]$ and $\mathcal{F}[\![A_2]\!]$ and to check $\mathcal{F}[\![A_1]\!] \approx_{\mathrm{e}} \mathcal{F}[\![A_2]\!]$ we do not have to consider all possible instantiations of that name). □

Now we can extend the faithfulness of $\mathcal{F}$ to the full congruence $\approx^{\mathrm{c}}$.

Theorem 5.1.11 ($\mathcal{F}$ **respects** $\approx^{\mathrm{c}}$) *For $A_1, A_2 \in THO\pi^{\circ}$, we have*

$$A_1 \approx^{\mathrm{c}} A_2 \text{ iff } \mathcal{F}[\![A_1]\!] \approx^{\mathrm{c}} \mathcal{F}[\![A_2]\!]$$

PROOF: In view of the characterisation of $\approx^{\mathrm{c}}$ in terms of $\cong^{\mathrm{c}}_{\mathrm{Tr}}$ in HOπ and of $\approx^{\mathrm{c}}_{\mathrm{e}}$ in π-calculus, two HOπ or π-calculus agents are in the relation $\approx^{\mathrm{c}}$ if they are in the relation $\approx$ for all name substitutions. Therefore, for closed agents the result of this theorem follows from Theorem 5.1.10 since, for any A, the agents A and $\mathcal{F}[\![A]\!]$ have the same sets of free names. This does not hold if A is open, since the free variables of A become free names in $\mathcal{F}[\![A]\!]$. However, the result for open agents can be proved exploiting the result for closed agents. We omit the details, which are simple. □

$$
\mathcal{C}[\![X]\!] \stackrel{def}{=} \begin{cases} \mathcal{C}[\![(Y)X\langle Y\rangle]\!] & \text{if } X \text{ is a higher-order abstraction} \\ \mathcal{C}[\![(a)X\langle a\rangle]\!] & \text{otherwise} \end{cases}
$$

$$
\mathcal{C}[\![\alpha.P]\!] \stackrel{def}{=} \begin{cases} (\overline{a}\langle m\rangle.\mathcal{C}[\![P]\!])\,\{m := \mathcal{C}[\![F]\!]\} & \text{if } \alpha = \overline{a}\langle F\rangle \\ a(x).\mathcal{C}[\![P]\!] & \text{if } \alpha = a(X) \\ \alpha.\mathcal{C}[\![P]\!] & \text{otherwise} \end{cases}
$$

$$
\begin{array}{llll}
\mathcal{C}[\![X\langle F\rangle]\!] \stackrel{def}{=} (\overline{x}\langle m\rangle.\mathbf{0})\,\{m := \mathcal{C}[\![F]\!]\} & & & \\
\mathcal{C}[\![X\langle b\rangle]\!] \stackrel{def}{=} \overline{x}\langle b\rangle.\mathbf{0} & & \mathcal{C}[\![!\,P]\!] \stackrel{def}{=} & !\mathcal{C}[\![P]\!] \\
\mathcal{C}[\![P \mid Q]\!] \stackrel{def}{=} \mathcal{C}[\![P]\!] \mid \mathcal{C}[\![Q]\!] & & \mathcal{C}[\![\sum_{i\in I} P_i]\!] \stackrel{def}{=} & \sum_{i\in I} \mathcal{C}[\![P_i]\!] \\
\mathcal{C}[\![\boldsymbol{\nu}\, a\, P]\!] \stackrel{def}{=} \boldsymbol{\nu}\, a\, \mathcal{C}[\![P]\!] & & \mathcal{C}[\![[a=b]P]\!] \stackrel{def}{=} & [a=b]\mathcal{C}[\![P]\!] \\
\mathcal{C}[\![(X)P]\!] \stackrel{def}{=} (x)\mathcal{C}[\![P]\!] & & \mathcal{C}[\![(a)P]\!] \stackrel{def}{=} & (a)\mathcal{C}[\![P]\!]
\end{array}
$$

Table 5–2: The transformation $\mathcal{C}$

Remark 5.1.12 The reason for proving the correctness of $\mathcal{F}$ w.r.t. weak equivalences is that later we want to combine $\mathcal{F}$ with mapping $\mathcal{T}$ defined in Section 4.5.2, and $\mathcal{T}$ does not respect the strong equivalences. We conjecture however that the faithfulness of $\mathcal{F}$ holds in the strong case too. □

5.2 The compilation of HOπ into π-calculus

By putting together mappings $\mathcal{T}$ and $\mathcal{F}$, we obtain a compilation $\mathcal{C}$ from higher-order agents and sortings to first-order agents and sortings. $\mathcal{C}$ is defined on sorting as $\mathcal{F}$, since $\mathcal{T}$ does not modify a sorting. The direct definition of $\mathcal{C}$ on agents is reported in Table 5–2. In the table, it is supposed that m and x are fresh names, that is, they do not occur in the source agent. We recall that a higher-order abstraction is an abstraction which takes an agent as argument, i.e. its sort is (S), for some S. We also recall that, from the constructions of $\mathcal{T}$ and $\mathcal{F}$, the definition of $\mathcal{C}$ depends upon the selection of the subsorting $\mathcal{SO}b$ of $\mathcal{O}b$,

from which the name-triggers are drawn. From the results proved for $\mathcal{T}$ and $\mathcal{F}$ (Theorems 4.5.8, 5.1.10 5.1.11), we get:

Theorem 5.2.1 (correctness of C) *For every $A_1, A_2 \in HO\pi^\circ$ and sorting Ob, with $A_1, A_2 : Ob$, it holds that*

1. *$C[\![A_1]\!]$ is a π-calculus process, and $C[\![A_1]\!] :: C[\![Ob]\!]$,*
2. *$C[\![A_1]\!] \approx C[\![A_2]\!]$ iff $A_1 \approx A_2$,*
3. *$C[\![A_1]\!] \approx^c C[\![A_2]\!]$ iff $A_1 \approx^c A_2$.* □

We refer to Examples 5.1.1 and 5.1.2 to see how C behaves. More interesting examples will be provided in Chapter 6, when considering the encoding of λ-calculus. Note that the compilation C is obtained combining $\mathcal{T}$ and $\mathcal{F}$ in one *single* step, as opposite to a definition by induction in which at each step the depth of the object sorts appearing in the sorting Ob is decreased by one. The single step allows us to use the compilation C also in the case in which Ob is infinite and there is no limit on the depth of the sorts in Ob. Moreover, an inductive definition would have not much simplified the correctness proofs of C.

A few considerations to emphasise the faithfulness of C are worthwhile. By itself Theorem 5.2.1 does not reveal anything about how closely $C[\![P]\!]$ simulates P; actually, nothing prevents from obtaining the same results with a very bizarre encoding. First of all, let us show the operational correspondence existing between P and $C[\![P]\!]$, as derived by composing Lemmas 4.5.10 and 5.1.5

Lemma 5.2.2 (operational correspondence between P and $C[\![P]\!]$) *Suppose $m \notin fn(P, C[\![P]\!])$:*

1. *(a) If $P \xrightarrow{(\nu \tilde{b})\overline{a}\langle F\rangle} P'$, then $C[\![P]\!] \xrightarrow{(\nu\, m)\overline{a}\langle m\rangle} \equiv C[\![\boldsymbol{\nu}\, \tilde{b}\,(P' \mid !m \star F)]\!]$;*

   (b) if $P \xrightarrow{a\langle Tr_m\rangle} P'$, then $C[\![P]\!] \xrightarrow{a\langle m\rangle} C[\![P']\!]$,

   (c) if $P \xrightarrow{\tau} P'$, then either

- $C[\![P]\!] \xrightarrow{\tau} C[\![P']\!]$ *or*
- $\widetilde{b}, G, F$ *exist s.t.*
  $P' \equiv \boldsymbol{\nu}\,\widetilde{b}\,G\langle F\rangle$, *and* $C[\![P]\!] \xrightarrow{\tau} \equiv C[\![\boldsymbol{\nu}\,\widetilde{b}\,(G\langle Tr_m\rangle\ \{m := F\})]\!]$;

(d) if $P \xrightarrow{\mu} P'$, *and* μ *is a first-order input or output, then* $C[\![P]\!] \xrightarrow{\mu} C[\![P']\!]$;

2. the converse of (1), i.e:

(a) If $C[\![P]\!] \xrightarrow{(\nu\, m)\overline{a}\langle m\rangle} P''$, *then* $\widetilde{b}$, F, P' *exist s.t.*
$P \xrightarrow{(\nu\widetilde{b})\overline{a}\langle F\rangle} P'$, *and* $P'' \equiv C[\![\boldsymbol{\nu}\,\widetilde{b}\,(P' \mid !m \star F)]\!]$;

(b) if $C[\![P]\!] \xrightarrow{a\langle m\rangle} P''$ *then* P' *exists s.t.*
$P \xrightarrow{a\langle Tr_m\rangle} P'$, *and* $P'' = C[\![P']\!]$;

(c) if $C[\![P]\!] \xrightarrow{\tau} P''$ *then either*

- P' *exists s.t.* $P \xrightarrow{\tau} P'$ *and* $P'' = C[\![P']\!]$, *or*
- $\widetilde{b}, G, F$ *exist s.t.*
  $P \xrightarrow{\tau} \equiv \boldsymbol{\nu}\,\widetilde{b}\,(G\langle F\rangle)$ *and* $P'' \equiv C[\![\boldsymbol{\nu}\,\widetilde{b}\,(G\langle Tr_m\rangle\ \{m := F\})]\!]$;

(d) if $C[\![P]\!] \xrightarrow{\mu} P''$ *and* μ *is a first-order input or output, then* P' *exists s.t.*
$P \xrightarrow{\mu} P'$, *and* $P'' = C[\![P']\!]$. □

Secondly, let us point out that by definitions of $\mathcal{T}$ and $\mathcal{F}$, if P is a first-order process then it is not modified by C, i.e.

$$C[\![P]\!] = P$$

Thirdly, suppose that P is a HOπ process which can only perform first-order actions. This does not imply that P is also a π-calculus process, as *internally* P could perform communications of agents. But if some extension of the domain of $\approx_e$ is conceded, then $\approx_e$ can be used to compare P with π-calculus processes, and from Lemma 5.2.2 we get

$$P \approx_e C[\![P]\!]$$

Optimisations?

There are critical points in the definition of $\mathcal{C}$ on agents which is worth indicating. We doubt that non-trivial improvements are possible without losing full abstraction. The first optimisation one might be tempted of, is on the output of agent-variables, defining

$$\mathcal{C}[\![\overline{a}\langle X\rangle.Q]\!] \stackrel{def}{=} \overline{a}\langle x\rangle.\mathcal{C}[\![Q]\!] \qquad (*)$$

This is shorter than the rule we gave; for instance, in the simplest case where X is a first-order abstraction, the definition in Table 5–2 gives:

$$\mathcal{C}[\![\overline{a}\langle X\rangle.Q]\!] \stackrel{def}{=} \left(\overline{a}\langle m\rangle.\mathcal{C}[\![Q]\!]\right)\{m := (y)\overline{x}\langle y\rangle\} \qquad (**)$$

The intuitive justification for $(*)$ would be the following. Suppose that previous interactions have instantiated the variable X of the term $\overline{a}\langle X\rangle.Q$ with F. In the encoding π-calculus terms the simulation of these interactions causes the instantiation of the name x with a trigger, say m_F, to $\mathcal{C}[\![F]\!]$. Now, with the rule $(*)$ this same trigger m_F is then transmitted in the output along a. Instead, with the rule $(**)$ a new name-trigger m to the term $(y)\overline{m_F}\langle y\rangle$ is transmitted: This just seems to introduce a further level of indirection to the activation of $\mathcal{C}[\![F]\!]$.

But rule $(*)$ in general is *not* sound. The problem has to do with sharing. With rule $(*)$ two outputs of the same variable become at first-order outputs of name-triggers, or pointers, to the same term; this identity can be recognised and affect the successive behaviour. Consider in fact

$$P \stackrel{def}{=} \boldsymbol{\nu}\, a\left(\overline{a}\langle F\rangle \mid a(X).\overline{b}\langle X\rangle.\overline{b}\langle X\rangle\right)$$

$$Q \stackrel{def}{=} \boldsymbol{\nu}\, a\left(\overline{a}\langle F\rangle \mid a(X).\overline{b}\langle F\rangle.\overline{b}\langle F\rangle\right)$$

Clearly P and Q are equivalent, since they reduce in one step to the same term. But adopting rule $(*)$ their translations are not! In fact we would have

$$\mathcal{C}[\![P]\!] \stackrel{def}{=} \boldsymbol{\nu}\, a\left(\overline{a}\langle m\rangle\,\{m := F\} \mid a(x).\overline{b}\langle x\rangle.\overline{b}\langle x\rangle\right)$$

$$\mathcal{C}[\![Q]\!] \stackrel{def}{=} \boldsymbol{\nu}\, a\left(\overline{a}\langle m\rangle\,\{m := F\} \mid a(x).\left(\overline{b}\langle m_1\rangle.\overline{b}\langle m_2\rangle\,\{m_2 := F\}\right)\{m_1 := F\}\right)$$

which can be distinguished since after the initial interaction, $\mathcal{C}[\![Q]\!]$ can perform two outputs of private (and hence distinct) names at b, namely m_1 and m_2, whereas in $\mathcal{C}[\![P]\!]$ the two outputs at b communicate the same name.

For similar reasons, the optimisation

$$\mathcal{C}[\![X\langle Y\rangle]\!] \stackrel{def}{=} \overline{x}\langle y\rangle.\mathbf{0}$$

is not sound. For this, take the equivalent HOπ processes

$$c(X).\boldsymbol{\nu}\, a\left(a(Y).(X\langle Y\rangle \mid X\langle Y\rangle) \mid \overline{a}\langle F\rangle\right)$$

and

$$c(X).\boldsymbol{\nu}\, a\left(a(Y).(X\langle F\rangle \mid X\langle F\rangle) \mid \overline{a}\langle F\rangle\right).$$

Their π-calculus translations would be distinguished by reasoning similarly as above. There are however situations when the optimisations considered are indeed sound; we leave for future work more precise answers to this issue.

5.3 Related work: Thomsen's encoding and higher-order bisimulation

The first attempt at encoding a higher-order process calculus into π-calculus was made by Thomsen. For this, he used Plain CHOCS (PC) which, as mentioned in Section 2.3, corresponds to the subcalculus of HOπ which has the sorting $\{Name \mapsto ((\,))\}$.

Thomsen's study acted as stimulus and basis for our work. When applied to PC, our compilation $\mathcal{C}$ coincides with Thomsen's translation and in this sense can be seen as an extension of the latter. Our analysis however strengthens and completes Thomsen's in various aspects. First, PC is a (special case of) a second-order language, whereas HOπ is of ω-order. Secondly, to establish an operational correspondence between PC processes and their π-calculus encodings, Thomsen has to make some modifications to the transition system for PC. This seems rather

arbitrary and obscures the meaning of the results obtained. Thirdly, Thomsen does not succeed in proving full abstraction for his encoding, i.e. something like

$$P \stackrel{\sim}{\leftrightarrow}_1 Q \text{ iff } \mathcal{C}[\![P]\!] \stackrel{\sim}{\leftrightarrow}_2 \mathcal{C}[\![Q]\!] \tag{5.1}$$

for some natural equivalence $\stackrel{\sim}{\leftrightarrow}_1$ and $\stackrel{\sim}{\leftrightarrow}_2$. But he conjectures that this is true if $\stackrel{\sim}{\leftrightarrow}_1$ is the variant of higher-order bisimulation obtained using the modified transition system and $\stackrel{\sim}{\leftrightarrow}_2$ is the ordinary bisimulation for π-calculus as defined in [MPW92].

But we can show that the implication from right to left fails. Take the processes

$$P \stackrel{def}{=} \overline{a}\langle \mathbf{0} \rangle.\mathbf{0} \quad \text{and} \quad Q \stackrel{def}{=} (\boldsymbol{\nu}\, x)\overline{a}\langle \overline{x}.\mathbf{0} \rangle.\mathbf{0} \tag{5.2}$$

There is no way using higher-order bisimulation, to avoid the distinction between them (see Section 1.3), although one reasonably expects them to be equivalent. This is also confirmed by the result of the encoding into π-calculus, where $\mathcal{C}[\![P]\!]$ and $\mathcal{C}[\![Q]\!]$ are strongly bisimilar.

All this offers us the opportunity of a comparison between higher-order bisimulation and barbed bisimulation in higher-order process calculi. First, the precise definition of higher-order bisimulation. We present here its version for HOπ.

Definition 5.3.1 *Higher-order bisimulation, written $\sim_{\mathrm{H}}$, is the largest symmetric relation on HOπ processes s.t. $\sim_{\mathrm{H}}$ is a first-order simulation and $P \sim_{\mathrm{H}} Q$ implies:*

1. *whenever $P \xrightarrow{a\langle F \rangle} P'$, there exists Q' s.t. $Q \xrightarrow{a\langle F \rangle} Q'$ and $P' \sim_{\mathrm{H}} Q'$,*
2. *whenever $P \xrightarrow{(\nu \widetilde{b})\overline{a}\langle F \rangle} P'$ with $\widetilde{b} \cap fn(P,Q) = \emptyset$, there exist Q', E and $\widetilde{c}$ with $\widetilde{c} \cap fn(P,Q) = \emptyset$ s.t. $Q \xrightarrow{(\nu \widetilde{c})\overline{a}\langle E \rangle} Q'$ and $F \sim_{\mathrm{H}} E$, $P' \sim_{\mathrm{H}} Q'$.*

[In clause (2), $\sim_{\mathrm{H}}$ is defined on abstractions and open agents as expected, i.e. $F \sim_{\mathrm{H}} E$ if for every G (resp. for every name a if F, E are first-order abstractions), $F\langle G \rangle \sim_{\mathrm{H}} E\langle G \rangle$ (resp. $F\langle a \rangle \sim_{\mathrm{H}} E\langle a \rangle$).] □

There is a necessary remark to make about clause (2). If we had faithfully followed the definition of $\sim_{\mathrm{H}}$ on PC in [Tho90], we would have proposed:

(2′) whenever $P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle F\rangle} P'$ with $\widetilde{b} \cap fn(P,Q) = \emptyset$, there exist Q', E s.t. $Q \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle E\rangle} Q'$ and $F \sim_{\mathrm{H}} E$, $P' \sim_{\mathrm{H}} Q'$.

The difference is that in (2′) the same set of names, namely $\widetilde{b}$, is exported by P and Q. Now, this might appear reasonable if, following Thomsen, we added the transition rule

$$\frac{P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle F\rangle} P'}{P \xrightarrow{(\nu\,\widetilde{b}')\overline{a}\langle F\rangle} P'} \quad \widetilde{b} \subseteq \widetilde{b}' \text{ and } \widetilde{b}' \cap fn(P) = \emptyset \tag{5.3}$$

which allows to augment the set of names exported. Unfortunately, rule (5.3) does not seem to be sufficient. For instance, consider the following processes, in which for simplicity we have used only names which can either carry a process or the empty tuple:

$$P = \overline{a}\langle \mathbf{0}\rangle.\mathbf{0} \qquad\qquad Q = (\boldsymbol{\nu}\, c)\overline{a}\langle \boldsymbol{\nu}\, b\,(b.c)\rangle.\mathbf{0}$$

Since in both cases the object part of the name a is a deadlocked process, we would like to have $P \sim_{\mathrm{H}} Q$. However, they are *not* equivalent applying clause (2′), and even adding the rule (5.3). In fact, in every action of Q at least one restriction — namely the restriction on c — is exported. This is not true for P, since we have $P \xrightarrow{\overline{a}\langle 0\rangle} \mathbf{0}$ which hence, cannot be matched by Q. Building on this example, one can also show that if (2′) is used, then the congruence of $\sim_{\mathrm{H}}$ over the restriction operator is lost. In fact, take $R \stackrel{def}{=} \overline{a}\langle \boldsymbol{\nu}\, b\,(b.c)\rangle.\mathbf{0}$; then $P \sim_{\mathrm{H}} R$, but

$$P \sim_{\mathrm{H}} \boldsymbol{\nu}\, c\, P \not\sim_{\mathrm{H}} \boldsymbol{\nu}\, c\, R = Q$$

A way of adjusting clause (2′) could also be to keep the rule (5.3) but then replace $Q \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle E\rangle} Q'$ in (2′) with $Q \xrightarrow{(\nu\,\widetilde{b}')\overline{a}\langle E\rangle} Q'$, for some $\widetilde{b}' \supseteq \widetilde{b}$. Then we conjecture that the induced relation would be the same as the one obtained from our Definition 5.3.1.

We now try to compare our version of weak higher-order bisimulation, denoted by $\approx_{\mathrm{H}}$ and defined in the standard way from $\sim_{\mathrm{H}}$, with barbed equivalence.

Theorem 5.3.2 *The relation $\approx_{\mathrm{H}}$ is strictly included in $\approx$.*

One can show $\approx_{\mathrm{H}} \neq \approx$ using the processes P, Q in (5.2). The hard part is to show the inclusion. The "simplest" way we could find is based on the proof that

$$\mathcal{R} = \Big\{(P,Q) \;:\; \exists \tilde{b}, n,\ P_i \approx_{\mathrm{H}} Q_i, \text{ for } 1 \leq i \leq n, \text{ and } P = \boldsymbol{\nu}\,\tilde{b}\big(\Pi_{i=1}^{n}\, \mathcal{T}[\![P_i]\!]\big),\ Q = \boldsymbol{\nu}\,\tilde{b}\big(\Pi_{i=1}^{n}\, \mathcal{T}[\![Q_i]\!]\big)\Big\} \quad \subseteq \approx_{\mathrm{Tr}}$$

From this, if $P \approx_{\mathrm{H}} Q$ then we can infer $\mathcal{T}[\![P]\!] \approx_{\mathrm{Tr}} \mathcal{T}[\![Q]\!]$, from which, using the theory developed in Chapter 4, also $P \approx Q$. We cannot hope to get the first of these implications by proving that $\mathcal{T}$ respects $\approx_{\mathrm{H}}$ because this is in general false. The proof that $\mathcal{R} \subseteq \approx_{\mathrm{Tr}}$ is non-trivial; for this — among the others — we need a more powerful up-to technique than those described in Section 2.5.3; for instance the one employing expansions and described in the last section of [SM92] would do. We have not reported the details here, on one hand because they involve technicalities which go beyond the scope of this thesis, on the other hand because if we had available the congruence of $\approx_{\mathrm{H}}$ over parallel composition and restriction (which seem non-trivial), the inclusion $\approx_{\mathrm{H}} \subseteq \approx$ could be derived with a very simple proof. Therefore, since without these congruences $\approx_{\mathrm{H}}$ is probably almost useless, we thought more interesting to present the simpler – although incomplete – proof, to show where the need for the congruence of $\approx_{\mathrm{H}}$ arises.

We want to prove that

$$\mathcal{R}' = \{(P,Q) \;:\; P \approx_{\mathrm{H}} Q\}$$

is a $\approx_{\mathrm{Nr}}$-bisimulation. The interesting part is the verification of the clause for higher-order outputs. If $P \xrightarrow{(\nu\,\tilde{b})\overline{a}\langle F\rangle} P'$, then we need $\tilde{c}, Q'$, and E s.t. $Q \xrightarrow{(\nu\,\tilde{c})\overline{a}\langle E\rangle} Q'$ and

$$\boldsymbol{\nu}\,\tilde{b}(\,!m \star F \mid P')\ \mathcal{R}'\ \boldsymbol{\nu}\,\tilde{c}(\,!m \star E \mid Q') \tag{5.4}$$

First, using α-conversion we can suppose $\tilde{b} \cap fn(P,Q) = \emptyset$. Now, since $P \approx_{\mathrm{H}} Q$, we have

$$Q \xrightarrow{(\nu\,\tilde{c})\overline{a}\langle E\rangle} Q',$$

where $\tilde{c} \cap fn(P,Q) = \emptyset$, $F \approx_{\mathrm{H}} E$, $P' \approx_{\mathrm{H}} Q'$. At this point, we would like to use the congruence properties of $\approx_{\mathrm{H}}$ and infer

$$(\boldsymbol{\nu}\, \tilde{b} \cup \tilde{c})(!m \star F \mid P') \approx_{\mathrm{H}} (\boldsymbol{\nu}\, \tilde{b} \cup \tilde{c})(!m \star E \mid Q')$$

from here, (5.4) can be derived by canceling the appropriate restrictions. □

Notice that the result above also proves that the implication from left to right in (5.1) holds.

So what can be our conclusion on the comparison between $\approx_{\mathrm{H}}$ and $\approx$? First of all, in our view the latter is more natural than the former; some distinction made by $\approx_{\mathrm{H}}$ seem difficult to justify. However, at least we can say that $\approx_{\mathrm{H}}$ is *sound*, in that the equivalences that it gives are true for $\approx$. This means that $\approx_{\mathrm{H}}$ can represent a useful tool for proving $\approx$ via *decomposition*, since in the clause for output, $\approx_{\mathrm{H}}$ allows a separate analysis of the object part and of the continuation. But to make this conceivable, it remains to prove the congruence properties of $\approx_{\mathrm{H}}$ and to check whether the quantification over all agents in clause (1) of the definition of $\approx_{\mathrm{H}}$ can be removed by employing triggers (or something similar to these) as it is for $\cong_{\mathrm{Nr}}$.

Chapter 6

An Investigation into Functions as Processes

6.1 Introduction

In [Mil90a], Milner presents the encodings of lazy λ-calculus and (a weak version of) call-by-value λ-calculus into π-calculus. In each case, he compares the equivalence induced on λ-terms, where two λ-terms are identified if their process encodings are behavioral equivalent, with Abramsky's applicative bisimulation. He proves that the former implies the latter, but that the other way round fails. It is left as an open problem to characterise this induced equivalence. It remains also to be studied which kind of λ-model – if any – can be constructed from the π-terms.

The purpose of this chapter is twofold:

- To continues the study of Milner's encodings;
- To show the usefulness of compilation $\mathcal{C}$.

Using the abstraction power of the HOπ, we give encodings both for the lazy and for the call-by-value λ-calculus which are easier to understand and to deal with than those available in the π-calculus. This is especially evident for the lazy

λ-calculus which, following Boudol's terminology in [Bou89], is made a ***subcalculus*** of HOπ, in the sense that β-reduction becomes an *instance* of the communication law with a one-to-one correspondence between the two. By applying compilation $\mathcal{C}$, we can turn the HOπ encodings into π-calculus encodings which can then be compared with Milner's. This is interesting

- to test the canonicity of the encodings,
- to examine the efficiency of $\mathcal{C}$.

In the lazy λ-calculus the outcome of applying $\mathcal{C}$ to the HOπ encoding is precisely Milner's encoding (see the commutative diagram at page 12). Hence all results and properties proved for one of the encodings can then be extended to the other. By working in the HOπ, we are able to show that they give rise indeed to a λ-model, where a weak form of extensionality holds. The model is not fully abstract, though. To obtain full abstraction we follow two directions: In the *restrictive* approach, based on the use of barbed bisimulation, the semantic domain is cut down; in the *expansive* approach λ-calculus is enriched with constants to obtain a *direct* characterisation of the equivalence induced by the encodings.

In call-by-value the situation is less clear. In [Mil90a] Milner presents two candidates for the encoding, and it is not obvious which one should be preferred: The first one allows an easier reasoning, but the second one is more efficient. Moreover, when applied to the HOπ encoding, compilation $\mathcal{C}$ does not return either of them. Apparently, to obtain them, some code transformation has to be carried out. The study of these transformations leads to interesting conclusions. First, it suggests a correction of the order in which some actions appear in Milner's encodings. This rearrangement does not affect the operational correspondence between λ- and π-terms. However, it affects the equivalence on the encoding π-terms, in a way which makes the encoding more faithful to the encoded call-by-value discipline. Secondly, the study of the transformations reveals that β-reduction is not valid in Milner's second encoding, which reduces a lot its importance. The counter-

example is fairly sophisticated and we doubt we could have obtained it without going through HOπ. [1]

We adopt $\approx^c$ (weak barbed congruence) as behavioural equivalence for the terms of our calculi. In the proofs, we shall often use the characterisations of $\approx^c$ in terms of weak early congruence for π-calculus (Corollary 4.7.9), and in terms of weak context congruence and weak normal congruence for HOπ (Corollary 3.3.4). However, due to the particular structure of agents encoding λ-terms, the results we present should be — to a reasonable extent — independent from the choice of the process equivalence.

A remark about the use of the full abstraction of compilation $\mathcal{C}$ and of the characterisations of $\approx^c$, presented in the previous chapters. These results were proved on agents which may exchange one only value in their communications. We already stressed that this does not intend to be a limitation on their applicability. The choice of unary-tuple was made to make the proof-techniques more readable — by distinguishing the requirements on higher-order and first-order objects — and the generalisation to arbitrary-tuple communications is completely smooth (see also the discussion in Section 4.7.3). In this chapter, we shall apply such results to the polyadic calculi (precisely, to processes which may perform *binary* communications).

Structure of the chapter: In Section 6.2 we recall some preliminaries on λ-calculus. In Section 6.3 we present the encodings of lazy λ-calculus into π-calculus and HOπ and we make the comparison between them via $\mathcal{C}$. In Section 6.4 we do the same for call-by-value λ-calculus. In Sections 6.5 and 6.6 we examine the lazy λ-calculus encodings from the point of view of the model theory of λ-calculus and we study the equivalence on the λ-terms induced by the behavioural equivalence adopted on the process calculi.

[1] The version of [Mil90a] appeared in the *Jour. of Math. Structures* was written when the results in this chapter were already known and thus presents only the first encoding.

6.2 λ-calculus: Preliminaries

6.2.1 Syntax and notation

λ-calculus is the commonly accepted foundation for functional programming. The syntax is simple, and based on the operators of abstraction and application. An evaluation mechanism defines the operational meaning of a term built out of this syntax.

We use x and y to range over the set of λ-calculus variables. To simplify the encodings, x, y will also represent names in the π-calculus. We use L, M, N to range over the class Λ° of open λ-terms, as usual given by:

$$M \quad ::= \quad x \quad | \quad \lambda x.M \quad | \quad MN$$

The definitions of free variables, closed terms, substitutions, α-conversion etc. are taken to be the standard ones (see [Bar84], [HS86]). We shall not distinguish between α-convertible terms and we write $M = N$ if M and N are α-convertible. The subclass of closed λ-terms is denoted by Λ. A closed term $C_\lambda[\cdot]$ with a single hole in it is a *λ-context*. We use Ω to denote the (divergent) term $\lambda x.xx(\lambda x.xx)$.

6.2.2 λ-calculus encodings

The idea common to all various attempts at embedding λ-calculus into a process calculus [Bou89,Bou92,Tho90,Mil90a] is to view functional application as a particular parallel combination of two agents, the function and its argument, and β reduction as a particular case of communication. The encodings we shall present follow the same idea.

A λ-term is encoded as a map from names of a certain sort s to processes, that is, as an abstraction of sort (s). To lighten the exposition, we shall omit the sortings. To derive them, it is enough to say that they are always the expected ones, and that in each case, at most three sorts are needed; one for names p, q, r, another one for x, y, the last one for v, w. Moreover, these sortings always behave

well w.r.t. compilation $\mathcal{C}$, in the following sense: Whenever an HOπ encoding Ob_H is compared with a π-calculus encoding Ob_π via $\mathcal{C}$, and Ob_H, Ob_π do not include redundant sorts (i.e. sorts not used in the encoding) then it is always true that $\mathcal{C}[\![Ob_H]\!] = Ob_\pi$.

In the algebraic manipulations on HOπ processes sometimes we shall need the following simple form of the *expansion law*. Suppose that the process R can only perform visible actions at names in $\widetilde{m}$, i.e., $\{a \,:\, R \Downarrow_a\} \subseteq \widetilde{m}$; then

$$\boldsymbol{\nu}\, a, \widetilde{m}\left(\overline{a}\langle \widetilde{K}\rangle.P \mid a(\widetilde{U}).Q \mid R\right) \approx^{\mathrm{c}} \boldsymbol{\nu}\, a, \widetilde{m}\left(P \mid Q\{\widetilde{K}/\widetilde{U}\} \mid R\right)$$

Note that for $R = \mathbf{0}$, the result is proved by Lemmas 4.4.4(1) and 4.4.2, and Corollary 4.7.9. We shall use this expansion law a few times in this chapter.

We adopt the following — trivial — simplification of the encoding $\mathcal{C}$ on output processes whose continuation is the $\mathbf{0}$ process: We shall use

$$\mathcal{C}[\![\overline{a}\langle F\rangle.\mathbf{0}]\!] \stackrel{def}{=} \boldsymbol{\nu}\, m\,(\overline{a}\langle m\rangle.\,!\,m \star \mathcal{C}[\![F]\!])$$

rather than

$$\mathcal{C}[\![\overline{a}\langle F\rangle.\mathbf{0}]\!] \stackrel{def}{=} \boldsymbol{\nu}\, m\,(\overline{a}\langle m\rangle.\mathbf{0} \mid\, !\,m \star \mathcal{C}[\![F]\!])$$

6.3 The Lazy λ-Calculus Encoding

The lazy λ-calculus is proposed by Abramsky in [Abr87] as a basis for lazy functional programming languages and the evaluation mechanism is guided by what the implementation of such languages suggests; in particular reductions are forbidden within a λ-abstraction.

The reduction relation $\Longrightarrow \subseteq \Lambda \times \Lambda$ is defined by rules (β) and (App), the core rules for the lazy λ-calculus, plus rules *Refl* and *Trans*, describing the reflexive and transitive nature of $\Longrightarrow$.

$$(\beta)\ (\lambda x.M)N \Longrightarrow M\{N/x\} \qquad (App)\ \frac{M \Longrightarrow M'}{MN \Longrightarrow M'N}$$

$$(Refl)\ M \Longrightarrow M \qquad (Trans)\ \frac{M \Longrightarrow M' \qquad M' \Longrightarrow M''}{M \Longrightarrow M''}$$

6.3.1 Applicative Bisimulation

The lazy reduction strategy admits only reductions at the extreme left of a term and hence is deterministic. A term either converges to an abstraction or diverges. The convergence predicate $\Downarrow$ is so defined:

$$\lambda x.M \Downarrow \qquad \frac{M \Longrightarrow M' \Downarrow}{M \Downarrow}$$

Inspired by the work of Milner and Park in concurrency [Par81,Mil83], Abramsky [Abr87] introduces an operational equivalence on λ-terms called *applicative bisimulation*, built on the idea that convergence is the only observable property.[1]

Definition 6.3.1 Applicative bisimulation *is the largest symmetric relation* $\cong \subseteq \Lambda \times \Lambda$ *s.t. if* $M \cong N$ *and* $M \Longrightarrow \lambda x.M'$, *then some* N' *exists s.t.* $N \Longrightarrow \lambda x.N'$ *and for all* L, $M'\{L/x\} \cong N'\{L/x\}$. □

The relation $\cong$ is extended to open λ-terms using substitutions: if M and N are open λ-terms with $\{x_1, \ldots, x_n\}$ as free variables, then $M \cong N$ if for all closed $L_1, \ldots, L_n$, it holds that $M\{L_1/x_1, \ldots, L_n/x_n\} \cong N\{L_1/x_1, \ldots, L_n/x_n\}$. It is easy to see that $\cong$ is an equivalence. Moreover, since the reduction relation is deterministic, it holds that

Lemma 6.3.2 *If* $M \longrightarrow M'$, *then* $M \cong M'$. □

The relation $\cong$ has been extensively studied by Abramsky and Ong [Ong88, AO89]. They show that $\cong$ is a congruence using the characterisation in Theorem 6.3.4 below.

Definition 6.3.3 *Let* $M, N \in \Lambda$. *We write*

[1] Abramsky's original definition is in terms of *simulation* rather then bisimulation. In fact, he mainly works with preorders. He then considers the equivalence induced by such preorder: Since the reduction relation is deterministic, his definition is equivalent to our Definition 6.3.1. Moreover Abramsky uses the name applicative bisimulation for the preorder; but it seems more accurate and in line with the concurrency tradition to reserve this word for the equivalence.

- $M \dot{\approx} N$ *when* $M \Downarrow$ *iff* $N \Downarrow$.
- $M \approx^{c} N$ *when* $C_\lambda[M] \dot{\approx} C_\lambda[N]$ *holds for each closed* λ*-context* $C_\lambda[\cdot]$.

The symbology in Definition 6.3.3 is motivated by the fact that if we think of λ as the only port of λ-calculus and $M \Downarrow$ as meaning "M can interact at λ", then the relations introduced are the corresponding on λ-calculus of weak barbed bisimulation and congruence.

Theorem 6.3.4 (Abramsky, Ong) *On closed* λ*-term,* $\cong$ *and* $\approx^{c}$ *coincide.*

PROOF: The difficult part is the inclusion $\cong \subseteq \approx^{c}$. This result is proved in [Abr87] by going through domain logic. A simpler proof has been suggested by Alan Stoughton and proceeds by induction on the structure of the contexts. See [AO89] for the details. □

Notice how close the above result resembles those in Chapters 4 and 5 about direct characterisations of barbed congruence in π-calculus and HOπ. This similitude will be in fact exploited in Section 6.6.1 to obtain certain full abstraction results.

Corollary 6.3.5 *For each* $M, N \in \Lambda^{o}$ *and* λ*-contexts* $C_\lambda[\cdot]$, $M \cong N$ *implies* $C_\lambda[M] \cong C_\lambda[N]$.

PROOF: If M and N are closed, the result follows from Theorem 6.3.4. The result is extended to open terms proceeding by induction on the structure of $C_\lambda[\cdot]$. □

6.3.2 Encoding into π-calculus

We use $\mathcal{P}$ to denote Milner's encoding of the lazy λ-calculus into π-calculus. If $\mathcal{P}[\![M]\!]\langle p \rangle$ is a process resulting from the translation of the term M, the name p acts as a pointer to the argument sequence for M. If M reduces to a λ-abstraction, the corresponding derivative of $\mathcal{P}[\![M]\!]\langle p \rangle$ will receive along the name p a pointer

to its first argument and a pointer to the rest of its argument sequence. In the application rule, the restriction on q prevents interference from other processes.

$$\begin{aligned}
\mathcal{P}[\![\lambda x.M]\!] &\stackrel{def}{=} (p)p(x,q).\mathcal{P}[\![M]\!]\langle q\rangle \\
\mathcal{P}[\![x]\!] &\stackrel{def}{=} (p)\overline{x}\langle p\rangle.\mathbf{0} \\
\mathcal{P}[\![MN]\!] &\stackrel{def}{=} (p)\boldsymbol{\nu}\, q\left(\mathcal{P}[\![M]\!]\langle q\rangle \mid (\boldsymbol{\nu}\, x)\overline{q}\langle x,p\rangle.!\, x(r).\mathcal{P}[\![N]\!]\langle r\rangle\right) \\
& \qquad x \text{ not free in } \mathcal{P}[\![N]\!]
\end{aligned}$$

It seems established that $\mathcal{P}$ is canonical, i.e. it is the "best" or "simplest" encoding of the lazy λ-calculus into π-calculus. Nevertheless, one has to think carefully about it – in particular at the encoding of application – to understand that it really does work (see [Mil90a] for examples).

6.3.3 Encoding into HOπ

The encoding into HOπ makes very transparent the idea of regarding function application as a special case of communication. For convenience, a variable x of the λ-calculus is mapped into its upper-case variable X in HOπ.

$$\begin{aligned}
\mathcal{H}[\![\lambda x.M]\!] &\stackrel{def}{=} (p)p(X,q).\mathcal{H}[\![M]\!]\langle q\rangle \\
\mathcal{H}[\![x]\!] &\stackrel{def}{=} X \\
\mathcal{H}[\![MN]\!] &\stackrel{def}{=} (p)\boldsymbol{\nu}\, q\left(\mathcal{H}[\![M]\!]\langle q\rangle \mid \overline{q}\langle\mathcal{H}[\![N]\!],p\rangle.\mathbf{0}\right)
\end{aligned}$$

The theorem below shows how the encoding behaves. There is a one-to-one correspondence between reductions in λ-terms and in their encodings; moreover, to capture validity of β reduction, we simply need an elementary use of structural congruence. Therefore, following Boudol [Bou89], we can claim that *lazy λ-calculus is a subcalculus of HOπ*.

Proposition 6.3.6 (operational correspondence) *Let $M \in \Lambda$:*

1. *(a) If $M \longrightarrow M'$, then $\mathcal{H}[\![M]\!]\langle p\rangle \longrightarrow\equiv \mathcal{H}[\![M']\!]\langle p\rangle$,*

(b) *the converse, i.e.*

if $\mathcal{H}[\![M]\!]\langle p\rangle \longrightarrow Q$, *then* $\exists M' \in \Lambda$ *s.t.* $Q \equiv \mathcal{H}[\![M']\!]\langle p\rangle$ *and* $M \longrightarrow M'$.

2. (a) *If* $\mathcal{H}[\![M]\!]\langle p\rangle \xrightarrow{\mu}$ *for some* $\mu \neq \tau$, *then* M *must be of the form* $\lambda x.M'$,

(b) *the converse, i.e.*

if M *is of the form* $\lambda x.M'$, *then* $\mathcal{H}[\![M]\!]\langle p\rangle \xrightarrow{\mu}$, *for some* $\mu \neq \tau$.

PROOF: We only consider (1), since (2) is easy. Both (a) and (b) can be proved by induction on the structure of M.

- $M = \lambda x.M'$.

  M and $\mathcal{H}[\![M]\!]\langle p\rangle$ have no reduction.

- $M = M_1 M_2$ and $M_1 = \lambda x.M_3$.

  Then $M \longrightarrow M_3\{M_2/x\}$. On the other hand,

$$\begin{aligned}
\mathcal{H}[\![(\lambda x.M_3)M_2]\!]\langle p\rangle &= \\
\boldsymbol{\nu}\, q\left(q(X,r).\mathcal{H}[\![M_3]\!]\langle r\rangle \mid \overline{q}\langle \mathcal{H}[\![M_2]\!], p\rangle\right) &\longrightarrow \\
\boldsymbol{\nu}\, q\left(\mathcal{H}[\![M_3]\!]\langle p\rangle\{\mathcal{H}[\![M_2]\!]/X\}\right) &\equiv \qquad (q \text{ not free in the body}) \\
\mathcal{H}[\![M_3]\!]\langle p\rangle\{\mathcal{H}[\![M_2]\!]/X\} &= \qquad \text{(because of the compositional} \\
\mathcal{H}[\![M_3\{M_2/x\}]\!]\langle p\rangle & \qquad \text{definition of the encoding)}
\end{aligned}$$

- $M = M_1 M_2$ and M_1 is not of the form $\lambda x.M_3$.

  Immediate using the inductive hypothesis, since it must be that $M_1 \longrightarrow M_1'$ and $M' \equiv M_1' M_2$. □

6.3.4 Correspondence between the two encodings

It is easy to verify that the output of compilation $\mathcal{C}$ on the HOπ encoding is exactly Milner's encoding. Using '$\circ$' to denote function composition, we therefore have the following:

Theorem 6.3.7 $\mathcal{C} \circ \mathcal{H} = \mathcal{P}$. □

In virtue of Theorem 5.2.1, this means that we can infer for Milner's encoding all results we can prove working with the HOπ encoding. The latter is simpler to study because it yields a more direct correspondence with the encoded λ-terms; it will be accurately examined in Sections 6.5 and 6.6.

6.4 The Call-by-Value Encoding

In call-by-value λ-calculus, reductions may only occur when the argument is a value, i.e. an abstraction. The reduction relation $\Longrightarrow \subseteq \Lambda \times \Lambda$ used by Milner in [Mil90a] is described by the usual rules *Refl* and *Trans* plus the rules β_v, App_L, App_R.

$$(\beta_v)\ (\lambda x.M)\lambda y.N \Longrightarrow M\{\lambda y.N/x\}$$

$$(App_L)\ \frac{M \Longrightarrow M'}{MN \Longrightarrow M'N} \qquad (App_R)\ \frac{N \Longrightarrow N'}{MN \Longrightarrow MN'}$$

Reduction $\Longrightarrow$ is no longer deterministic. However, convergence remains deterministic and *strong*: that is if $M \Longrightarrow \lambda x.M'$, then $\lambda x.M'$ is unique and all reduction sequences from M are finite. Convergence and applicative bisimulation are defined for call-by-value in the same way as for lazy λ-calculus.

Now we shall try to repeat for call-by-value what we did in Section 6.3 for lazy λ-calculus: We propose an encoding into HOπ and then we compare it with Milner's into π-calculus through compilation $\mathcal{C}$.

With respect to lazy λ-calculus, the call-by-value encodings are slightly more involved. They also lose their apparent canonicity, which is implicitly confirmed by the fact that in his original work [Mil90a], Milner presents *two* candidates for the encoding. Basically, the problems in the translation of call-by-value are originated by the following *dichotomy* in the behaviour of a λ-abstraction. Take the term MN: Both M and N could reduce to an abstraction; but if it is M, then the abstraction is destined to perform an *input* of a value, whereas if it is N, the

abstraction represents an *output* value. This causes disagreement on whether the process which encodes a λ-abstraction should first perform an input or an output.

6.4.1 Encoding into π-calculus

Milner's two candidates for the encoding in [Mil90a] differ only in the rule for the translation of a variable. In the second encoding this rule is simpler, but the first one allows easier reasoning and proofs. In both cases, p is used in $\mathcal{P}_i[\![M]\!]\langle p\rangle$ to communicate (with an output action) that M has reduced to a value. In contrast with lazy λ-calculus, the translation of an application allows the two arguments M and N to run in parallel. Then the dichotomy in the behaviour of a λ-abstraction is solved by the arbiter $App\langle p,q,r\rangle$ which imposes the correct interaction between M and N.

FIRST CANDIDATE

$$\begin{aligned}
\mathcal{P}_1[\![\lambda x.M]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.!\,y(w).w(x,q).\mathcal{P}_1[\![M]\!]\langle q\rangle \\
\mathcal{P}_1[\![x]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.!\,y(w).\overline{x}\langle w\rangle.\mathbf{0} \\
\mathcal{P}_1[\![MN]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{P}_1[\![M]\!]\langle q\rangle \mid \mathcal{P}_1[\![N]\!]\langle r\rangle \mid App\langle p,q,r\rangle\big) \\
&\qquad \text{where } App \stackrel{def}{=} (p,q,r)q(x).(\boldsymbol{\nu}\, v)\overline{x}\langle v\rangle.r(y).\overline{v}\langle y,p\rangle.\mathbf{0}
\end{aligned}$$

SECOND CANDIDATE

Let us denote it with $\mathcal{P}_2$. The only difference with the first encoding is in the clause for the variable, which now is

$$\mathcal{P}_2[\![x]\!] \stackrel{def}{=} (p)\overline{p}\langle x\rangle.\mathbf{0}$$

6.4.2 Encoding into HOπ

The idea of the encoding into HOπ is the same as for the encoding into π-calculus, but it can be implemented more directly.

$$\begin{aligned}
\mathcal{H}[\![\lambda x.M]\!] &\stackrel{def}{=} (p)\overline{p}\big\langle (w)w(X,q).\mathcal{H}[\![M]\!]\langle q\rangle\big\rangle.\mathbf{0} \\
\mathcal{H}[\![x]\!] &\stackrel{def}{=} (p)\overline{p}\langle X\rangle.\mathbf{0} \\
\mathcal{H}[\![MN]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{H}[\![M]\!]\langle q\rangle \mid \mathcal{H}[\![N]\!]\langle r\rangle \mid App_H\langle p,q,r\rangle\big) \\
&\qquad \text{where } App_H \stackrel{def}{=} (p,q,r)q(X).r(Y).\boldsymbol{\nu}\, v\,(X\langle v\rangle \mid \overline{v}\langle Y,p\rangle.\mathbf{0})
\end{aligned}$$

The format of the translation of variables resembles the one in Milner's second encoding $\mathcal{P}_2$. But the similarity is only syntactical. In section 6.4.3 we shall see, using the compilation $\mathcal{C}$, that semantically $\mathcal{H}$ agrees with $\mathcal{P}_1$ rather than $\mathcal{P}_2$.

It is enlightening to relate the HOπ encoding for call-by-value to the one for lazy λ-calculus. In the above rules for abstraction and variable, the "core" is the object part of the output at p, and has the same format of the corresponding rule for lazy λ-calculus. Now the rule for application in call-by-value should become clearer: The arbiter App_H receives along p and r the "cores" of $\mathcal{H}[\![M]\!]$ and $\mathcal{H}[\![N]\!]$ and then imposes on them the "lazy application".

The encoding into HOπ is therefore easier to understand than those in the π-calculus, although the abstraction power of HOπ does not seem to make the same difference as for lazy λ-calculus. To see how $\mathcal{H}$ works, we show the validity of β reduction.

Lemma 6.4.1 (validity of β reduction) $\mathcal{H}[\![(\lambda x.M)\lambda y.N]\!] \approx^{c} \mathcal{H}[\![M\{\lambda y.N/x\}]\!]$

PROOF: Let us abbreviate $\mathcal{H}[\![\lambda x.M]\!]\langle q\rangle$ as $\overline{q}\langle F_M\rangle$ and $\mathcal{H}[\![\lambda x.N]\!]\langle r\rangle$ as $\overline{r}\langle F_N\rangle$, where

$$F_M = (w)w(X,p).\mathcal{H}[\![M]\!]\langle p\rangle \quad \text{and} \quad F_N = (w)w(X,p).\mathcal{H}[\![N]\!]\langle p\rangle$$

By repeated use of the expansion law and some obvious simplification, we get

$$\begin{aligned}
\mathcal{H}[\![(\lambda x.M)\lambda y.N]\!]\langle p\rangle &= \\
(\boldsymbol{\nu}\, q,r)\big(\overline{q}\langle F_M\rangle \mid \overline{r}\langle F_N\rangle \mid App_H\langle p,q,r\rangle\big) &\approx^{\mathrm{c}} \\
\boldsymbol{\nu}\, v\,\big(F_M\langle v\rangle \mid \overline{v}\langle F_N, p\rangle\big) &= \qquad (*) \\
\boldsymbol{\nu}\, v\,\big(v(X,p).\mathcal{H}[\![M]\!]\langle p\rangle \mid \overline{v}\langle F_N,p\rangle\big) &\approx^{\mathrm{c}} \\
\mathcal{H}[\![M\{F_N/X\}]\!]\langle p\rangle &= \\
\mathcal{H}[\![M\{\lambda y.N/x\}]\!]\langle p\rangle &
\end{aligned}$$

where the last equality follows from the compositional definition of the encoding. Notice that $(*)$ is an instance of the lazy application in HOπ. The validity of the latter gives the correctness of the successive reduction.

6.4.3 Correspondence among the encodings

Let us apply compilation $\mathcal{C}$ to the HOπ encoding and see what we get back.

$$\begin{aligned}
\mathcal{C}[\![\mathcal{H}[\![\lambda x.M]\!]]\!] &= (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.!\,y(w).w(x,q).\mathcal{C}[\![\mathcal{H}[\![M]\!]]\!]\langle q\rangle \\
\mathcal{C}[\![\mathcal{H}[\![x]\!]]\!] &= (p)(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.!\,y(w).\overline{x}\langle w\rangle.\mathbf{0} \\
\mathcal{C}[\![\mathcal{H}[\![MN]\!]]\!] &= (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{C}[\![\mathcal{H}[\![M]\!]]\!]\big)\langle q\rangle \mid \mathcal{C}[\![\mathcal{H}[\![N]\!]]\!]\langle r\rangle \mid \\
&\qquad q(x).r(y).\boldsymbol{\nu}\, v\,\big(\overline{x}\langle v\rangle \mid (\boldsymbol{\nu}\, m)\overline{v}\langle m,p\rangle.!\,m(w).\overline{y}\langle w\rangle.\mathbf{0}\big) \\
&\qquad (p)(\boldsymbol{\nu}\, q,r)\big(\mathcal{C}[\![\mathcal{H}[\![M]\!]]\!]\big)\langle q\rangle \mid \mathcal{C}[\![\mathcal{H}[\![N]\!]]\!]\langle r\rangle \mid \\
&\qquad q(x).r(y).\boldsymbol{\nu}\, v\,\overline{x}\langle v\rangle.(\boldsymbol{\nu}\, m)\overline{v}\langle m,p\rangle.!\,m(w).\overline{y}\langle w\rangle.\mathbf{0}
\end{aligned}$$

The above use of $\sim^{\mathrm{c}}$ (strong barbed congruence) has been derived from the algebraic law

$$\boldsymbol{\nu}\, v\,(\overline{x}\langle v\rangle \mid \overline{v}\langle\widetilde{y}\rangle.P) \sim^{\mathrm{c}} \boldsymbol{\nu}\, v\,(\overline{x}\langle v\rangle.\overline{v}\langle\widetilde{y}\rangle.P)$$

(intuitively, since v is restricted, the action $\overline{v}\langle\widetilde{y}\rangle$ may only fire after the action $\overline{x}\langle v\rangle$). There are two differences between the encoding $\mathcal{C}\circ\mathcal{H}$ and Milner's encodings $\mathcal{P}_1$ and $\mathcal{P}_2$:

1. The order of the actions $r(y)$ and $\boldsymbol{\nu}\, v\,\overline{x}\langle v\rangle$ in the rule for application is reversed,

2. the process $\boldsymbol{\nu}\, m\, \overline{v}\langle m, p\rangle.!\, m(w).\overline{y}\langle w\rangle.\mathbf{0}$ is "optimised" as $\overline{v}\langle y, p\rangle.\mathbf{0}$ in $\mathcal{P}_1$ and $\mathcal{P}_2$; further, in $\mathcal{P}_2$ a similar optimisation occurs in the rule for variable. .

Each of the above differences brings up interesting issues, which we are going to analise in the order. We look first at the point (1), and we consider its effect on $\mathcal{P}_1$ and $\mathcal{P}_2$. We call $\mathcal{P}'_i$, $i = 1,2$ the encoding obtained from $\mathcal{P}_i$ by commuting the actions $r(y)$ and $\boldsymbol{\nu}\, v\, \overline{x}\langle v\rangle$ in the application rule. The action rearrangement in question is sound if the terms M and N of the application are closed; i.e. if $M, N \in \Lambda$ we have (we underline the exchanged actions):

$$\begin{aligned}\mathcal{P}_i[\![MN]\!] &\stackrel{def}{=} (p)(\boldsymbol{\nu}\, q, r)\big(\mathcal{P}_i[\![M]\!]\langle q\rangle \mid \mathcal{P}_i[\![N]\!]\langle r\rangle \mid q(x).\underline{(\boldsymbol{\nu}\, v)\overline{x}\langle v\rangle.r(y)}.\overline{v}\langle y, p\rangle.\mathbf{0}\big)\\ &\approx^{c} (p)(\boldsymbol{\nu}\, q, r)\big(\mathcal{P}_i[\![M]\!]\langle q\rangle \mid \mathcal{P}_i[\![N]\!]\langle r\rangle \mid q(x).\underline{r(y).(\boldsymbol{\nu}\, v)\overline{x}\langle v\rangle}.\overline{v}\langle y, p\rangle.\mathbf{0}\big)\end{aligned}$$

Intuitively, the equivalence holds because the commutation does not affect the order of actions which interact with the same component $\mathcal{P}_i[\![M]\!]$ or $\mathcal{P}_i[\![N]\!]$. This result has an immediate consequence, due to the fact that the λ-strategy we are encoding only allows reductions between closed terms: The operational correspondence between λ and π-terms showed by Milner for the encodings $\mathcal{P}_i$'s, extends to the $\mathcal{P}'_i$'s as well.

However, the difference between $\mathcal{P}_i$ and $\mathcal{P}'_i$ does show up when the terms M and N of the application are open. The encodings $\mathcal{P}'_i$'s appear closer than the $\mathcal{P}_i$'s to the call-by-value intuition. We justify this affirmation with an example. Consider the λ-term $\lambda x.x\Omega$: Since the call-by-value application $x\Omega$ has a divergent argument Ω, the term $x\Omega$ is supposed not to produce any visible behaviour. Consequently, we expect that a faithful encoding of call-by-value equates $\lambda x.x\Omega$ and $\lambda x.\Omega$. But this is true only for the encodings $\mathcal{P}'_i$'s, whereas it fails for the $\mathcal{P}_i$'s.

Lemma 6.4.2 *For $i = 1,2$, we have:*

- $\mathcal{P}'_i[\![\lambda x.x\Omega]\!] \approx^{c} \mathcal{P}'_i[\![\lambda x.\Omega]\!]$,
- $\mathcal{P}_i[\![\lambda x.x\Omega]\!] \not\approx^{c} \mathcal{P}_i[\![\lambda x.\Omega]\!]$

PROOF: (Sketch)It is enough to examine the encodings of $x\Omega$ and Ω. The process $\mathcal{P}_i[\![\Omega]\!]\langle p\rangle$ and $\mathcal{P}'_i[\![\Omega]\!]\langle p\rangle$ do not interact with the environment, hence

$$\mathcal{P}_i[\![\Omega]\!]\langle p\rangle \approx^{\mathrm{c}} \mathcal{P}'_i[\![\Omega]\!]\langle p\rangle \approx^{\mathrm{c}} \mathbf{0} \tag{6.1}$$

Similarly, it holds that $\mathcal{P}'_i[\![x\Omega]\!] \approx^{\mathrm{c}} \mathbf{0}$. We show this for $\mathcal{P}'_1[\![x\Omega]\!]$. The transformations below are obtained applying the expansion law of Section 6.2.2, the law (6.1) and a few simple structural congruence rules. We have:

$$\begin{aligned}\mathcal{P}'_1[\![x\Omega]\!] &\stackrel{def}{=} (\boldsymbol{\nu}\, q,r\,)\big(\boldsymbol{\nu}\, z\,\overline{q}\langle z\rangle.!z(w).\overline{x}\langle w\rangle \mid \mathcal{P}'_1[\![\Omega]\!](r) \mid q(z).r(y).\boldsymbol{\nu}\, v\,\overline{z}\langle v\rangle.\overline{v}\langle y,p\rangle\big)\\ &\approx^{\mathrm{c}} (\boldsymbol{\nu}\, r,z\,)\big(!z(w).\overline{x}\langle w\rangle \mid \mathbf{0} \mid r(y).\boldsymbol{\nu}\, v\,\overline{z}\langle v\rangle.\overline{v}\langle y,p\rangle\big) \approx^{\mathrm{c}} \mathbf{0}\end{aligned}$$

where the last equality holds because the term on the left is deadlocked and hence equal to $\mathbf{0}$. In contrast, we can show that $\mathcal{P}_1[\![x\Omega]\!]$ does exhibit a visible behaviour:

$$\begin{aligned}\mathcal{P}_1[\![x\Omega]\!] &\stackrel{def}{=} (\boldsymbol{\nu}\, q,r\,)\big(\boldsymbol{\nu}\, z\,\overline{q}\langle z\rangle.!z(w).\overline{x}\langle w\rangle \mid \mathcal{P}_1[\![\Omega]\!](r) \mid q(z).\boldsymbol{\nu}\, v\,\overline{z}\langle v\rangle.r(y).\overline{v}\langle y,p\rangle\big)\\ &\approx^{\mathrm{c}} (\boldsymbol{\nu}\, r,z\,)\big(!z(w).\overline{x}\langle w\rangle \mid \mathbf{0} \mid \boldsymbol{\nu}\, v\,\overline{z}\langle v\rangle.r(y).\overline{v}\langle y,p\rangle\big)\\ &\approx^{\mathrm{c}} (\boldsymbol{\nu}\, r,z,v\,)\big(\overline{x}\langle v\rangle \mid !z(w).\overline{x}\langle w\rangle \mid \mathbf{0} \mid r(y).\overline{v}\langle y,p\rangle\big) \quad \stackrel{\boldsymbol{\nu}\, v\,\overline{x}\langle v\rangle}{\longrightarrow}\end{aligned}$$

Similarly,

$$\begin{aligned}\mathcal{P}_2[\![x\Omega]\!] &\stackrel{def}{=} (\boldsymbol{\nu}\, q,r\,)\big(\overline{q}\langle x\rangle \mid \mathcal{P}_1[\![\Omega]\!](r) \mid q(z).\boldsymbol{\nu}\, v\,\overline{z}\langle v\rangle.r(y).\overline{v}\langle y,p\rangle\big)\\ &\approx^{\mathrm{c}} (\boldsymbol{\nu}\, r\,)\big(\mathbf{0} \mid \mathbf{0} \mid \boldsymbol{\nu}\, v\,\overline{x}\langle v\rangle.r(y).\overline{v}\langle y,p\rangle\big) \quad \stackrel{\boldsymbol{\nu}\, v\,\overline{x}\langle v\rangle}{\longrightarrow}\end{aligned}$$

Now we turn to the examination of the second difference between the encoding $\mathcal{C} \circ \mathcal{H}$ and Milner's $\mathcal{P}_i$'s (everything we shall say for the $\mathcal{P}_i$'s holds for their "rectified" $\mathcal{P}'_i$'s). We remind that the difference is the optimisations of

$$\boldsymbol{\nu}\, m\,\overline{v}\langle m,p\rangle.!\,m(w).\overline{y}\langle w\rangle.\mathbf{0} \quad \text{with} \quad \overline{v}\langle y,p\rangle.\mathbf{0}$$

in the application rule and, only for $\mathcal{P}_2$, the similar optimisation of

$$(\boldsymbol{\nu}\, y)\overline{p}\langle y\rangle.!\,y(w).\overline{x}\langle w\rangle.\mathbf{0} \quad \text{with} \quad \overline{p}\langle x\rangle.\mathbf{0}$$

in the variable rule. We call the former *optimisation 1* and the latter *optimisation 2*. Both of them can be recovered in $\mathcal{C} \circ \mathcal{H}$ by adopting the optimisation of $\mathcal{C}$ discussed in Section 5.2, namely the replacement of the rule

$$\mathcal{C}[\![\overline{a}\langle X\rangle.\mathbf{0}]\!] \stackrel{def}{=} \boldsymbol{\nu}\, m\left(\overline{a}\langle m\rangle.\mathbf{0} \mid !\, m(y).\overline{x}\langle y\rangle\right)$$

where X is a variable representing a first-order abstraction, with the simpler

$$\mathcal{C}[\![\overline{a}\langle X\rangle.\mathbf{0}]\!] \stackrel{def}{=} \overline{a}\langle x\rangle.\mathbf{0} \qquad (*)$$

We showed in Section 5.2 that rule $(*)$ is not sound in general — it can break the full abstraction for $\mathcal{C}$. However $(*)$ is sound in some cases, and we believe that the mentioned optimisation 1 falls in this category. This would give us a factorisation for $\mathcal{P}_1$ (or better, for its rectified $\mathcal{P}_1'$) through the HOπ encoding and the compilation $\mathcal{C}$, up-to some code-optimisation. We defer the analysis of the soundness of optimisation 1, as well as of possible other optimisations of $\mathcal{C}$, for future research.

On the contrary, the use of $(*)$ in optimisation 2 is not sound. We show this by proving that β-conversion is not valid for $\mathcal{P}_2$ — whereas it is valid in $\mathcal{C} \circ \mathcal{H}$ by Lemma 6.4.1 and the full abstraction of $\mathcal{C}$.

Lemma 6.4.3 *There are β-convertible terms M and N s.t. $\mathcal{P}_2[\![M]\!] \not\approx^{\mathrm{c}} \mathcal{P}_2[\![N]\!]$.*

PROOF: Take $M = \big(\lambda x.(\lambda y.x)\big)(\lambda z.z)$ and $N = \lambda y.(\lambda z.z)$. With a β-conversion, M reduces to N. However, $\mathcal{P}_2[\![M]\!] \not\approx^{\mathrm{c}} \mathcal{P}_2[\![N]\!]$, as we are going to prove. We exploit the characterisation of $\approx^{\mathrm{c}}$ in terms of weak early congruence in π-calculus (Corollary 3.3.4). Since weak early congruence is the congruence induced by weak early bisimulation ($\approx_{\mathrm{e}}$), it is enough to show that $\mathcal{P}_2[\![M]\!] \not\approx_{\mathrm{e}} \mathcal{P}_2[\![N]\!]$.

Let $R_1 \stackrel{def}{=} x'(w').w'(y', r').\overline{r'}\langle y'\rangle$. By repeated use of the expansion law and simple application of the structural congruence relation, we get

$$\mathcal{P}_2[\![M]\!]\langle p\rangle \;\approx_{\mathrm{e}}\; (\boldsymbol{\nu}\, x, x')\big(\overline{p}\langle x\rangle.!\, x(w).w(y,r).\overline{r}\langle x'\rangle \mid !\, R_1\big) \stackrel{def}{=} P$$

On the other hand, we have

$$\mathcal{P}_2[\![N]\!]\langle p\rangle \;=\; (\boldsymbol{\nu}\, x)\overline{p}\langle x\rangle.!\, x(w).w(y,r).(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.!\, R_1 \stackrel{def}{=} Q$$

We have to show that $P \not\approx_{\mathrm{e}} Q$. It is the different position of the restriction $\boldsymbol{\nu}\, x'$ which causes the difference between P and Q. Consider the sequence below

of transitions from P. Here R_2 stands for $x(w).w(y,r).\overline{r}\langle x'\rangle$ and the underlined action or component is the one causing the successive action.

$$\begin{array}{lll}
P & \xrightarrow{(\nu\, x)\overline{p}\langle x\rangle} & \boldsymbol{\nu}\, x'\left(\,!\,\underline{R_2}\mid\,!\,R_1\right)\\
& \xrightarrow{x\langle w\rangle} & \boldsymbol{\nu}\, x'\left(\,!\,R_2\mid\underline{w(y,r).\overline{r}\langle x'\rangle}\mid\,!\,R_1\right)\\
& \xrightarrow{w\langle y,r\rangle} & \boldsymbol{\nu}\, x'\left(\,!\,R_2\mid\underline{\overline{r}\langle x'\rangle}\mid\,!\,R_1\right)\\
& \xrightarrow{(\nu\, x')\overline{r}\langle x'\rangle} & !\,\underline{R_2}\mid\,!\,R_1\\
& \xrightarrow{x\langle w''\rangle} & !\,R_2\mid\underline{w''(y,r).\overline{r}\langle x'\rangle}\mid\,!\,R_1\\
& \xrightarrow{w''\langle y,r\rangle} & !\,R_2\mid\underline{\overline{r}\langle x'\rangle}\mid\,!\,R_1\\
& \xrightarrow{\overline{r}\langle x'\rangle} & !\,R_2\mid\,!\,R_1
\end{array}$$

In this sequence, the last action is a *free* output. Instead, in the corresponding sequence for Q, the last action is a ***bound*** output. Below, R_3 stands for $x(w).w(y,r).(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1$.

$$\begin{array}{lll}
Q & \xrightarrow{(\nu\, x)\overline{p}\langle x\rangle} & !\,\underline{R_3}\\
& \xrightarrow{x\langle w\rangle} & !\,R_3\mid\underline{w(y,r).}(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1\\
& \xrightarrow{w\langle y,r\rangle} & !\,R_3\mid\underline{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle}.\,!\,R_1\\
& \xrightarrow{(\nu\, x')\overline{r}\langle x'\rangle} & !\,\underline{R_3}\mid\,!\,R_1\\
& \xrightarrow{x\langle w''\rangle} & !\,R_3\mid\underline{w''(y,r).}(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle.\,!\,R_1\mid\,!\,R_1\\
& \xrightarrow{w''\langle y,r\rangle} & !\,R_3\mid\underline{(\boldsymbol{\nu}\, x')\overline{r}\langle x'\rangle}.\,!\,R_1\mid\,!\,R_1\\
& \xrightarrow{(\nu\, x')\overline{r}\langle x'\rangle}\equiv & !\,R_3\mid\,!\,R_1
\end{array}$$

So, we have found a deficiency in $\mathcal{P}_2$. Nevertheless it is true that $\mathcal{P}_2$ yields a precise operational correspondence between λ-terms and their process encodings, and intuitively one "expects" $\mathcal{P}_2$ to be correct. We leave for future research how to best formalise such a perception.[1] An alternative question is whether there is

[1] By the time the review of this thesis was finished, a positive answer to this question has been obtained by Pierce and Sangiorgi in the paper "Typing and subtyping for mobile processes", to appear in the proceedings of Eigth Annual IEEE Symposium on Logics in Computer Science (LICS'93), Montreal, Canada, June 1993.

an encoding as efficient as $\mathcal{P}_2$ but which does not have the above deficiency.

6.5 A λ-model from process terms

Having shown that there is an exact operational correspondence between lazy λ-terms and their HOπ encodings (Proposition 6.3.6), it is legitimate to ask ourselves whether the lazy encoding $\mathcal{H}$ gives rise to a λ-model, and if yes, what kind of λ-model it represents.

We chose the lazy λ-calculus for this study because of its simplicity. For the time being, we prefer to leave the corresponding analysis for call-by-value as future work, together with the question whether a simpler encoding exists. We conduct our study in the HOπ; by Theorem 6.3.7 and full abstraction of the compilation $\mathcal{C}$, all results obtained can be transported to the π-calculus encoding. From now on up to the end of the chapter, the word encoding and the symbol $\mathcal{H}$ refer to the HOπ encoding of lazy λ-calculus.

6.5.1 λ-models

There are simple syntax-free definitions of λ-model (i.e. they do not mention λ-terms). But since we already have the mapping from λ to process terms, it is more convenient a definition where we can use such a mapping explicitly. A *valuation* is a function from the set of λ-variables to the domain D of the λ-model; $[\boldsymbol{d}/x]\rho$ is the valuation which maps x to $\boldsymbol{d}$ and which behaves like ρ on the remaining elements.

Definition 6.5.1 *A* λ-model *is a triple* $< D, \cdot, \mathcal{M} >$, *where* D *is a set with at least two elements,* '$\cdot$' *is a mapping from* $D \times D$ *to* D *and* $\mathcal{M}$ *is a mapping which assigns, to each* λ*-term* M *and evaluation* ρ, *a member* $\mathcal{M}[\![M]\!]_\rho \in D$ *s.t. :*

1. $\mathcal{M}[\![x]\!]_\rho = \rho(x)$
2. $\mathcal{M}[\![MN]\!]_\rho = \mathcal{M}[\![M]\!]_\rho \cdot \mathcal{M}[\![N]\!]_\rho$

3. $\mathcal{M}[\![\lambda x.M]\!]_\rho \cdot \boldsymbol{d} = \mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho}$, $\forall \boldsymbol{d} \in D$

4. $\mathcal{M}[\![M]\!]_\rho = \mathcal{M}[\![M]\!]_\sigma$ *if* $\rho(x) = \sigma(x)\ \forall x \in fv(M)$

5. $\mathcal{M}[\![\lambda x.M]\!]_\rho = \mathcal{M}[\![\lambda y.M\{y/x\}]\!]_\rho$, *if* $y \notin fv(M)$

6. *if* $\forall \boldsymbol{d} \in D$, $\mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho} = \mathcal{M}[\![N]\!]_{[\boldsymbol{d}/x]\rho}$, *then* $\mathcal{M}[\![\lambda x.M]\!]_\rho = \mathcal{M}[\![\lambda x.N]\!]_\rho$.

6.5.2 The model

Our definition λ-model should respect the semantic relation adopted in HOπ. So, let us denote by $[A]_{\approx^c}$ the equivalence class of the agent A, namely

$$\{A' \ : \ A' \text{ is an HO}\pi \text{ agent and } A \approx^c A'\}$$

The elements of the domain D of the model will be the equivalence classes of the closed HOπ agents with the same sort (s) as the agents encoding λ-terms.

$$D \stackrel{def}{=} \{[F]_{\approx^c} \ : \ F \in \text{HO}\pi \text{ and } F : (s)\}$$

The definition of application on these elements follow the translation of λ-application in $\mathcal{H}$:

$$[G]_{\approx^c} \cdot [F]_{\approx^c} \stackrel{def}{=} [(p)\boldsymbol{\nu}\, q\, (G\langle q\rangle \mid \overline{q}\langle F, p\rangle)]_{\approx^c} \quad \text{for } p, q \text{ not free in } G, F$$

Note that the definition of application is consistent: by the congruence properties of $\approx^c$, the result of the application does not depend upon the representatives G and F chosen from the equivalence classes. We are left with the definition of the mapping $\mathcal{M}[\![M]\!]_\rho$. The evaluation ρ maps λ-variables to equivalence classes of $\approx^c$. Given an evaluation ρ, we denote by ρ^H a substitution from HOπ variables to HOπ agents s.t.

$$\text{for each } \lambda\text{-variable } x, \quad \rho^H(\mathcal{H}[\![x]\!]) \in \rho(x)$$

Therefore, ρ^H is the "conversion" of ρ which operates on the HOπ variable $\mathcal{H}[\![x]\!]$ and which selects a representative out of the equivalence class of $\rho(x)$. Now, the mapping $\mathcal{M}$ of the λ-model is defined using $\mathcal{H}$ as follows:

$$\mathcal{M}[\![M]\!]_\rho \stackrel{def}{=} [\,\mathcal{H}[\![M]\!]\rho^H\,]_{\approx^c}$$

Note that since $\approx^c$ is a congruence, this definition is independent from the representatives of the equivalence classes selected by ρ^H. We denote by $\mathcal{D}$ be the triple $< D, \cdot, \mathcal{M} >$ so obtained.

Theorem 6.5.2 *$\mathcal{D}$ is a λ-model.*

PROOF: We consider each of the clauses (1-6), in the order.

(1) We have:

$$\mathcal{M}[\![x]\!]_\rho \stackrel{def}{=} [\,\mathcal{H}[\![x]\!]\rho^H\,]_{\approx^c} = [\,\rho^H(\mathcal{H}[\![x]\!])\,]_{\approx^c} = \rho(x)$$

where the last equality holds because, by definition, $\rho^H(\mathcal{H}[\![x]\!])$ belongs to the equivalence class $\rho(x)$.

(2) We have:

$$\begin{aligned}
\mathcal{M}[\![MN]\!]_\rho &\stackrel{def}{=} \\
[\,\mathcal{H}[\![MN]\!]\rho^H\,]_{\approx^c} &= \text{for } p, q \text{ fresh} \\
[\,(p)\boldsymbol{\nu}\, q\,\big(\mathcal{H}[\![M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle\mathcal{H}[\![N]\!]\rho^H, p\rangle\big)\,]_{\approx^c} &= \\
[\,\mathcal{H}[\![M]\!]\rho^H\,]_{\approx^c} \cdot [\,\mathcal{H}[\![N]\!]\rho^H\,]_{\approx^c} &\stackrel{def}{=} \mathcal{M}[\![M]\!]_\rho \cdot \mathcal{M}[\![N]\!]_\rho
\end{aligned}$$

(3) Let $\boldsymbol{d} = [\,F\,]_{\approx^c}$. We have to show that for p and q fresh

$$\mathcal{M}[\![\lambda x.M]\!]_\rho \cdot \boldsymbol{d} = [\,(p)\boldsymbol{\nu}\, q\,\big(\mathcal{H}[\![\lambda x.M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle F, p\rangle\big)\,]_{\approx^c}$$

is the same equivalence class as

$$\mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho} \stackrel{def}{=} [\,\mathcal{H}[\![M]\!]([\boldsymbol{d}/x]\rho)^H\,]_{\approx^c} = [\,\mathcal{H}[\![M]\!]\{F/X\}\rho^H\,]_{\approx^c}$$

For this, we show that for all p, with $p \neq q$,

$$\boldsymbol{\nu}\, q\,\big(\mathcal{H}[\![\lambda x.M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle F, p\rangle\big) \approx^c \big(\mathcal{H}[\![M]\!]\{F/X\}\rho^H\big)\langle p\rangle$$

By α-conversion we can assume that the bound variable X is not modified by ρ^H and then we have:

$$\begin{aligned}
\boldsymbol{\nu}\, q\left(\mathcal{H}[\![\lambda x.M]\!]\rho^H\langle q\rangle \mid \overline{q}\langle F, p\rangle\right) &= \\
\boldsymbol{\nu}\, q\left(q(X,r).\mathcal{H}[\![M]\!]\rho^H\langle r\rangle \mid \overline{q}\langle F, p\rangle\right) &\approx^{\mathrm{c}} \quad \text{(expansion law)} \\
\boldsymbol{\nu}\, q\left(\mathcal{H}[\![M]\!]\{F/X\}\rho^H\langle p\rangle\right) &\equiv \quad (q \text{ not free in the body}) \\
\left(\mathcal{H}[\![M]\!]\{F/X\}\rho^H\right)\langle p\rangle &
\end{aligned}$$

(4) If $\rho(x) = \sigma(x)$, then $\rho^H(X) \approx^{\mathrm{c}} \sigma^H(X)$; since this holds for all x and $\approx^{\mathrm{c}}$ is a congruence,

$$\mathcal{H}[\![M]\!]\rho^H \approx^{\mathrm{c}} \mathcal{H}[\![M]\!]\sigma^H$$

From this, we get

$$\mathcal{M}[\![M]\!]_\rho \stackrel{def}{=} [\,\mathcal{H}[\![M]\!]\rho^H\,]_{\approx^{\mathrm{c}}} = [\,\mathcal{H}[\![M]\!]\sigma^H\,]_{\approx^{\mathrm{c}}} \stackrel{def}{=} \mathcal{M}[\![M]\!]_\sigma$$

(5) An α-conversion on λ-terms is reflected on the α-conversion on their HOπ encodings.

(6) By hypothesis, for all F, if $\boldsymbol{d} = [\,F\,]_{\approx^{\mathrm{c}}}$, we have:

$$\mathcal{M}[\![M]\!]_{[\boldsymbol{d}/x]\rho} = \mathcal{M}[\![N]\!]_{[\boldsymbol{d}/x]\rho}$$

Then from the definition of $\mathcal{M}$, we infer that for all F,

$$\mathcal{H}[\![M]\!]\{F/X\}\rho^H \approx^{\mathrm{c}} \mathcal{H}[\![N]\!]\{F/X\}\rho^H$$

which means that

$$\mathcal{H}[\![M]\!]\rho^H \approx^{\mathrm{c}} \mathcal{H}[\![N]\!]\rho^H$$

Since $\approx^{\mathrm{c}}$ is a congruence, we get

$$(p)p(X,q).\mathcal{H}[\![M]\!]\rho^H\langle q\rangle \approx^{\mathrm{c}} (p)p(X,q).\mathcal{H}[\![N]\!]\rho^H\langle q\rangle$$

which, by definition of $\mathcal{M}$, gives

$$\mathcal{M}[\![\lambda x.M]\!]_\rho = \mathcal{M}[\![\lambda x.N]\!]_\rho$$

Remark 6.5.3 One could have tried to be more selective in the definition of the domain $\mathcal{D}$, and take as domain $D^* = \{[\mathcal{H}[\![M]\!]]_{\approx^c} : M \in \Lambda\}$; then $\mathcal{D}^* =< D^*, \cdot, \mathcal{M} >$ represents the *interior* of $\mathcal{D}$ (see [HS86, pag.107]). But it turns out that $\mathcal{D}^*$ is *not* a λ-model: Clause (6) in Definition 6.5.1 fails. As counterexample, take the terms L_1 and L_2 as will be defined in Section 6.6. Following the proof in [AO89] that that $L_1 \cong L_2$, one can show that for all closed N, $\mathcal{H}[\![L_1\{N/x\}]\!] \approx^c \mathcal{H}[\![L_2\{N/x\}]\!]$; this means that

$$\text{for all } \boldsymbol{d} \in D^* \quad \mathcal{M}[\![L_1]\!]_{[d/x]\rho} = \mathcal{M}[\![L_2]\!]_{[d/x]\rho}$$

However,

$$\mathcal{M}[\![\lambda x.L_1]\!]_\rho \not\approx^c \mathcal{M}[\![\lambda x.L_2]\!]_\rho$$

which can be derived from the fact that $\mathcal{H}[\![L_1]\!] \not\approx^c \mathcal{H}[\![L_2]\!]$ (the difference between them appears when instantiating their free variable with an agent which is not the encoding of a λ-term, as discussed in Section 6.6). Therefore $\mathcal{D}$ is a λ-model whose interior is not a λ-model. See [HL80] for two more examples. □

Having proved that $\mathcal{D}$ is a λ-model, we can infer for it all properties of λ-models. In particular, we get that:

- Every provable equation of $\lambda\beta$ is valid for the encoding, up-to $\approx^c$ (where $\lambda\beta$ is the formal theory given by α, β-conversion plus the rules of inference for equivalence and congruence).

- $< D, \cdot >$ is a combinatory algebra (and hence is combinatorially complete) where the two distinguished elements $\boldsymbol{k}$ and $\boldsymbol{s}$ can be defined as $\boldsymbol{k} \stackrel{def}{=} \mathcal{H}[\![\lambda xy.x]\!]$, and $\boldsymbol{s} \stackrel{def}{=} \mathcal{H}[\![\lambda xyz.xz(yz)]\!]$.

We now turn to examining the questions of extensionality and full abstraction for $\mathcal{D}$.

6.5.3 Extensionality

Model D is *not* extensional, i.e. it is not a $\lambda\eta$ model. As counterexample, take Ω and $\lambda x.\Omega x$. Then we have $\mathcal{H}[\![\Omega]\!]\langle p\rangle \not\approx^c \mathcal{H}[\![\lambda x.\Omega x]\!]\langle p\rangle$, since $\mathcal{H}[\![\Omega]\!]\langle p\rangle \approx^c \mathbf{0}$, whereas $\mathcal{H}[\![\lambda x.\Omega x]\!]\langle p\rangle$ can perform a visible action at p.

This failure is not too surprising, since our encoding mimics the lazy λ-calculus, in which the η-rule is not valid. However, in the lazy λ-calculus a weak form of η-rule holds, namely

$$M \Downarrow \text{ implies } \lambda x.Mx \cong M, \text{ for } x \notin fv(M).$$

The same weak η-rule holds for our encoding.

Theorem 6.5.4 (weak extensionality) *$M \Downarrow$ implies $\mathcal{H}[\![\lambda x.Mx]\!] \approx^c \mathcal{H}[\![M]\!]$, for $x \notin fv(M)$.*

PROOF: By exploiting the characterisation of $\approx^c$ in terms of weak normal congruence: For any p, $\mathcal{H}[\![\lambda x.Mx]\!]\langle p\rangle$ and $\mathcal{H}[\![M]\!]\langle p\rangle$ are processes willing to perform an input at p. Using Proposition 6.3.6 and the definition of the encoding, one can show that for each input provided by the environment, $\mathcal{H}[\![\lambda x.Mx]\!]\langle p\rangle$ and $\mathcal{H}[\![M]\!]\langle p\rangle$ evolve to equivalent derivations. □

Remark 6.5.5 Only as a remark, and without reporting the proofs of our claims, let us mention *λ-transitions systems (lts)*: They are introduced in Section 6.3 of [Abr87] as general operational models of the lazy λ-calculus. Then Abramsky shows that every *lts*:

- is also a λ-model
- satisfies weak extensionality.

If we had proved that our model $\mathcal{D}$ is an *lts*, then we could have inferred directly the results in Sections 6.5.2 and 6.5.3. However this is *not* the case (and neither its

interior $\mathcal{D}'$ is an *lts*). The reason is that the equivalence which has to be defined on an *lts* (see [Abr87, Definition 6.3.2]) would not be compatible with $\approx^c$; in other words, in order to obtain an *lts* we would have to quotient the HOπ agents with a relation which does not seem to be particularly appealing in a process calculus setting. □

6.6 Full abstraction

The full abstraction problem was first studied by Milner [Mil77] and Plotkin [Plo77]. It is concerned with the problem of finding a denotational interpretation for a programming language s.t. the resulting semantic equality coincides with a notion of operational indistinguishableness. Simply typed λ-calculus is the classical setting in which the problem has been developed. With the introduction of the operational equivalence resulting from applicative bisimulation, it can be neatly transferred to the untyped λ-calculus and it has motivated elegant works by Abramsky, Ong, Boudol ([AO89,Bou91]), in the setting of lazy λ-calculus.

A denotational interpretation is said to be

- *sound* if it only equates operationally equivalent terms,
- *complete* if it equates all operational equivalent terms,
- *fully abstract* if it is sound and complete.

Let us consider what happens with the encoding $\mathcal{H}$ of lazy λ-calculus into HOπ. It is *sound*, since it is true that $\mathcal{H}[\![M]\!] \approx^c \mathcal{H}[\![N]\!]$ implies $M \cong N$. This can be established using Proposition 6.3.6 and the characterisation of $\approx^c$ in terms of weak context congruence. However, $\mathcal{H}$ is not *complete*. To prove this, consider the terms

$$L_1 = x(\lambda y.(x\Xi\Omega y))\Xi \qquad L_2 = x(x\Xi\Omega)\Xi$$

where Ξ is an always convergent term (that is, for all $\widetilde{N}$, it holds that $\Xi\widetilde{N} \Downarrow$), like $(\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx))$. Terms L_1, L_2 are used by Abramsky and Ong [AO89] to show that their canonical model for lazy λ-calculus is not fully abstract. They show that $L_1 \cong L_2$ by induction on the order of the term which instantiates x. On the other hand, they show that L_1 and L_2 can be distinguished using a new combinator, called *convergence test* and denoted by ∇, which is definable in the canonical model but is not in lazy λ-calculus. The operational behaviour of ∇ is given by

$$(\nabla 1)\ \frac{M \Downarrow}{\nabla M \Longrightarrow I} \qquad\qquad (\nabla 2)\ \frac{M \Longrightarrow M'}{\nabla M \Longrightarrow \nabla M'}$$

Using ∇, we have $L_1\{\nabla/x\} \Downarrow$, whereas $L_2\{\nabla/x\} \not\Downarrow$. We also have $\mathcal{H}[\![L_1]\!] \not\approx^c \mathcal{H}[\![L_2]\!]$. This follows from the corresponding result for the encoding into π-calculus which Milner obtains by implementing convergence test as a π-process, and Theorem 6.3.7. In terms of our model $\mathcal{D}$ this means that L_1 and L_2 have different denotations and that $\mathcal{D}$ is not a fully abstract model for the lazy λ-calculus.

Given a denotational interpretation which is not fully abstract, there are two natural directions to achieve full abstraction.

- To cut down the existing "over-generous" semantics domain (*restrictive approach*).
- To enrich the language (*expansive approach*).

These two approaches are exemplified by the solutions to the full abstraction problem for PCF (a typed λ-calculus extended with fixed points, boolean and arithmetic features) proposed by Milner [Mil77] and Plotkin [Plo77]; in the latter, PCF is augmented with a parallel 'or' operator. We shall see that in the case of our encoding $\mathcal{H}$ both these approaches lead to interesting constructions.

6.6.1 The restrictive approach

Such approach is based on the use of barbed bisimulation. More specifically, we exploit the possibility of parametrising this equivalence on a specific class of contexts. As λ-terms are *only* used in λ-calculus contexts, so we require that their encodings be used *only* in encodings of λ-contexts. The encoding $\mathcal{H}$ is extended to λ-contexts by mapping the λ hole to the HOπ hole, i.e. $\mathcal{H}[\![[\cdot]]\!] \stackrel{def}{=} [\cdot]$. Thus, the class of contexts we are interested in is

$$\mathcal{L} = \{\mathcal{H}[\![C_\lambda[\cdot]]\!] \ s.t. \ C_\lambda[\cdot] \text{ is a closed } \lambda\text{-context}\}$$

If M and N are closed λ-terms, we set $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ if for every $C[\cdot] \in \mathcal{L}$ and for every p, it holds that $C[\mathcal{H}[\![M]\!]\langle p\rangle \stackrel{\cdot}{\approx} C[\mathcal{H}[\![N]\!]\langle p\rangle$ (i.e. they are weak barbed bisimilar). The definition of $\approx_{\mathcal{L}}$ is extended to encodings of open λ-terms as expected: If $fv(\mathcal{H}[\![M]\!], \mathcal{H}[\![N]\!]) = \{X_1, \ldots, X_n\}$, then $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ if for all closed λ-terms $L_1, \ldots, L_n$, it holds that

$$\mathcal{H}[\![M]\!]\{\mathcal{H}[\![L_1]\!]/X_1, \ldots, \mathcal{H}[\![L_n]\!]/X_n\} \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]\{\mathcal{H}[\![L_1]\!]/X_1, \ldots, \mathcal{H}[\![L_n]\!]/X_n\}.$$

We prove that $M \cong N$ iff $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ via the characterisation of $\cong$ given in Theorem 6.3.4. This characterisation used the relations $\stackrel{\cdot}{\approx}$ and $\approx^c$, which are the specialisation to the λ-calculus of barbed bisimulation and congruence and were introduced in Definition 6.3.3.

Lemma 6.6.1 *Let $M \in \Lambda$: then for each p, $M \Downarrow$ iff $\mathcal{H}[\![M]\!]\langle p\rangle \Downarrow$.*

PROOF: Use Proposition 6.3.6. □

Lemma 6.6.2 *For all $M, N \in \Lambda$ and for each name p, it holds that $M \stackrel{\cdot}{\approx} N$ iff $\mathcal{H}[\![M]\!]\langle p\rangle \stackrel{\cdot}{\approx} \mathcal{H}[\![N]\!]\langle p\rangle$.*

PROOF: Use Lemma 6.6.1 together with the operational correspondence shown in Proposition 6.3.6. □

Proposition 6.6.3 *If $M, N \in \Lambda^o$, then $M \cong N$ iff $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$.*

PROOF: It is enough to prove the result for closed terms. By Theorem 6.3.4, $M \cong N$ iff for each closed λ-context $C_\lambda[\cdot]$, $C_\lambda[M] \stackrel{\cdot}{\approx} C_\lambda[N]$. Similarly, by definition of $\approx_{\mathcal{L}}$, $\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!]$ iff for each context $C[\cdot]$ encoding of a closed λ-context $C_\lambda[\cdot]$, and for each name p, it holds that

$$C[\mathcal{H}[\![M]\!]\langle p\rangle = \mathcal{H}[\![C_\lambda[M]]\!]\langle p\rangle \stackrel{\cdot}{\approx} \mathcal{H}[\![C_\lambda[N]]\!]\langle p\rangle = C[\mathcal{H}[\![N]\!]\langle p\rangle.$$

Then the assertion of the proposition follows from Lemma 6.6.2. □

Lemma 6.6.4 ($\approx_{\mathcal{L}}$ is preserved by encodings of λ-contexts) *Suppose that $C[\cdot] = \mathcal{H}[\![C_\lambda[\cdot]]\!]$, for some (possibly open) λ-context $C_\lambda[\cdot]$, and that $M, N \in \Lambda^\circ$. Then*

$$\mathcal{H}[\![M]\!] \approx_{\mathcal{L}} \mathcal{H}[\![N]\!] \textit{ implies } C[\mathcal{H}[\![M]\!]] \approx_{\mathcal{L}} C[\mathcal{H}[\![N]\!]].$$

PROOF: We have $C[\mathcal{H}[\![M]\!]] = \mathcal{H}[\![C_\lambda[M]]\!]$ and $C[\mathcal{H}[\![N]\!]] = \mathcal{H}[\![C_\lambda[N]]\!]$. Then the result of the lemma follows from the analogous result for λ-terms in Corollary 6.3.5 and from the correspondence between $\cong$ on λ-terms and $\approx_{\mathcal{L}}$ on their HOπ encodings in Proposition 6.6.3. □

This result allows us to construct a fully abstract model for the lazy λ-calculus. Let $[A]_{\approx_{\mathcal{L}}}$ be the equivalence class of A modulo $\approx_{\mathcal{L}}$, '$\cdot$' and $\mathcal{M}[\![M]\!]$ as defined in Section 6.5.2 but with $[\,]_{\approx_{\mathcal{L}}}$ in place of $[\,]_{\approx^c}$; then take

$$\begin{aligned} D' &\stackrel{def}{=} \{\mathcal{M}[\![M]\!]_\rho \,:\, M \in \Lambda\} \\ \mathcal{D}' &\stackrel{def}{=} < D', \cdot, \mathcal{M} > \end{aligned}$$

Theorem 6.6.5 (full abstraction) *$\mathcal{D}'$ is a fully abstract model for the lazy λ-calculus.*

PROOF: Full abstraction follows from Proposition 6.6.3. The proof that $\mathcal{D}'$ is a λ-model is analogous to the proof that $\mathcal{D}$ is a λ model in Theorem 6.5.2. (The proof of Theorem 6.5.2 used the congruence properties of $\approx^c$; in this case, we need the congruence properties of $\approx_{\mathcal{L}}$ showed in Lemma 6.6.4. We also need the fact that $\approx^c$ implies $\approx_{\mathcal{L}}$). □

Indeed, the model $\mathcal{D}'$ is also *fully expressive*: all objects of the domain of interpretation are λ-definable. These results show that if from HOπ and π-calculus we discard everything which is extraneous to the encoding of the lazy λ-calculus, in particular adopting $\approx_{\mathcal{L}}$ as semantic equivalence, then the structure that we get back is the "same thing" as the lazy λ-calculus. In our view, this was really the decisive test for the correctness of these encodings.

The cut of the domain D' with respect to the domain D in Section 6.5 is in two dimensions: The behavioural equivalence is $\approx_{\mathcal{L}}$ rather than the more discriminanting $\approx^{c}$; and only the interior of D is taken into account. The first restriction is necessary to get full abstraction; the second to make consistent the definition of application. It is interesting to note the relationship between the choices of the classes of HOπ agents and of the behavioural equivalence in the definitions of the domains D and D'. In the former, we took the class $\mathcal{A}$ of *all* admissible agents, and then in the behavioural equivalence we had to use quantification over the class $\mathcal{C}nt$ of *all* admissible contexts (definition of $\approx^{c}$); in the latter we restrained ourselves to the *"interior"* of $\mathcal{A}$, and then in the behavioural equivalence we also had to restrain ourselves to the *"interior"* of $\mathcal{C}nt$ (definition of $\approx_{\mathcal{L}}$).

6.6.2 The expansive approach

In this section we study the equivalence induced on λ-terms by the encoding. We call it *λ-observational equivalence.*

Definition 6.6.6 (λ-observational equivalent) *We say that two Λ terms M and N are* λ-observational equivalent *if $\mathcal{H}[\![M]\!] \approx^{c} \mathcal{H}[\![N]\!]$.*

The main result will be a *direct* characterisation of λ-observational equivalence (i.e. a characterisation not mentioning the encoding). From the characterisation of $\approx$ (weak barbed equivalence) and $\approx^{c}$ (weak barbed congruence) in terms of, respectively, weak normal bisimulation and weak normal congruence (Corollaries 4.7.8 and 4.7.9), it follows that $\approx^{c}$ means "$\approx$ under all substitutions of names".

Since the HOπ encoding of a λ-term does not have names free in it, the relations $\approx^c$ and $\approx$ coincide on such terms. In view of this fact, in this section we shall use $\approx$ to study λ-observational equivalence. We shall often exploit the characterisation of $\approx$ in terms of weak context bisimulation and weak normal bisimulation in the proofs.

λ-observational equivalence

To study λ-observational equivalence means to understand the effect on λ-terms of the use of "richer" contexts, in which also concurrent features may be present. There are also motivations from the theory itself of lazy λ-calculus.

As originally defined, applicative bisimulation carries some problems. It is based on the notion of termination, but you cannot "speak" in the pure λ-calculus about termination. For instance, you cannot define the convergence test [AO89]. Such an operator can be used to show that Abramsky's canonical model for lazy λ-calculus and the model $\mathcal{D}$ of Section 6.5.2 are not fully abstract. Now, what does this mean? Maybe Abramsky's canonical domain and the encoding $\mathcal{H}$ are not good enough for the lazy λ-calculus; but maybe the problem is just that the pure λ-calculus is too weak versus the predicate of convergence. Moreover, since applicative bisimulation is based on equivalence notions originally developed for frameworks of reactive and concurrent systems, one might consider appealing the introduction of parallel operators.

Various enrichments of the lazy λ-calculus with operators not λ-definable have already appeared in the literature. However either the operators themselves — as in the case of convergence test and parallel convergence in [AO89,Ong88] — or the semantics chosen — as for the non-deterministic choice and parallel operator in [Bou90] and [Bou91] — are rather ad hoc, aimed to achieve full abstraction for some canonical domain. Or at least, from a programming language point of view, they do not seem to be justified by the common practice. Moreover it remains unclear whether the equivalences induced are insensitive to the adoption of more

operators.

λ-observational equivalence seems a *robust* equivalence. Firstly, it enjoys simple and nice operational and denotational characterisations. Secondly, it coincides with the equivalence obtained when λ-calculus is augmented with the whole class of *well-formed operators*, intuitively operators whose behaviour depends only on the semantics – not on the syntax – of its operands. Put in other words, the encoding into π-calculus/HOπ induces maximal observational discrimination on λ-terms. Further, the adoption of certain simple non-deterministic operators is sufficient and necessary to induce such a discrimination. These results are showed in [San92].

Direct characterisation

To derive a direct characterisation for λ-observational equivalence we have to enrich the λ-calculus with constants. The standard way to treat a constant is to introduce it together with some rules describing its operational behaviour. We call such symbols *operators*; an example is convergence test. When only operators are used, λ-abstraction remains the only sensible normal form for closed terms. We keep the word *constant* to denote instead symbols which are added to the language without specifying any operational rule. Such a use of the constants can be found in the well-known technique of the *top-down specification and analysis*, where a system is developed through a series of refinement steps each representing a different level of abstraction; a lower level implements some details which at a higher level have been abstracted. A constant c is then an high-level primitive standing for some lower-level procedure K_c; at this stage we might want to explicitly abstract from the behaviour of K_c to facilitate the reasoning, or it might just be that we cannot make assumptions on the behaviour of K_c (for instance, we might be interested in refinments of c with different K_c's). Now, $c\widetilde{M}$ becomes a sensible normal form too. Operationally we really can see it as the *output* of the tuple $\widetilde{M}$ along the channel c and towards K_c. The definition of applicative bisim-

ulation has been generalised according to such interpretation to the case where also constants are admitted.

Let C be a set containing an infinite, countable number of constants. The class of Λ^o_C terms, i.e. λ-terms enriched with constants in C, is defined from the grammar of Λ^o by adding constants among the constructs:

$$M = c \mid x \mid \lambda x.M \mid M_1 M_2, \qquad \text{where } c \in C.$$

When generalising applicative bisimulation to terms in Λ_C, the main question is which condition should be imposed on the equality between the terms $c\widetilde{M}$ and $c\widetilde{N}$. According to our interpretation of the constants given above, the most natural thing to do seems to require that the *ordered* sequence of arguments represented by $\widetilde{M}$ and $\widetilde{N}$ be equivalent; hence the clause (2) in the following definition.

Definition 6.6.7 *A relation $\mathcal{R} \subseteq \Lambda_C \times \Lambda_C$ is a $\cong_C$-simulation, if $(M,N) \in \mathcal{R}$ implies:*

1. *whenever $M \Longrightarrow \lambda x.M'$ then N' exists s.t. $N \Longrightarrow \lambda x.N'$ and for all $L \in \Lambda_C$ it holds that $(M'\{L/x\}, N'\{L/x\}) \in \mathcal{R}$,*

2. *whenever $M \Longrightarrow cM_1 ... M_n$, for some $n \geq 0$ and $c \in C$, then $N_1, \ldots N_n$ exist s.t. $N \Longrightarrow cN_1 \ldots N_n$ and $(M_i, N_i) \in \mathcal{R}$, $1 \leq i \leq n$.*

A relation $\mathcal{R}$ is an $\cong_C$-bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are $\cong_C$-simulations. M and N are applicative bisimilar over Λ_C, *written $M \cong_C N$, if $(M,N) \in \mathcal{R}$, for some $\cong_C$-bisimulation $\mathcal{R}$.* □

Remark 6.6.8 On pure λ-calculus we could have obtained the same equivalence as $\cong_C$ if in Definition 6.6.7 we had used variables instead of constants and accepted to work on open terms [San92]. Our choice is justified by essentially two reasons. First, variables and constants play distinct rôles in Definition 6.6.7 and therefore we think it is correct to keep them separated. Secondly, the use of constants is

convenient to get easier and more intuitive proofs of the theorems which follow. The problem using variables is that open λ-terms are encoded into open HOπ terms, which have to be closed to perform transitions: This would make rather obscure the operational correspondence between λ and HOπ terms. $\square$

EXTENSION OF THE ENCODING $\mathcal{H}$ TO Λ_C TERMS

We extend the encoding $\mathcal{H}$ to Λ_C terms and prove that $\cong_C$ is exactly the equivalence induced. We have to say what the encoding of a constant is. We use each constant $c \in C$ also as an HOπ name, whose sort is supposed to be different from those of the other names appearing in the encoding and we decree:

$$\mathcal{H}[\![c]\!] \stackrel{def}{=} Tr_c \quad \text{where } Tr_c \stackrel{def}{=} (p)\overline{c}\langle p\rangle.$$

Agents like Tr_c were introduced in Section 4.3.1 and called triggers. The results in Chapter 4 show the discriminating power given by triggers. It turns out that this corresponds exactly to the discriminating power that constants provide on λ-terms.

We shall adopt the following abbreviations, already used in various occasions in the previous chapters; here F represents a first-order abstraction:

$$\begin{aligned} \tau^n &= \underbrace{\tau \ldots \tau}_{n} \\ \tau^n \star F &= (p)\tau^n.F\langle p\rangle \end{aligned}$$

To prove that $M \cong_C N$ iff $\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!]$, the implication form left to right is the hardest. To work it out, it is technically convenient to work with an encoding $\mathcal{H}'$ which expands $\mathcal{H}$ by adding some τ-actions. The number of these extra τ-actions is not fixed, hence $\mathcal{H}'[\![M]\!]$ defines a *set* of agents rather than a *single* agent. We require that, however, at least one τ-action is added in the output action of the application rule; this τ-action is underlined for clearness in the definition of $\mathcal{H}'$ below. The set (of first-order abstractions) $\mathcal{H}'[\![M_1]\!]$ is defined inductively bottom-up on the structure of M_1; for every $F_M \in \mathcal{H}'[\![M]\!]$, $F_N \in \mathcal{H}'[\![N]\!]$ and $n \geq 0$

we have:

$$\begin{aligned}
\mathcal{H}'[\![\lambda x.M]\!] &\ni (p)\tau^n.p(X,q).F_M\langle q\rangle \\
\mathcal{H}'[\![x]\!] &\ni \tau^n \star X \\
\mathcal{H}'[\![MN]\!] &\ni (p)\tau^n.\boldsymbol{\nu}\, v\left(F_M\langle v\rangle \mid \overline{v}\left\langle \underline{\tau}\star F_N, p\right\rangle.\mathbf{0}\right) \\
\mathcal{H}'[\![c]\!] &\ni \tau^n \star Tr_c
\end{aligned}$$

We denote by $\mathcal{H}'[\![M]\!]\langle p\rangle$ the set of processes

$$\{F\langle p\rangle \ : \ F \in \mathcal{H}'[\![M]\!]\}$$

We write $P \equiv\in \mathcal{H}'[\![M]\!]\langle p\rangle$ if $P \equiv P' \in \mathcal{H}'[\![M]\!]\langle p\rangle$, for some P'. Occasionally, we use these same notations, "$\in$" and "$\equiv\in$", for $\mathcal{H}$; therefore, for instance, we might write $P \in \mathcal{H}[\![M]\!]\langle p\rangle$ to mean that $P = \mathcal{H}[\![M]\!]\langle p\rangle$.

By Theorem 4.4.1 and Lemma 4.4.2, the τ-actions added in $\mathcal{H}'$ do not affect weak context congruence. Since by Corollary 4.7.8 weak context congruence coincides with weak barbed congruence and the latter implies weak barbed equivalence, we have:

Lemma 6.6.9 *If $M, N \in \Lambda_C$ and $F_M \in \mathcal{H}'[\![M]\!]$, $F_N \in \mathcal{H}'[\![N]\!]$, then*

$$\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!] \quad \textit{iff} \quad F_M \approx F_N. \qquad \square$$

Now we present some properties of $\mathcal{H}$ (as previously extended to Λ_C) and of $\mathcal{H}'$ needed later in the proofs of the main theorems (6.6.17 and 6.6.19). We start with the operational correspondence between λ and process terms.

Lemma 6.6.10 (operational correspondence) *Let $M \in \Lambda_C$ and $\mathcal{K} \in \{\mathcal{H}, \mathcal{H}'\}$:*

1. *Suppose that $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \xrightarrow{\mu}$, for $\mu \neq \tau$ and $sbj(\mu) \neq p$.*

   Then, for some $c, M_1, ..., M_n$, it must be $M = cM_1...M_n$ and $\mu \equiv (\boldsymbol{\nu}\, q)\overline{c}\langle q\rangle$

2. *Suppose that $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \xrightarrow{\mu}$, for $\mu \neq \tau$ and $sbj(\mu) = p$.*

   Then for some M_1, it must be $M = \lambda x.M_1$ and μ is an input action.

3. *Suppose* $M \longrightarrow M'$. *Then for any* $P \in \mathcal{K}[\![M]\!]\langle p\rangle$, *there is* Q *s.t.* $P \Longrightarrow Q \equiv\in \mathcal{K}[\![M']\!]$.

4. *Suppose* $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \longrightarrow Q$. *Then it holds that* $Q \in \mathcal{K}[\![M]\!]\langle p\rangle$ *or there is* M' *s.t.* $M \longrightarrow M'$ *and* $Q \equiv\in \mathcal{K}[\![M']\!]\langle p\rangle$.

PROOF: All cases can be proved by simple inductions on the structure of M, and are similar to the proof of Proposition 6.3.6. For $\mathcal{H}'$ one also needs the fact that if $F_1 \in \mathcal{H}'[\![M_1]\!]$ and $F_2 \in \mathcal{H}'[\![M_2]\!]$, for $M_1, M_2 \in \Lambda_C^o$, then $F_1\{F_2/X\} \in \mathcal{H}'[\![M_1\{M_2/x\}]\!]$; this property follows immediately from the compositional definition of $\mathcal{H}'$. □

In the previous lemma, for $\mathcal{K} = \mathcal{H}$ the correspondence between λ-reduction and HOπ-reductions in cases (3) and (4) can be made one-to-one as in Proposition 6.3.6. However, for our purposes, the weaker version presented will suffice.

A process $P \in \mathcal{K}[\![M]\!]\langle p\rangle$ has only one possible reduction, that is, if $P \longrightarrow Q$ and $P \longrightarrow Q'$, then $Q = Q'$: This means that P can be equated to $\tau.Q$ and therefore, reasoning similarly as for Lemma 6.6.9, we have:

Corollary 6.6.11 *Let* $M \in \Lambda_C$ *and* $\mathcal{K} \in \{\mathcal{H}, \mathcal{H}'\}$. *If* $\mathcal{K}[\![M]\!]\langle p\rangle \ni P \longrightarrow Q$, *then* $P \approx Q$. □

Now we introduce the agent

$Out_n \stackrel{def}{=}$

$$(q, p, X_1, ..., X_n)(\boldsymbol{\nu}\, r_1, ..., r_{n-1})\big(\overline{q}\langle X_1, r_1\rangle|\overline{r}_1\langle X_2, r_2\rangle|...|\overline{r}_{n-2}\langle X_{n-1}, r_{n-1}\rangle|\overline{r}_{n-1}\langle X_n, p\rangle\big)$$

If s is the sort of names p, q, r, then $Out_n : (s, s, \underbrace{(s), ..., (s)}_{n})$.

$Out_n\langle q, p, F_1, \ldots, F_n\rangle$ is a process which outputs $F_1, \ldots, F_n$ in the order, each along a different name. Such a name had been exported, as a private name, in the previous output of the process; exceptions are the first output, which takes place at q, and the last one, where the name emitted is p. We use Out_n to represent in a compact and manageable way encodings of terms with a constant in head position. In fact it holds that:

Lemma 6.6.12

1. $\mathcal{H}[\![cM_1...M_n]\!]\langle p\rangle \equiv \boldsymbol{\nu}\, q\left(\overline{c}\langle q\rangle \mid Out_n\left\langle q, p, \mathcal{H}[\![M_1]\!], ..., \mathcal{H}[\![M_n]\!]\right\rangle\right)$

2. *If* $P \in \mathcal{H}'[\![cM_1...M_n]\!]\langle p\rangle$, *then for some* n, *and* m,

$$P \equiv \tau^n \boldsymbol{\nu}\, q\left(\tau^m.\overline{c}\langle q\rangle \mid Out_n\left\langle q, p, \tau \star F_1, \ldots, \tau \star F_2\right\rangle\right)$$

where $F_i \in \mathcal{H}'[\![M_i]\!]$ *for each* $1 \leq i \leq n$.

PROOF: Simple inductions on n. □

We recall that $\simeq_{\mathrm{Ct}}$ and $\simeq_{\mathrm{Nr}}$ represent strong context and normal bisimulation, introduced in Chapter 4, and that $\approx_{\mathrm{Ct}}$, $\approx_{\mathrm{Nr}}$ are their weak versions (by Corollary 4.7.8 all coincide with $\approx$). Lemmas 6.6.13 and 6.6.15 below show properties of the agent Out_n which will be needed in Proposition 6.6.16 and Theorem 6.6.19, respectively.

Lemma 6.6.13 *Let* $\widetilde{m} = m_1, ..., m_n$ *and suppose* $\widetilde{m} \cap \left(fn(F_1, ..., F_n) \cup \{q, p\}\right) = \emptyset$. *Then* $Q_1 \simeq_{\mathrm{Nr}} Q_2$, *where*

$$\begin{aligned} Q_1 &\stackrel{def}{=} Out_n\left\langle q, p, \tau \star F_1, ..., \tau \star F_n\right\rangle \\ Q_2 &\stackrel{def}{=} \boldsymbol{\nu}\, \widetilde{m}\left(Out_n\left\langle q, p, Tr_{m_1}, \ldots, Tr_{m_n}\right\rangle \mid \textstyle\prod_{i=1}^{n} !\, m_i(r).F_i\langle r\rangle\right) \end{aligned}$$

PROOF: Use Lemma 4.4.6 and the inclusion $\simeq_{\mathrm{Ct}} \subset \simeq_{\mathrm{Nr}}$ (the requirements in the definition of $\simeq_{\mathrm{Nr}}$ are a subset of those in the definition of $\simeq_{\mathrm{Ct}}$). □

In the following lemma, the assertion (2) also holds when F and G are higher-order abstractions. We have not reported it because we shall not need it.

Lemma 6.6.14

1. *Let* $\mathcal{N}_{Q_i} = \{a \ : \ Q_i \Downarrow_a\}$, $i = 1, 2$, *and suppose* $fn(P_1, P_2) \cap \left(\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_2}\right) = \emptyset$. *Then* $P_1 \mid Q_1 \approx_{\mathrm{Nr}} P_2 \mid Q_2$ *implies* $P_1 \approx_{\mathrm{Nr}} P_2$.

2. *Let* $m \notin fn(F, G)$. *Then* $!\, m(x).F\langle x\rangle \approx_{\mathrm{Nr}} !\, m(x).G\langle x\rangle$ *implies* $F \approx_{\mathrm{Nr}} G$.

PROOF: We first prove (1). Take

$$\mathcal{R} = \{(P_1, P_2) \;:\; P_1 \mid Q_1 \cong_{\mathrm{Nr}} P_2 \mid Q_2 \quad \text{for some } Q_1, Q_2 \text{ with } fn(P_1, P_2) \cap \left(\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_2}\right) = \emptyset \;\}$$

Since $fn(P_1, P_2) \cap (\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_1}) = \emptyset$, we do not have to "test" P_1 and P_2 with names in $\mathcal{N}_{Q_1} \cup \mathcal{N}_{Q_2}$ and therefore the side condition in the definition of $\mathcal{R}$ can be preserved; moreover no interaction between P_i and Q_i is possible, $i = 1, 2$. Then the proof that $\mathcal{R}$ is a $\cong_{\mathrm{Nr}}$-bisimulation becomes straightforward.

Now the assertion (2) of the lemma. We have to show that if y ranges over the free names of F and G plus a fresh name, it holds that $F\langle y\rangle \cong_{\mathrm{Nr}} G\langle y\rangle$. Since m is not free in F and G, we can assume, without loss of generality, that y is different from m. Since $!\,m(x).F\langle x\rangle \cong_{\mathrm{Nr}} !\,m(x).G\langle x\rangle$ and $!\,m(x).F\langle x\rangle \xrightarrow{m\langle y\rangle} F\langle y\rangle \mid !\,m(x).F\langle x\rangle$, we have, for some P

$$!\,m(x).G\langle x\rangle \xrightarrow{m\langle y\rangle} P \cong_{\mathrm{Nr}} F\langle y\rangle \mid !\,m(x).F\langle x\rangle \tag{6.2}$$

Since m does not occur in $G\langle y\rangle$, no interaction between $G\langle y\rangle$ and $!\,m(x).G\langle x\rangle$ may have occurred; therefore P is of the form $P_G \mid !\,m(x).G\langle x\rangle$, for some P_G s.t.

$$G\langle y\rangle \Longrightarrow P_G. \tag{6.3}$$

Thus (6.2) can be written as $P_G \mid !\,m(x).G\langle x\rangle \cong_{\mathrm{Nr}} F\langle y\rangle \mid !\,m(x).F\langle x\rangle$. From this, we get

$$P_G \cong_{\mathrm{Nr}} F\langle y\rangle \tag{6.4}$$

using the assertion (1) of the lemma, since $\mathcal{N}_{!m(x).G\langle x\rangle} = \mathcal{N}_{!m(x).F\langle x\rangle} = \{m\}$ and m is not free in $F\langle y\rangle$ and P_G. In a symmetric way (just exchange F and G), we can derive, for some P_F s.t. $F\langle y\rangle \Longrightarrow P_F$,

$$P_F \cong_{\mathrm{Nr}} G\langle y\rangle \tag{6.5}$$

Now, we exploit (6.4) and (6.5) to show that $F\langle y\rangle \cong_{\mathrm{Nr}} G\langle y\rangle$: For this we prove that

$$\mathcal{R} = \{(F\langle y\rangle, G\langle y\rangle)\} \cup \cong_{\mathrm{Nr}}$$

is a $\cong_{\mathrm{Nr}}$-bisimulation. Suppose $F\langle y\rangle \xrightarrow{\mu} Q_F$ and μ is an input action (the cases in which μ is an output or a silent action are similar). We show how $G\langle y\rangle$ can match this move: Since $F\langle y\rangle \cong_{\mathrm{Nr}} P_G$, we have $P_G \stackrel{\mu}{\Longrightarrow} Q_G \cong_{\mathrm{Nr}} Q_F$; therefore, by (6.3), $G\langle y\rangle \Longrightarrow P_G \stackrel{\mu}{\Longrightarrow} Q_G \cong_{\mathrm{Nr}} Q_F$, which closes the bisimulation. The case in which $G\langle y\rangle$ moves first is analogous. □

Lemma 6.6.15 *It holds that*

$$Out_n\langle q, p, F_1, \ldots, F_n\rangle \cong_{\mathrm{Ct}} Out_n\langle q, p, G_1, \ldots, G_n\rangle \text{ iff for all } 1 \leq i \leq n,\ F_i \approx G_i.$$

PROOF: The implication from right to left can be inferred from the congruence properties of $\cong_{\mathrm{Ct}}$ (Theorem 4.4.1). Thus, we only have to consider the implication from left to right. We exploit the characterisation of $\cong_{\mathrm{Ct}}$ in terms of $\cong_{\mathrm{Nr}}$ (Corollary 4.7.8) and we proceed by induction on n. We only consider the inductive case. Let

$$P \stackrel{def}{=} Out_n\langle q, p, F_1, \ldots, F_n\rangle \qquad Q \stackrel{def}{=} Out_n\langle q, p, G_1, \ldots, G_n\rangle$$

The only transitions they can perform are

$$\begin{array}{lll} P & \xrightarrow{(\nu r_1)\overline{q}\langle F_1, r_1\rangle} & Out_{n-1}\langle r_1, p, F_2, \ldots, F_n\rangle \stackrel{def}{=} P_1 \\ Q & \xrightarrow{(\nu r_1)\overline{q}\langle G_1, r_1\rangle} & Out_{n-1}\langle r_1, p, G_2, \ldots, G_n\rangle \stackrel{def}{=} Q_1 \end{array}$$

Then, by definition of $\cong_{\mathrm{Nr}}$ (see also the discussion at the end of Section 4.7.3), $P \cong_{\mathrm{Nr}} Q$ implies

$$!\,m(x).F_1\langle x\rangle \mid P_1 \cong_{\mathrm{Nr}} !\,m(x).G_1\langle x\rangle \mid Q_1$$

where m is a fresh name. Since $\{a \ :\ P_1 \Downarrow_a \ \text{or}\ Q_1 \Downarrow_a\} = \{r_1\}$ with $r_1 \notin fn(F_1, G_1, \{m\})$ and $m \notin fn(P_1, Q_1)$, using Lemma 6.6.14(1) twice we infer

$$!\,m(x).F_1\langle x\rangle \cong_{\mathrm{Nr}} !\,m(x).G_1\langle x\rangle \quad \text{and} \quad P_1 \cong_{\mathrm{Nr}} Q_1$$

Now from the former we get $F_1 \cong_{\mathrm{Nr}} G_1$ using Lemma 6.6.14(2); from the latter we get $F_i \cong_{\mathrm{Nr}} G_i$, $2 \leq i \leq n$ using the inductive assumption. □

Proposition 6.6.16 *For every $M, N \in \Lambda_C$, and $F_M \in \mathcal{H}'[\![M]\!]$, $F_n \in \mathcal{H}'[\![N]\!]$, it holds that $M \cong_C N$ implies $F_M \approx F_N$.*

PROOF: We shall make some abuse of language, and use the same symbol $\mathcal{H}'[\![M]\!]\langle p\rangle$ to denote both the set of processes $\mathcal{H}'[\![M]\!]\langle p\rangle$ and a generic process of this set. Thus, for instance, $P = Q \mid \mathcal{H}'[\![M]\!]\langle p\rangle$ means that $P = Q \mid P_M$, for some $P_M \in \mathcal{H}'[\![M]\!]\langle p\rangle$.

Again, we use the characterisation of $\approx$ in terms of $\approx_{\mathrm{Nr}}$. Let $\mathcal{R}$ be the set of all the pairs (P, Q) s.t.

$$P = R_i \mid R_z \mid R_v \mid \prod_{j=1}^{n_j} !\, m_j(q).\mathcal{H}'[\![M_j]\!]\langle q\rangle \mid \prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$$

$$Q = R_i \mid R_z \mid R_v \mid \prod_{j=1}^{n_j} !\, m_j(q).\mathcal{H}'[\![N_j]\!]\langle q\rangle \mid \prod_{r=1}^{n_r} \mathcal{H}'[\![N_r]\!]\langle q_r\rangle$$

where

$$R_i \stackrel{def}{=} \prod_{i=1}^{n_i} \overline{q_i}\langle Tr_{c_i}, p_i\rangle, \quad R_z \stackrel{def}{=} \prod_{z=1}^{n_z} !\, m_z(q).Tr_{c_z}\langle q\rangle, \quad R_v \stackrel{def}{=} \prod_{v=1}^{n_v} \overline{c_v}\langle q_v\rangle$$

and where

- all free names and numerical variables appearing in the definition of P, Q are existentially quantified,
- for all j, z, it holds that $M_j \cong_C N_j$ and $M_z \cong_C N_z$,
- names m_z, m_j are different each other and different from all other names occurring free in P, Q.

We show that $\mathcal{R}$ is a $\approx_{\mathrm{Nr}}$-bisimulation up-to $\simeq_{\mathrm{Nr}}$ and up-to restriction. This would prove the theorem, because if $M \cong_C N$, then we have, for any p,

$$\mathcal{H}'[\![M]\!]\langle p\rangle \equiv \quad \mathcal{R} \quad \equiv \mathcal{H}'[\![N]\!]\langle p\rangle$$

and hence $\mathcal{H}'[\![M]\!] \approx_{\mathrm{Nr}} \mathcal{H}'[\![N]\!]$. Let (P, Q) be as in the definition of $\mathcal{R}$. We have to consider the possible moves by P and show how Q can match each of them.

We proceed by a case analysis on the component(s) of P which have caused the action.

Case (a) The action has been caused by R_i.

This means that for some i, $P \stackrel{\overline{q_i}\langle Tr_{c_i}, p_i\rangle}{\longrightarrow} P'$. Then, using the same R_i, also $Q \stackrel{\overline{q_i}\langle Tr_{c_i}, p_i\rangle}{\longrightarrow} Q'$ and if m is a fresh name

$$!m(q).Tr_{c_i}\langle q\rangle \mid P' \equiv \mathcal{R} \equiv !m(q).Tr_{c_i}\langle q\rangle \mid Q'$$

as required by definition of $\cong_{\mathrm{Nr}}$ (see also the discussion in Section 4.7.3).

Case (b) The action has been caused by R_z or R_v.

Easy.

Case (c) The action has been caused by $\prod_{j=1}^{n_j} !m_j(a).\mathcal{H}'[\![M_j]\!]\langle a\rangle$.

Easy.

Case (d) The action has been caused by $\prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$.

It is crucial here to use the operational correspondence of Lemma 6.6.10. One thing which follows from it is that processes encoding λ-terms cannot interact each other. Therefore, if $\prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$ produces an action, then the action must have been originated by some specific $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$. We shall only show how $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle$ can match this move. It will be easy to see from here how to infer the action of Q which matches the one by P, and we omit it.

Suppose $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$ can perform an input action. Then, M_r has to be an abstraction, say $M_r = \lambda x.M_r'$. By definition of $\cong_{\mathrm{Nr}}$, if c is a fresh constant, we have to check that for every q, $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle$ can match the following action:

$$\mathcal{H}'[\![\lambda x.M_r']\!]\langle q_r\rangle \stackrel{q_r\langle Tr_c, q\rangle}{\longrightarrow} \mathcal{H}'[\![M_r']\!]\langle q\rangle\{Tr_c/X\} = \mathcal{H}'[\![M_r'\{c/x\}]\!]\langle q\rangle$$

Since $M_r \cong_C N_r$, for some N_r',

$$N_r \Longrightarrow \lambda x.N'_r \quad \text{with} \quad M'_r\{c/x\} \cong_C N'_r\{c/x\}.$$

Then, by Lemma 6.6.10(3), $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle \Longrightarrow\equiv \mathcal{H}'[\![\lambda x.N'_r]\!]\langle q_r\rangle$ and therefore, by definition of $\mathcal{H}'$,

$$\mathcal{H}'[\![\lambda x.N'_r]\!]\langle q_r\rangle \stackrel{q_r\langle Tr_c,q\rangle}{\Longrightarrow} \mathcal{H}'[\![N'_r\{c/x\}]\!]\langle q\rangle$$

which matches the action by $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$. Suppose now that $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle$ can perform a silent action; by Lemma 6.6.10(4), such action must be of the form

$$\mathcal{H}'[\![M_r]\!]\langle q_r\rangle \longrightarrow\equiv \mathcal{H}'[\![M'_r]\!]\langle q_r\rangle, \text{ for some } M'_r \text{ s.t. } M'_r = M_r \text{ or } M_r \longrightarrow M'_r.$$

Since by Lemma 6.3.2 $M'_r \cong M_r \cong N_r$, the process $\mathcal{H}'[\![N_r]\!]\langle q_r\rangle$ does not need to move.

Finally, suppose that $\mathcal{H}'[\![M_r]\!]\langle q_r\rangle \xrightarrow{\mu} P_1$ and μ is an output action. Then, using Lemma 6.6.10(1) and 6.6.12(2), we get that

$$\mu = (\boldsymbol{\nu}\, q)\overline{c}\langle q\rangle, \qquad M_r = cM_1 \ldots M_n \qquad \text{and}$$
$$P_1 \equiv Out_n\big\langle q, q_r, \tau \star \mathcal{H}'[\![M_1]\!], ..., \tau \star \mathcal{H}'[\![M_n]\!]\big\rangle$$

Since $M_r \cong_C N_r$, there exist $N_1, ..., N_n$ s.t. $N \Longrightarrow cN_1...N_n$ with $M_i \cong_C N_i$, for $1 \leq i \leq n$. By Lemma 6.6.10(3) and again Lemma 6.6.12(2), also

$$\mathcal{H}'[\![N_r]\!]\langle p\rangle \Longrightarrow\equiv \mathcal{H}'[\![cN_1...N_n]\!]\langle q_r\rangle \stackrel{(\boldsymbol{\nu}\, q)\overline{c}\langle q\rangle}{\Longrightarrow}\equiv$$
$$Out_n\big\langle q, q_r, \tau \star \mathcal{H}'[\![N_1]\!], ..., \tau \star \mathcal{H}'[\![N_n]\!]\big\rangle \stackrel{def}{=} Q_1$$

This is the point where we need Lemma 6.6.13. Using it we infer

$$P_1 \simeq_{\mathrm{Nr}}$$
$$(\boldsymbol{\nu}\, m_1, ..., m_n)\Big(\prod_{i=1}^{n} !\, m_i(r).\mathcal{H}'[\![M_i]\!]\langle r\rangle \mid Out_n\big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\big\rangle\Big) \stackrel{def}{=} P_2$$

$Q_1 \simeq_{\mathbf{Nr}}$
$(\boldsymbol{\nu}\, m_1, ..., m_n)\big(\prod_{i=1}^{n} !\, m_i(r).\mathcal{H}'[\![N_i]\!]\langle r\rangle \mid Out_n\big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\big\rangle\big) \stackrel{def}{=} Q_2$

Since $\mathcal{R}$ is a bisimulation up-to restriction, from P_2 and Q_2 we can discharge the outermost restrictions $m_1, \ldots, m_n$, as well as those inside the process $Out_n\big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\big\rangle$ (to do this, the restrictions have first to be pushed at the outermost level using $\equiv$). At the end, we are left with processes which are in a format admissible for $\mathcal{R}$ (what is left of $Out_n\big\langle q, q_r, Tr_{m_1}, ..., Tr_{m_n}\big\rangle$ is the parallel composition of processes in the same format as the processes R_i of the definition of $\mathcal{R}$).

Case (e) The action comes from an interaction between R_i, which performs the output, and $\prod_{r=1}^{n_r} \mathcal{H}'[\![M_r]\!]\langle q_r\rangle$, which performs the input.

Let $\overline{q_t}\langle Tr_{c_t}, p_t\rangle$ and $\mathcal{H}'[\![M_t]\!]\langle q_t\rangle$ be the processes involved. By Lemma 6.6.10, for $\mathcal{H}'[\![M_t]\!]\langle q_t\rangle$ to be able to perform an input, M_t must be an abstraction, say $M_t = \lambda x.M'_t$. Then the interaction which is produced is of the form

$$\overline{q_t}\langle Tr_{c_t}, p_t\rangle \mid q_t(X, q).\mathcal{H}'[\![M'_t]\!]\langle q\rangle \longrightarrow \mathcal{H}'[\![M'_t\{c_t/x\}]\!]\langle p_t\rangle \qquad (6.6)$$

Similarly as for case (d), we only show how the corresponding components in Q can match this interaction, yielding to processes in a format admissible for $\mathcal{R}$. From $M_t \cong_C N_t$ and Lemma 6.6.10 we get that for some N'_t,

$$\mathcal{H}'[\![N_t]\!]\langle q_t\rangle \Longrightarrow\equiv \mathcal{H}'[\![\lambda x.N'_t]\!]\langle q_t\rangle \Longrightarrow q_t(X, q).\mathcal{H}'[\![N'_t]\!]\langle q\rangle,$$

with $\lambda x.M'_t \cong_C \lambda x.N'_t$. Then

$$\overline{q_t}\langle Tr_{c_t}, p_t\rangle \mid q_t(X, q).\mathcal{H}'[\![N'_t]\!]\langle q\rangle \longrightarrow\equiv \mathcal{H}'[\![N'_t\{c_t/x\}]\!]\langle p_t\rangle$$

with $M'_t\{c_t/x\} \cong_C N'_t\{c_t/x\}$. Therefore this matches the move (6.6).

No other interaction between components of P is possible. Hence the above cases are exhaustive.

Theorem 6.6.17 *If* $M, N \in \Lambda_C$ *and* $M \cong_C N$ *then* $\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!]$

PROOF: Follows from Proposition 6.6.16 and Lemma 6.6.9.

Remark 6.6.18 We could have obtained the result in Theorem 6.6.17 in a much simpler way if we had available for the HOπ the technique of bisimulation up-to context ([Sana]). It replaces the ordinary clause in the definition of bisimulation, namely (we consider here the strong case of CCS)

- if $P \xrightarrow{\mu} P'$, some Q' exists s.t. $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{R} Q'$

with the clause

- if $P \xrightarrow{\mu} P'$, then some P_1, Q_1 and a context $C[\cdot]$ exist s.t.

$$P' = C[P_1],\ Q \xrightarrow{\mu} C[Q_1] \text{ and } P_1 \mathcal{R} Q_1.$$

(Notice that bisimulation up-to restriction is a particular case of bisimulation up-to context). Using bisimulation up-to context, we could have avoided the introduction of the encoding $\mathcal{H}'$, and we could have given Proposition 6.6.16 directly in terms of $\mathcal{H}$. Moreover, in the proof it would have been enough to work with the set of pairs

$$\mathcal{R} = \{(\mathcal{H}[\![M]\!]\langle p \rangle, \mathcal{H}[\![N]\!]\langle p \rangle) \ : \ M \cong_C N\}$$

However, the generalisation of the bisimulation up-to context technique for HOπ is not trivial and we have preferred to leave it for future work. Another result whose proof would become much easier with this technique is Proposition 4.6.5.

From the work in this thesis we can draw the conclusion that the development of techniques to facilitate the construction of bisimulations, such us bisimulation up-to and bisimulation up-to context, is an important direction to pursue in higher-order process calculi.

The inverse implication of Theorem 6.6.17 is easier:

Theorem 6.6.19 *If $M, N \in \Lambda_C$ and $\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!]$ then $M \cong_C N$.*

PROOF: If $\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!]$ then, by definition of $\approx$ on abstractions, for any p, $\mathcal{H}[\![M]\!]\langle p\rangle \approx \mathcal{H}[\![N]\!]\langle p\rangle$. We exploit the characterisation of $\approx$ in terms of $\cong_{\mathrm{Ct}}$ and prove that

$$\mathcal{R} = \{(M,N) \,:\, \mathcal{H}[\![M]\!]\langle p\rangle \cong_{\mathrm{Ct}} \mathcal{H}[\![N]\!]\langle p\rangle, \text{ for some } p\}$$

is a $\cong_C$-bisimulation. First, suppose $M \Longrightarrow \lambda x.M'$. We have to find N' s.t. $N \Longrightarrow \lambda x.N'$ and for all $L \in \Lambda_C$, $M'\{L/x\}\ \mathcal{R}\ N'\{L/x\}$. Since $M \Longrightarrow \lambda x.M'$, using Lemma 6.6.10(3)

$$\mathcal{H}[\![M]\!]\langle p\rangle \Longrightarrow\equiv \mathcal{H}[\![\lambda x.M']\!]\langle p\rangle = p(X,q).\mathcal{H}[\![M']\!]\langle q\rangle$$

Since $\mathcal{H}[\![M]\!]\langle p\rangle \cong_{\mathrm{Ct}} \mathcal{H}[\![N]\!]\langle p\rangle$ and $\mathcal{H}[\![\lambda x.M']\!]\langle p\rangle$ can perform an input action **at** p, using Lemmas 6.6.10 (and Corollary 6.6.11) we get that for some $\lambda x.N'$,

$$\mathcal{H}[\![N]\!]\langle p\rangle \Longrightarrow\equiv \mathcal{H}[\![\lambda x.N']\!]\langle p\rangle = p(X,q).\mathcal{H}[\![N']\!]\langle q\rangle$$

with

$$p(X,q).\mathcal{H}[\![N']\!]\langle q\rangle \cong_{\mathrm{Ct}} p(X,q).\mathcal{H}[\![M']\!]\langle q\rangle$$

By definition of $\cong_{\mathrm{Ct}}$, the above equivalence means that for all F and q

$$\mathcal{H}[\![M']\!]\langle q\rangle\{F/X\} \cong_{\mathrm{Ct}} \mathcal{H}[\![N']\!]\langle q\rangle\{F/X\}$$

In particular, this is true when F is the encoding of a Λ_C-term; thus we get $M'\{L/X\}\ \mathcal{R}\ N'\{L/X\}$, for every $L \in \Lambda_C$.

The case when $M \Longrightarrow cM_1 \ldots M_n$ can be worked out similarly. From $\mathcal{H}[\![M]\!]\langle p\rangle \cong_{\mathrm{Ct}} \mathcal{H}[\![N]\!]\langle p\rangle$ and Lemmas 6.6.10, 6.6.12(1), we get

$$\begin{array}{lcll} \mathcal{H}[\![M]\!]\langle p\rangle & \Longrightarrow\equiv & \mathcal{H}[\![cM_1 \ldots M_n]\!] & \equiv \quad Out_n\big\langle c, p, \mathcal{H}[\![M_1]\!], \ldots, \mathcal{H}[\![M_n]\!]\big\rangle \\ \mathcal{H}[\![N]\!]\langle p\rangle & \Longrightarrow\equiv & \mathcal{H}[\![cN_1 \ldots N_{n'}]\!] & \equiv \quad Out_{n'}\big\langle c, p, \mathcal{H}[\![N_1]\!], \ldots, \mathcal{H}[\![N_{n'}]\!]\big\rangle \end{array}$$

with $Out_n\langle c, p, \mathcal{H}[\![M_1]\!], \ldots, \mathcal{H}[\![M_n]\!]\rangle \cong_{\mathrm{Ct}} Out_{n'}\langle c, p, \mathcal{H}[\![N_1]\!], \ldots, \mathcal{H}[\![N_{n'}]\!]\rangle$. This forces $n = n'$ (otherwise, one process could perform more output actions than the other and hence they would not be equivalent). Then by Lemma 6.6.15, $\mathcal{H}[\![M_i]\!] \cong_{\mathrm{Ct}} \mathcal{H}[\![N_i]\!]$ for all $1 \leq i \leq n$, and by Lemma 6.6.10(4), $N \Longrightarrow cN_1 \ldots N_n$. This yields $M_i \mathcal{R} N_i$, as required by definition of $\cong_C$-bisimulation. □

Using Theorems 6.6.17 and 6.6.19, we get

Theorem 6.6.20 *If $M, N \in \Lambda_C$, it holds that $M \cong_C N$ iff $\mathcal{H}[\![M]\!] \approx \mathcal{H}[\![N]\!]$* □

As pointed out at the beginning of this section, on encodings of λ-terms, $\approx$ and $\approx^c$ coincide. Since two λ-terms are defined to be λ-observational equivalent if their encodings are in the relation $\approx^c$, the above result gives us a direct characterisation of λ-observational equivalence.

Corollary 6.6.21 (direct characterisation of λ-observational equivalence) *Two Λ terms M and N are λ-observational equivalent iff $M \cong_C N$* □

Let $\mathcal{D}_C$ be the extension to Λ_C of the model $\mathcal{D}$ of Section 6.5.2; $\mathcal{D}_C$ is defined as $\mathcal{D}$ with Λ_C in place of Λ, and utilising the extension of $\mathcal{H}$ to Λ_C given in this section.

Corollary 6.6.22 (full abstraction) *$\mathcal{D}_C$ is a fully abstract model for the lazy λ-calculus enriched with constants.* □

Starting from these results, the study of λ-observational equivalence has been deepened in [San92]. The main outcomes have been mentioned in the introduction of Section 6.6.2; in particular, they — surprisingly — allow us to obtain λ-observational equivalence by enriching λ-calculus with a simple non-deterministic operator (in place of constants).

Corollary 6.6.22 suggests an analogy with the full abstraction result for PCF using error generator and escape handlers recently studied by Cartwright, Felleisen and Curien [CF92,Cur92]. Intuitively, in both cases the symbols or operators with

which the λ-calculus is enriched are used to explore the internal structure of λ-terms. We leave for future investigations whether this analogy can be made deeper.

Chapter 7

Conclusions and Future Work

In this thesis we have investigated various issues on calculi for mobile systems. The following are the major contributions:

1. Introduction of barbed bisimulation as universal (i.e. usable in a wide range of calculi) tool to capture natural bisimilarity equivalences.

2. The definition and the analysis of HOπ. We have derived mathematical tools — like factorisation theorem, trigger and normal bisimulation — which make it possible to reason in a relatively simple way with a higher-order process calculus, notwithstanding the complexity of its transitions.

3. The proof of representability of HOπ in π-calculus via the compilation $\mathcal{C}$.

4. The comparison between λ-calculus and calculi for mobile processes. In particular for the lazy λ-calculus we have shown that Milner's encoding into π-calculus is factorised by our encoding into HOπ and the compilation $\mathcal{C}$. This strengthens the naturalness of the translations involved. It has also been used to find a direct characterisation of the equivalence on λ-terms induced by Milner's encoding.

We have proved the faithfulness of $\mathcal{C}$ (and of the other transformations) only w.r.t. barbed equivalence and congruence. But we believe that $\mathcal{C}$ respects most of

the well-known weak equivalences which admit a uniform definition over higher-order and first-order calculi, such as *testing equivalence* [DH84], or *refusal semantics* [Phi87]. The reason for this is the close operational correspondence between encoded and encoding agents shown in Lemma 5.2.2.

Indeed $\mathcal{C}$ might even be used to *define* equivalences in HOπ. Take for instance *trace semantics* as defined in [Hoa85], or *causal bisimulation* [DD89]. They have originally been proposed for calculi without explicit mobility, but can easily be adapted to π-calculus. More delicate is their extension to a higher-order calculus; as usual, it is not obvious which condition to impose on higher-order outputs. However, if P, Q are HOπ processes and $\ll\gg$ means weak trace equivalence or weak causal bisimulation, we might just define:

$$P \ll\gg Q \text{ if } \mathcal{C}[\![P]\!] \ll\gg \mathcal{C}[\![Q]\!]$$

and then look for a characterisation of $\ll\gg$ in HOπ which does not mention $\mathcal{C}$.

We have already pointed out specific directions for future work along the thesis. We conclude by mentioning areas with possible broader interest.

Generalisation of the results

Our process languages allow agents defined in terms of an *infinite* number of constants. We used this power in the proofs of direct characterisations of barbed equivalence, for the construction of the contexts with which the processes are tested. In these same proofs, the use of an infinity of constants has, in turn, forced us to allow infinite sums and infinite free names in the syntax of the agents of the calculi. On the other hand, the study of compilation $\mathcal{C}$ and of the algebraic properties of HOπ in Chapters 4 and 5 has been conducted in the subclass of agents defined in terms of a *finite* number of constants. This naturally raises two technical questions. The first is whether the results on barbed equivalence are still provable under the constrain of constant-finiteness, which is the same as to ask whether they are provable in the language in which constants have been

replaced by replication. This should be possible at least on those processes whose transition relation is finite. The second question is whether the proofs in Chapter 4 and 5 can be carried over to agents which use an infinity of constants.

Another technical issue which arises from the work on barbed equivalence is whether in its definition, the class of predicates $\{\downarrow_a\}_a$ (in the weak case $\{\Downarrow_a\}_a$) which are used to know the set of ports at which a process can perform a visible action, can be replaced by the single and less discriminating predicate $\downarrow$ (in the weak case $\Downarrow$) which simply detects whether the process can perform *some* visible action. The answer should be positive in the strong case, but it appears rather difficult in the weak case.

The effect of the sorting on the equivalence of processes

A question which seems very interesting but which has not been analysed, even for the π-calculus, is the effect of the sorting on the equivalence between processes. We shall not pursue this study here in depth; we limit ourselves to present a result in this direction which does not seem so obvious without the theory developed on triggers in Chapter 4.

Let us remind the reader that the results in Chapter 4 were obtained using a downward-closed property which ensures that if (S) is among the object sort of Ob, then also S is. Now, take the processes $a(X).P$ and $a(X).Q$; suppose both respect the downward-closed sorting Ob and that $a(X).P \approx a(X).Q$. This means that $P\{F/X\} \approx Q\{F/X\}$ for each $F : X$. But what happens to $a(X).P \approx a(X).Q$ if the sorting Ob is extended to a sorting $Ob' \supset Ob$? The set of candidates for F increases and with it also the potential capability of discriminating between P and Q. However, we can see that this extra richness is harmless by appealing to the characterisation of barbed equivalence in terms of normal bisimulation (Definition 4.7.1), since the tests required by the latter do not augment when the sorting is extended. Thus, if $Ob \vdash P \approx Q$ is used to to mean that $P \approx Q$ holds in the sorting Ob, we have:

Theorem 7.1 Let Ob be a downward-closed sorting with $P, Q : Ob$, and Ob' a downward-closed extension of Ob. It holds that $Ob \vdash P \approx Q$ iff $Ob' \vdash P \approx Q$. □

We do not know whether the downward-closure hypothesis in Theorem 7.1 can be lifted. Notice that the issue considered in Theorem 7.1 does not arise with first-order processes and sortings, because here extensions of the sorting do not change the set of possible values received in an input. In fact, we could also prove Theorem 7.1 exploiting the compilation $\mathcal{C}$ from higher-order to first-order agents.

Another issue on the connection between sorting and process equivalence is the following. In [San91] and [Tur] a notion of *Most General Sorting* (MGS) has been developed. It is the analogous of the notion of Most General Sorting for types. The MGS of an agent A has the property that any other valid sorting for A can be obtained by *refinement* of the MGS. Now, suppose we can prove the equivalence of two agents A, A' using the MGS of the two. Which informations can we deduce on the equivalence of A and A' in a more refined sorting? For which sortings would we be able to say something?

Adding data to HOπ

The study conducted with the λ-calculus exemplifies the usefulness of the abstraction power of HOπ w.r.t. the π-calculus.

Such abstraction power could be increased by adding some (simple) form of data, like integers, booleans, or lists. Accordingly, the format of object sorts should be enriched to allow for data-communications. For instance, to impose that the names of the sort s carry an integer, we might declare $s \mapsto (N)$, where N is the integer-type. Data should be taken into account also in the definition of the equivalences. The interesting thing is that the compilation $\mathcal{C}$ is easily generalisable to the extended HOπ, since data can be encoded in the π-calculus ([MPW92,Wal91]). Then the proof that the faithfulness of $\mathcal{C}$ is maintained would give us confidence that what we are developing is sensible.

Semantics of object-oriented languages

Two interesting approaches to the denotational semantics of parallel object-oriented languages are exhibited in [AdBKR89] and [Wal90]. In both cases the source language is POOL [Ame89]. Let us point out here their weaknesses (in our view). In the former, a heavy mathematical machinery — based on category of metric spaces, (generalisations of) Banach's theorem — is needed to ensure the well-definedness of the semantics. Consequently, the definition of the process domain and especially the definition of the operator of parallel composition on the process domain require a substantial effort.

In the latter, POOL is given semantics by translating it into the π-calculus. The translation however is "flat", in the following sense. First of all, there is no concept of type to give an overall idea of the use and the purpose of the various processes defined. Secondly, the translation of every syntactic phrase is "context-free", in that no argument — like state, continuation or environment — is supplied. Finally, because the π-calculus is "low-level", the protocols implementing interactions among different components sometimes are burdensome.

We would like to see if it is possible to gain some benefit by using the HOπ as target language. Higher-order sorts would play the rôle of types in [AdBKR89]. The theory developed for the HOπ could be employed to reason on the semantic objects. The representation should be more succinct and readable than the one in [Wal90], especially if data are added to the HOπ as suggested in the previous subsection. As for λ-calculus, using $\mathcal{C}$ the two translations could be compared to see if and where they are different.

Modification of the set of operators used

It is not clear to us at which extent the results on $\mathcal{C}$ and on the direct characterisation of barbed equivalence depend upon the choice of the operators in HOπ and π-calculus. We have already mentioned that — at least for $\mathcal{C}$ — we cannot

lift the restriction on guarded sums (but we do not see this as a strong limitation; we have been arguing in favour of guarded sums in Chapter 2). Sum is a *dynamic* operator because it is discharged when an action is produced. In general, it seems that for the above mentioned results dynamic operators are dangerous. Another example of dynamic operator is Lotos's disabling [BB89].

It would also be interesting to see whether there are a few simple and useful (i.e. with a practical relevance) operators which, when added to HOπ, make barbed equivalence and some variant of higher-order bisimulation (perhaps one based on the use of *location*, as mentioned in Section 4.2) coincide.

Expressiveness of π-calculus

Our study on the translation of HOπ and λ-calculus into π-calculus may be seen as just an aspect of a more general question: how expressive is the π-calculus? In [DS85], Robert de Simone has proved that Meije [AB84] and SCCS [Mil83] are *complete expressive* in the sense that any operator which can be defined by rules which obey certain natural conditions can also be defined directly in terms of the basic combinators of either calculus. How far here can we go with the π-calculus? Which kinds of operators can be represented within it?

Appendix A

Proof of Theorem 3.3.3(2)

In this appendix we show that in π-calculus weak early bisimulation coincides with weak barbed equivalence.

Theorem 3.3.3(2) On π-calculus processes, $\approx_e$ and $\approx$ coincide.

The proof generalises the one for CCS of Theorem 3.3.2. We suppose that the reader has fresh in his/her mind the notation and technicalities in Chapter 3, in particular in Section 3.3. The inclusion $\approx_e \subseteq \approx$ is easy (see Theorem 3.3.2), so we shall concentrate on the opposite containment.

The main difference from the proof of Theorem 3.3.2 is in the definition of the process V. We remind that the task of V is to interact with the processes P or Q of which we want to check the bisimilarity, so to reveal — using the observation predicates of barbed bisimulation — the actions which they can perform. In Theorem 3.3.2 we used the constant V with a parameter H which is a set of pairs of names; moreover, the projection of H on the first component of the pairs, called H_1, contained all names free in P or Q. In π-calculus, since the actions of processes are more complex than in CCS, the summands of V has to be made more sophisticated. Moreover, in π-calculus the set H may augment as effect of name-communications. Consequently, we have to allow enlargements of H. To be able to do this, we add the set of names Y among the parameters of V. This

Y is a countable infinite set of names, and is supposed to contain names which do not already occur in H. These names are used to augment H. Thus, as the computation proceeds, H becomes bigger and Y smaller — but always infinite. (We shall write $V\langle H, Y\rangle$ as parameter for V; to represent $V\langle H, Y\rangle$ as a tuple of names, as by definition of constant parameters in our language, use any standard set-theoretical method for linearising countable unions of countable sets).

We show the proof of the theorem for the monadic case, where all names belong to the same sort and carry exactly one name. In the polyadic case, where names may be partitioned into more than one sort, the use of names in the definition of V below has to be properly constrained, so to respect the sort compatibility in prefixes and matchings (we shall have an example of the use of names of different sorts in Appendix D, when considering the extension of this proof to the HOπ). We recall that the summands $(a1)$ and $(a2)$ of V are used to test, respectively, the input and the output actions of P and Q; the names in, ou are used to "signal" the specific summand of V which has been selected and therefore to know to the kind of action produced by P or Q; the subject and object part of the action can be reconstructed from the branching structure of the subcomponent $W\langle a', b', r\rangle$ (and the summation $\tau.\sum_{(b,b')\in H}[y = b]\, b'$ in (a2)). In $(a1)$, the inner summation $\sum_{(b,b')\in H\cup(y,y')}$ includes the new pair (y, y') in order to test P and Q's inputs also with a name not already in P or Q. In (a2) the summation $\tau.\sum_{(b,b')\in H}[y = b]\, b'$ is used to discriminate between free and bound outputs, as it will become clear later.

In the body of the definition of V below, it is supposed that y and y' are names drawn from Y. Moreover, we use H^+ and Y^- as abbreviations for $H \cup (y, y')$ and $Y - \{y, y'\}$, respectively.

$$
\begin{aligned}
V \stackrel{def}{=} (H,Y)\Big(& \\
& \textstyle\sum_{(a,a')\in H} \sum_{(b,b')\in H\cup(y,y')} \overline{a}\langle b\rangle.\overline{c}.\overline{c}.\left(\tau.W\langle a',b',in\rangle + \tau.V\langle H^+,Y^-\rangle\right) && (a1)\\
+ & \textstyle\sum_{(a,a')\in H} a(y).\overline{c}.\overline{c}. \\
& \qquad \left(\tau.W\langle a',y',ou\rangle + \tau.V\langle H^+,Y^-\rangle + \tau.\textstyle\sum_{(b,b')\in H}[y=b]\,b'\right) && (a2)\\
+ & \ \tau.d_0 \ + \ \tau.d_1 \ \Big) && (a3)
\end{aligned}
$$

where

$$W \stackrel{def}{=} (a',b',r)\big(a'+\tau(b'+r)\big)$$

Similarly as for CCS, the contexts we shall use are of the form

$$C_n^{H,Y}[\cdot] \stackrel{def}{=} [\cdot] \mid V\langle H,Y\rangle \mid Count_n$$

where the definition of $Count_n$ is the same as in Theorem 3.3.2. Now we are ready to give the definition of the relation $\mathcal{R}$ which we shall prove to be a weak early bisimulation. In this definition, the restrictions outside $C_n^{H,Y}[P]$ and $C_n^{H,Y}[Q]$ are produced by the interactions between $V\langle H,Y\rangle$ and P or Q in which a bound name is transmitted. We use $n(H)$ to denote the set of names which occur in H.

$$
\begin{aligned}
\mathcal{R} = \{(P,Q) \ : \ & n,H,Y,\tilde{x} \text{ exist s.t.} \\
& - fn(P,Q)\cup\{\tilde{x}\} \subseteq H_1 \\
& - n(H)\cap Y = \emptyset \\
& - \boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[P] \stackrel{\cdot}{\approx} \boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[Q] \qquad \}
\end{aligned}
$$

We show that $\mathcal{R}$ is weak early bisimulation up-to $\equiv$. Suppose $P \xrightarrow{\mu} P'$, with $\mu \neq \tau$ (the case $\mu = \tau$ is simpler). Without loss of generality, we can assume that the possible bound name of μ does not appear in H. Moreover, if V_1 and $V\langle H',Y'\rangle$ are the appropriate derivatives of $V\langle H,Y\rangle$, and $\tilde{z} = bn(\mu)$, we can infer

$$
\begin{array}{lll}
\boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[P] \longrightarrow\longrightarrow \equiv \boldsymbol{\nu}\,\tilde{x}\tilde{z}\,(P' \mid \overline{c}.V_1 \mid Count_{n+1}) & \stackrel{def}{=} & R_1 \\
\longrightarrow \quad \boldsymbol{\nu}\,\tilde{x}\tilde{z}\,(P' \mid V_1 \mid Count_{n+2}) & \stackrel{def}{=} & R_2 \\
\longrightarrow \quad \boldsymbol{\nu}\,\tilde{x}\tilde{z}\,(P' \mid V\langle H',Y'\rangle \mid Count_{n+2}) & \stackrel{def}{=} & R_3 = \boldsymbol{\nu}\,\tilde{x}\tilde{z}\,C_{n+2}^{H',Y'}[P']
\end{array}
$$

We want to find derivatives T_1, T_2 and T_3 of $\boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[Q]$ with which $\boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[Q]$ can match the above three moves by $\boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[P]$ and show that the structure of T_i, for $i = 1,2,3$ has strictly to mirror that of R_i. In particular T_3 must be structurally equivalent to a process of the form $\boldsymbol{\nu}\,\tilde{x}\tilde{z}\,(Q_3 \mid V\langle H', Y'\rangle \mid Count_{n+2})$, for some Q_3 s.t. $Q \stackrel{\mu}{\Longrightarrow} Q_3$.

We analyse only the first step; everything else can be easily adapted from the corresponding part in the proof of Theorem 3.3.2. We need a process T_1 s.t. $\boldsymbol{\nu}\,\tilde{x}\tilde{z}\,C_n^{H,Y}[Q] \Longrightarrow T_1 \stackrel{\cdot}{\approx} R_1$ and $T_1 \equiv \boldsymbol{\nu}\,\tilde{x}\tilde{z}\,(Q_1 \mid \overline{c}.V_1 \mid Count_{n+1})$, for some Q_1 s.t. $Q \stackrel{\mu}{\Longrightarrow} Q_1$. Process $\boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[Q]$ has to perform some move, because $\boldsymbol{\nu}\,\tilde{x}\,C_n^{H,Y}[P] \Downarrow_{d_n}$, but $R_1 \not\Downarrow_{d_n}$. But $R_1 \Downarrow_{d_{n+1}}$, hence $Count_n$ cannot participate in more than one interaction. Therefore T_1 must be of the form, up-to $\equiv$

$$\boldsymbol{\nu}\,\tilde{x}\,\boldsymbol{\nu}\,\tilde{z'}\left(Q_1 \mid \overline{c}.V_2 \mid Count_{n+1}\right)$$

for some $Q_1, V_2, \boldsymbol{\nu}\,\tilde{z'}$ s.t. $Q \stackrel{\mu'}{\Longrightarrow} Q_1$, $V\langle H, Y\rangle \stackrel{\mu''}{\longrightarrow}\stackrel{\overline{c}}{\longrightarrow} \overline{c}.V_2$, where μ'' is the complementary action of μ', and $\tilde{z'}$ collects the bound names of μ' or μ''. Suppose that μ is an input, say $\mu = a\langle b\rangle$. Since $R_1 \Longrightarrow\Downarrow_{\{a',b',in,d_{n+2}\}}$ and T_1 must do the same, μ' cannot be an output; moreover if also μ' is an input, say $\mu' = a_1\langle b_1\rangle$, then it must be $\{a,b\} = \{a_1,b_1\}$. The last possibility for having $\mu \neq \mu'$ is that $a \neq b$ and $a = b_1$, $b = a_1$: But then $R_1 \Longrightarrow\Downarrow_{\{b',in,d_{n+2}\}}$, which T_1 cannot match. The remaining cases for μ can be worked out similarly. Let us show however how the discrimination between μ and μ' is done when the former is a bound output and the latter a free output. This will explain the reason for the peculiar structure of the summand $(a2)$ in V. If μ is a bound output, say $\mu = (\boldsymbol{\nu}\,y)\overline{a}\langle y\rangle$, then since $y \notin n(H)$

$$\begin{aligned} V_1 &= \tau.W\langle a', y', ou\rangle + \tau.V\langle H^+, Y^-\rangle + \tau.\textstyle\sum_{(b,b')\in H}[y = b]\,b' \\ &\sim_{\mathrm{e}} \tau.W\langle a', y', ou\rangle + \tau.V\langle H^+, Y^-\rangle + \tau.\mathbf{0} \end{aligned}$$

Hence $R_1 \Longrightarrow\Downarrow_{\{y',a',ou,d_{n+2}\}}$. This forces V_2 to be a derivative of the summand $(a2)$ of V and μ' to be an output with subject a. Moreover, using the summand $\tau.\mathbf{0}$, also $R \Longrightarrow\Downarrow_{\{d_{n+2}\}}$, which forces μ' to be a bound output.

Appendix B

Proof of Proposition 4.2.6

We prove here Proposition 4.2.6, that is that $\simeq_{\mathrm{Ct}}$ is a congruence over substitution. In Chapter 4 we first stated Proposition 4.2.6 and then derived Theorem 4.2.7(1-5) from it. However, when we consider the proof, the ordering has to be reverted. That is to say, to prove the general case we have first to prove some specific instance of it.

In this appendix we shall use bold letters to distinguish open expressions. However, we might not use the bold case for an open agent A whose only free variable X is explicitly indicated (or known), like in $(X)A$ or $A\{F/X\}$. We will be working in the reduced HOπ of section 4.1, in which only unary abstractions are permitted. But for convenience, we shall use t-ary abstractions in a few places.

Lemma B.0.23

1. $(a\,X)\,\boldsymbol{A_1} \simeq_{\mathrm{Ct}} (a\,X)\,\boldsymbol{A_2}$ *iff* $(X\,a)\,\boldsymbol{A_1} \simeq_{\mathrm{Ct}} (X\,a)\,\boldsymbol{A_2}$;
2. $\boldsymbol{A_1} \simeq_{\mathrm{Ct}} \boldsymbol{A_2}$ *iff* $(X)\boldsymbol{A_1} \simeq_{\mathrm{Ct}} (X)\boldsymbol{A_2}$.

Theorem 4.2.7(1-5) $\boldsymbol{P_1} \simeq_{\mathrm{Ct}} \boldsymbol{P_2}$ *implies*

1. $\boldsymbol{\nu}\, a\, \boldsymbol{P_1} \simeq_{\mathrm{Ct}} \boldsymbol{\nu}\, a\, \boldsymbol{P_2}$;
2. $\boldsymbol{P_1} \mid \boldsymbol{R} \simeq_{\mathrm{Ct}} \boldsymbol{P_2} \mid \boldsymbol{R}$;

3. $!\boldsymbol{P_1} \simeq_{\mathrm{Ct}} !\boldsymbol{P_2}$;

4. $[a=b]\boldsymbol{P_1} \simeq_{\mathrm{Ct}} [a=b]\boldsymbol{P_2}$;

5. $\boldsymbol{\alpha}.\boldsymbol{P_1} + \boldsymbol{R} \simeq_{\mathrm{Ct}} \boldsymbol{\alpha}.\boldsymbol{P_2} + \boldsymbol{R}$, *if* $\boldsymbol{\alpha}$ *is not a first-order input;*

where, in clause (5), the subterm $+R$ *can also be missing from the left and right members.*

PROOF: We only look at (2). It is enough to prove the result on closed expressions. We shall do so by proving that

$$\mathcal{R} = \{(P_1 \mid R, P_2 \mid R) \;:\; P_1 \simeq_{\mathrm{Ct}} P_2\}$$

is a bisimulation up-to $\equiv$ and up-to restriction. Take $(P_1 \mid R, P_2 \mid R) \in \mathcal{R}$ and suppose $P_1 \mid R \xrightarrow{\mu} Q_1$. There are four cases to consider. Remember that since $P_1 \simeq_{\mathrm{Ct}} P_2$,

if $P_1 \xrightarrow{(\nu \widetilde{b})\overline{a}\langle F\rangle} P_1'$, there exist P_2', $\widetilde{c}$, E, s.t. $P_2 \xrightarrow{(\nu \widetilde{c})\overline{a}\langle E\rangle} P_2'$ (∗)
and for each G with $fn(G) \cap (\widetilde{b} \cup \widetilde{c}) = \emptyset$,
$\boldsymbol{\nu}\, \widetilde{b}\big(G\langle F\rangle \mid P_1'\big) \simeq_{\mathrm{Ct}} \boldsymbol{\nu}\, \widetilde{c}\big(G\langle E\rangle \mid P_2'\big)$.

Case 2.1 $P_1 \xrightarrow{\mu} P_1'$ and $Q_1 = P_1' \mid R$.

The cases when μ is an input or a τ action are pure routine. The case when μ is an output can be worked out using (∗) and simple manipulations in term of $\equiv$.

Case 2.2 $R \xrightarrow{\mu} R'$ and $Q_1 = P_1 \mid R'$.

Similar.

Case 2.3 $P_1 \xrightarrow{(\nu \widetilde{b})\overline{a}\langle F\rangle} P_1'$, $R \xrightarrow{a\langle F\rangle} R'$ and $Q_1 = \boldsymbol{\nu}\, \widetilde{b}\,(P_1' \mid R')$

If G is the abstraction which Lemma 4.1.3 associates to the action $R \xrightarrow{a\langle F\rangle} R'$, it holds that $R' = G\langle F\rangle$. Then for $P_2', \widetilde{c}, E$, as defined in (∗), we have

$$R \xrightarrow{a\langle E\rangle} G\langle E\rangle \text{ and hence } P_2 \mid R \xrightarrow{\tau} \boldsymbol{\nu}\, \widetilde{c}\,(P_2' \mid G\langle E\rangle).$$

Now, using (∗), we get $\boldsymbol{\nu}\,\tilde{b}(P_1' \mid G\langle F\rangle) \simeq_{\mathrm{Ct}} \boldsymbol{\nu}\,\tilde{c}(P_2' \mid G\langle E\rangle)$. Thus

$$\boldsymbol{\nu}\,\tilde{b}(P_1' \mid R') \equiv \boldsymbol{\nu}\,\tilde{b}(P_1' \mid G\langle F\rangle) \mid \mathbf{0}\ \mathcal{R}\ \boldsymbol{\nu}\,\tilde{c}(P_2' \mid G\langle E\rangle) \mid \mathbf{0} \equiv \boldsymbol{\nu}\,\tilde{c}(P_2' \mid G\langle E\rangle).$$

which is enough, because $\mathcal{R}$ is a bisimulation up-to $\equiv$.

Case 2.4 $P_1 \xrightarrow{a\langle F\rangle} P_1'$, $R \xrightarrow{(\boldsymbol{\nu}\,\tilde{b})\overline{a}\langle F\rangle} R'$ and $Q_1 \stackrel{def}{=} \boldsymbol{\nu}\,\tilde{b}(P_1' \mid R')$.

By definition of $\simeq_{\mathrm{Ct}}$, $P_2 \xrightarrow{a\langle F\rangle} P_2' \simeq_{\mathrm{Ct}} P_1'$; therefore $P_2 \mid R \xrightarrow{\tau} \boldsymbol{\nu}\,\tilde{b}(P_2' \mid R')$. This is enough, because $\mathcal{R}$ is a bisimulation up-to restriction. □

The previous theorem gives results to which we are more or less used in process algebra. Next result is by far more delicate. It could be rephrased by saying that $\simeq_{\mathrm{Ct}}$ is a congruence w.r.t. application on the right (that is, if $F_1 \simeq_{\mathrm{Ct}} F_2$, then $G\langle F_1\rangle \simeq_{\mathrm{Ct}} G\langle F_2\rangle$). The λ-calculus teaches us that congruence results of this kind can be very difficult. In our case, well-sortness is what makes the proof possible (in the same way in λ-calculus the reasoning can be facilitated by the use of types).

We remind that if $\boldsymbol{A}$ is an agent of sort S, then the *sort depth of* $\boldsymbol{A}$, briefly $sd(\boldsymbol{A})$, is the level of bracket nesting in S, and says how "high order" $\boldsymbol{A}$ is. Further, a variable X is *guarded in* $\boldsymbol{A}$ if each free occurrence of X in $\boldsymbol{A}$ is within some subexpression $\boldsymbol{\alpha}.\boldsymbol{Q}$ of $\boldsymbol{A}$. For instance, X is guarded in $\overline{a}\langle X\rangle.(P \mid X)$, but it is not guarded in $(\overline{a}\langle X\rangle.P) \mid X)$.

Proposition 4.2.6 $\boldsymbol{F_1} \simeq_{\mathrm{Ct}} \boldsymbol{F_2}$ *implies* $\boldsymbol{A}\{\boldsymbol{F_1}/X\} \simeq_{\mathrm{Ct}} \boldsymbol{A}\{\boldsymbol{F_2}/X\}$.

PROOF: We use induction on

$$\mathcal{VD}(\boldsymbol{A}) = max\Big\{\{sd(Y)\ :\ Y \in fv(\boldsymbol{A})\} \cup \{sd(\boldsymbol{A}) - 1\}\Big\}$$

Basic case: $\mathcal{VD}(\boldsymbol{A}) = 0$. That is, $\boldsymbol{A}$ is a process or a first-order abstraction and $\boldsymbol{A}$ does not contain free variables. There is nothing to prove.

Inductive case: $\mathcal{VD}(\boldsymbol{A}) = n + 1$. By definition of $\simeq_{\mathrm{Ct}}$ on open agents, it is enough to prove the result when $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ are closed; therefore, in the following, they will be written as F_1 and F_2 (i.e., not in bold). Let us use $\tilde{F}$ and $\widetilde{X}$ as

abbreviations for the tuples F_1, F_2 and X_1, X_2, respectively. We first prove an auxiliary lemma which, intuitively, allows us to use F_1 and F_2 interchangeably in all unguarded occurrences of X in $\boldsymbol{A}$ (this is clear by setting, in the assertion of the lemma, $\tilde{a} = \emptyset$, $\boldsymbol{A_1} = \boldsymbol{A_2} = \boldsymbol{A}$ and $X_1 = X$; then $\boldsymbol{A_2}$ is obtained from $\boldsymbol{A_1}$ by redenominating the unguarded occurrences of X to X_2).

Lemma AUX Suppose that $\mathcal{VD}(\boldsymbol{A_1}) \leq n+1$, that $X_1 \in fv(\boldsymbol{A_1})$, and that $X_2 \notin fv(\boldsymbol{A_1})$. Then there exists $\boldsymbol{A_2}$ such that

1. X_1 is guarded in $\boldsymbol{A_2}$;
2. $\boldsymbol{A_2}\{X_1/X_2\} = \boldsymbol{A_1}$;
3. if $F_1 \simeq_{\mathrm{Ct}} F_2$, and $\tilde{a} \cap fv(F_1, F_2) = \emptyset$, then
   $(\tilde{a})(\boldsymbol{A_1}\{F_1/X_1\}) \simeq_{\mathrm{Ct}} (\tilde{a})(\boldsymbol{A_2}\{\tilde{F}/\widetilde{X}\})$.

PROOF: By induction on the structure of $\boldsymbol{A_1}$. By η-conversion, we can assume that if $\boldsymbol{A_1}$ is an abstraction, it is of the form $(X)\boldsymbol{P}$ or $(x)\boldsymbol{P}$. The basis of the induction occurs when $\boldsymbol{A_1} = \boldsymbol{\alpha}.\boldsymbol{P}$. For this, take $\boldsymbol{A_2} \stackrel{def}{=} \boldsymbol{A_1}$. For the inductive cases, as (1) and (2) will always be trivial to verify, we only examine (3).

When $\boldsymbol{A_1}$ is a process of the form $\boldsymbol{P_1}|\boldsymbol{P_2}$, $\sum_{i\in I}\boldsymbol{P_i}$, $\boldsymbol{\nu}\, a\, \boldsymbol{P}$, $[a=b]\boldsymbol{P}$ or $!\boldsymbol{P}$, use the inductive hypothesis on the $\boldsymbol{P_i}$'s and Theorem 4.2.7 (remember that in the reduced HOπ we are using the indexing set I is finite).

The case $\boldsymbol{A_1} = (a)\boldsymbol{P_1}$ is straightforward. The remaining cases are more delicate.

Suppose $\boldsymbol{A_1} = (Y)\boldsymbol{P_1}$. In order to use the inductive hypothesis of the lemma on $\boldsymbol{P_1}$, we need to insure that $\mathcal{VD}(\boldsymbol{P_1}) \leq n+1$. Being $\boldsymbol{P_1}$ a process, we have $sd(\boldsymbol{P_1})-1 = 0$; so we only have to check that for each $X' \in fv(\boldsymbol{P_1})$, it holds that $sd(X') \leq n+1$. For $X' \neq Y$ this is true because $X' \in fv(\boldsymbol{A_1})$ and $\mathcal{VD}(\boldsymbol{A_1}) \leq n+1$. Suppose $X' = Y$: From

$\mathcal{VD}(\boldsymbol{A_1}) \leq n+1$ we get $sd(\boldsymbol{A_1}) \leq n+2$ and then, since $\boldsymbol{A_1} = (Y)\boldsymbol{P_1}$, we have $sd(Y) = sd(\boldsymbol{A_1}) - 1 \leq n+1$.

Hence, let $\boldsymbol{P_2}$ be the process obtained from $\boldsymbol{P_1}$ using the inductive hypothesis; by the assertion (3) of the lemma, we have

$$(\widetilde{a})\big(\boldsymbol{P_1}\{F_1/X_1\}\big) \simeq_{\mathrm{Ct}} (\widetilde{a})\big(\boldsymbol{P_2}\{\widetilde{F}/\widetilde{X}\}\big).$$

Using Lemma B.0.23, also

$$(\widetilde{a}\,Y)\left(\boldsymbol{P_1}\{F_1/X_1\}\right) \simeq_{\mathrm{Ct}} (\widetilde{a}\,Y)\left(\boldsymbol{P_2}\{\widetilde{F}/\widetilde{X}\}\right)$$

For $\boldsymbol{A_2} \stackrel{def}{=} (Y)\boldsymbol{P_2}$, this proves the assertion (3).

Suppose now $\boldsymbol{A_1}$: () and defined by an expression which has a variable Y in head position. We have to distinguish the cases $sd(Y) = 1$ and $sd(Y) > 1$. If $sd(Y) = 1$, then $\boldsymbol{A_1} = Y\langle a\rangle$. For this, take $\boldsymbol{A_2} \stackrel{def}{=} X_2\langle a\rangle$ if $X_1 = Y$, and $\boldsymbol{A_2} \stackrel{def}{=} \boldsymbol{A_1}$ if $Y \neq X_1$.

If $sd(Y) > 1$, then $\boldsymbol{A_1} = Y\langle \boldsymbol{E_1}\rangle$, for some $\boldsymbol{E_1}$. We shall only show the proof for $Y = X_1$; the case $Y \neq X_1$ is simpler. It is easy to check that $\mathcal{VD}(\boldsymbol{E_1}) \leq n+1$. Hence, let $\boldsymbol{E_2}$ be the abstraction which the inductive hypothesis of the lemma associates to $\boldsymbol{E_1}$. By the assertion (3) we have, for $\widetilde{a} \cap fn(F_1, F_2) = \emptyset$:

$$\boldsymbol{K_1} \stackrel{def}{=} (\widetilde{a})\big(\boldsymbol{E_1}\{F_1/X_1\}\big) \simeq_{\mathrm{Ct}} (\widetilde{a})\big(\boldsymbol{E_2}\{\widetilde{F}/\widetilde{X}\}\big) \stackrel{def}{=} \boldsymbol{K_2}.$$

Therefore, by definition of $\simeq_{\mathrm{Ct}}$ over abstractions, for every $\widetilde{b} : \widetilde{a}$,

$$\boldsymbol{K_1}\langle\widetilde{b}\rangle \simeq_{\mathrm{Ct}} \boldsymbol{K_2}\langle\widetilde{b}\rangle \tag{B.1}$$

Define $\boldsymbol{A_2} \stackrel{def}{=} X_2\langle \boldsymbol{E_2}\rangle$. If

$$\boldsymbol{H_1} \stackrel{def}{=} F_1\left\langle \boldsymbol{E_1}\{F_1/X_1\}\right\rangle \qquad \text{and} \qquad \boldsymbol{H_2} \stackrel{def}{=} F_2\left\langle \boldsymbol{E_2}\{\widetilde{F}/\widetilde{X}\}\right\rangle,$$

we have to show that $(\widetilde{a})\boldsymbol{H_1} \simeq_{\mathrm{Ct}} (\widetilde{a})\boldsymbol{H_2}$. This is true if for every $\widetilde{b} : \widetilde{a}$, it holds that

$$\boldsymbol{H_1}\{\widetilde{b}/\widetilde{a}\} = F_1\left\langle \boldsymbol{K_1}\langle\widetilde{b}\rangle\right\rangle \simeq_{\mathrm{Ct}} F_2\left\langle \boldsymbol{K_2}\langle\widetilde{b}\rangle\right\rangle = \boldsymbol{H_2}\{\widetilde{b}/\widetilde{a}\} \tag{B.2}$$

If we knew that

$$F_1\langle \boldsymbol{K_1}\langle \tilde{b}\rangle\rangle \simeq_{\mathrm{Ct}} F_1\langle \boldsymbol{K_2}\langle \tilde{b}\rangle\rangle \tag{B.3}$$

holds, then we would get (B.2) as follows:

$$\begin{aligned} F_1\langle \boldsymbol{K_1}\langle \tilde{b}\rangle\rangle &\simeq_{\mathrm{Ct}} \quad \text{(by (B.3))} \\ F_1\langle \boldsymbol{K_2}\langle \tilde{b}\rangle\rangle &\simeq_{\mathrm{Ct}} \quad \text{(because } F_1 \simeq_{\mathrm{Ct}} F_2\text{)} \\ F_2\langle \boldsymbol{K_2}\langle \tilde{b}\rangle\rangle & \end{aligned}$$

So, it remains to prove (B.3). For this, we use the inductive hypothesis of Proposition 4.2.6. By η-conversion, we can assume that the higher-order abstraction F_1 is of the form $F_1 = (X')P$, for some P. Therefore we have

$$F_1\langle \boldsymbol{K_1}\langle \tilde{b}\rangle\rangle = P\{\boldsymbol{K_1}\langle \tilde{b}\rangle/X'\} \quad \text{and}$$

$$F_1\langle \boldsymbol{K_2}\langle \tilde{b}\rangle\rangle = P\{\boldsymbol{K_2}\langle \tilde{b}\rangle/X'\}$$

Now, $\boldsymbol{K_1}\langle \tilde{b}\rangle \simeq_{\mathrm{Ct}} \boldsymbol{K_2}\langle \tilde{b}\rangle$ holds from (B.1). Moreover, it holds that $\mathcal{VD}(P) \leq n$ as the only free variable which might occur in P is X' and $sd(X') \leq n$ follows from $sd(F_1) = sd(X_1) \leq n+1$. Therefore we can apply the inductive hypothesis of Proposition 4.2.6 and conclude that $P\{\boldsymbol{K_1}\langle \tilde{b}\rangle/X'\} \simeq_{\mathrm{Ct}} P\{\boldsymbol{K_2}\langle \tilde{b}\rangle/X'\}$. □

Now using this Lemma Aux we can prove the inductive case of Proposition 4.2.6. By definition of $\simeq_{\mathrm{Ct}}$ on open agents and abstractions, it is enough to show the result when $\boldsymbol{A}\{F_1/X\}$ and $\boldsymbol{A}\{F_2/X\}$ are closed processes. To match the notation of Lemma Aux, let us set $X = X_1$ and $\boldsymbol{A} = P_1$. Thus, we have to show that:

$$P_1\{F_1/X_1\} \simeq_{\mathrm{Ct}} P_1\{F_2/X_1\} \tag{B.4}$$

where $sd(X_1) \leq n+1$. For this, we shall need Lemma Aux together with the following equivalence result: The relation

$$\mathcal{R} = \{(P_1\{F_2/X_1\}, P_1\{F_1/X_1\})$$
$$F_2 \simeq_{\mathrm{Ct}} F_1,\ \ X_1 \ \textit{guarded in}\ P_1\ \textit{and}\ sd(X_1) \leq n+1\}$$

is a bisimulation up-to $\simeq_{\mathrm{Ct}}$. Let $P_1\{F_2/X_1\}\ \mathcal{R}\ P_1\{F_1/X_1\}$. We show that the actions of $P_1\{F_2/X_1\}$ can be matched by $P_1\{F_1/X_1\}$. Since X_1 is guarded in P_1, it is enough to look at P_1 to infer the possible moves for $P_1\{F_2/X_1\}$ and $P_1\{F_1/X_1\}$. With some abuse of language, as P_1 might contain free occurrences of X_1, we write $P_1 \xrightarrow{\mu} P_1'$ to mean that whenever X_1 in P_1 is instantiated with an abstraction E, then $P_1\{E/X_1\} \xrightarrow{\mu\{E/X_1\}} P_1'\{E/X_1\}$. There are two cases:

Case (a) $P_1 \xrightarrow{\mu} P_1'$ and μ is an input or a silent move. Then

$$P_1\{F_2/X_1\} \xrightarrow{\mu} P_1'\{F_2/X_1\},\ \text{ and }\ P_1\{F_1/X_1\} \xrightarrow{\mu} P_1'\{F_1/X_1\}$$

As $\mathcal{VD}(P_1') \leq n+1$, we can apply Lemma Aux to P_1'; let P_2' be the abstraction returned. We have that

$$\begin{array}{rcll} P_1'\{F_2/X_1\} & = & \text{(by Lemma Aux(2))} & \text{(B.5)} \\ P_2'\{F_2, F_2/X_2, X_1\} & \mathcal{R} & (X_1 \text{ is guarded in } P_2' \text{ by Lemma Aux(1))} & \\ P_2'\{F_2, F_1/X_2, X_1\} & \simeq_{\mathrm{Ct}} & \text{(by Lemma Aux(3))} & \\ P_1'\{F_1/X_1\} & & & \end{array}$$

i.e. summarising, $P_1'\{F_2/X_1\}\ \mathcal{R}\ \simeq_{\mathrm{Ct}} P_1'\{F_1/X_1\}$, which closes up the bisimulation.

Case (b) $P \xrightarrow{(\nu \tilde{b})\overline{a}\langle E\rangle} P_1'$ (where E might contain free occurrences of X_1)

We need to ensure that for every G, with $\tilde{b} \cap fn(G) = \emptyset$

$$\big(\boldsymbol{\nu}\, \tilde{b}(G\langle E\rangle \mid P_1')\big)\{F_2/X_1\}\ \mathcal{R}\ \simeq_{\mathrm{Ct}} \big(\boldsymbol{\nu}\, \tilde{b}(G\langle E\rangle \mid P_1')\big)\{F_1\ X_1\}$$

This can be done exploiting Lemma Aux in the same way as we did in (B.5).

Having proved Lemma Aux and $\mathcal{R} \subseteq \simeq_{\mathrm{Ct}}$, we can now derive (B.4) and conclude the proof. Proceeding as in (B.5), the only difference being that now we can use $\simeq_{\mathrm{Ct}}$ instead of $\mathcal{R}$, for some P_2 we have:

$$P_1\{F_2/X_1\} = P_2\{F_2, F_2/X_2, X_1\} \simeq_{\mathrm{Ct}} P_2\{F_2, F_1/X_2, X_1\} \simeq_{\mathrm{Ct}} P_1\{F_1/X_1\}$$

Appendix C

Proof of Proposition 4.3.2

In this appendix, we prove Proposition 4.2.6, that is that $\{m := F\}$ distributes over substitution, on the hypothesis that m is used only in negative subject position. The presentation of this appendix is similar to the one for Appendix B, where we inferred congruence for $\simeq_{\mathrm{Ct}}$ over substitution. In both cases, we have first to prove some instance (Theorems 4.2.7 and 4.3.3, respectively) of the general result (Propositions 4.2.6 and 4.3.2, respectively). Moreover, the proof of Proposition 4.3.2 follows very closely the proof of Proposition 4.2.6.

We maintain the notation and the terminology introduced in Appendix B. In addition, we write "a nsp $\boldsymbol{A}$" to mean that the name a occurs free in the agent $\boldsymbol{A}$ only in negative subject position.

Our first target is to show that $\{m := F\}$ distributes over parallel composition. We cannot quite follow the proof technique used by Milner to show the analogous result for the π-calculus ([Mil91, Section 5.4]), due to the higher-order setting in which we are working. For our proof, Lemmas C.0.24 and C.0.25 are needed. Lemma C.0.24 relates the actions of the processes Q and Q', where Q' is obtained from Q through a substitution of one of its names. In the lemma, assertion (2) — the vice versa of (1) — is only possible because of the side condition x, m nsp Q which prevents new possible interactions from being generated as effect of the substitution $Q\{m/x\}$. The restrictions (a), (b), (c), on the actions α, β are imposed

to maintain the side conditions on the names x, m in the descendants of Q and Q'.

Lemma C.0.24 *Suppose* x, m nsp Q *and let* μ, μ' *be two actions satisfying the conditions:*

(a) $x \notin n(\mu')$;

(b) *if* μ *or* μ' *is of the form* $a\langle b\rangle$ *(first-order input), then* $x, m \neq b$;

(c) *if* μ *or* μ' *is of the form* $a\langle G\rangle$ *(higher-order input), then* x, m nsp G.

Then it holds that:

1. *if* $Q \xrightarrow{\mu} Q'$, *then* $Q\{m/x\} \overset{\mu\{m/x\}}{\longrightarrow} Q'\{m/x\}$.
2. *if* $Q\{m/x\} \xrightarrow{\mu'} Q''$, *then there exists* Q' *s.t.*
   $Q \xrightarrow{\mu} Q'$ *with* $\mu' = \mu\{m/x\}$ *and* $Q'' = Q'\{m/x\}$.

Moreover both in (1) and and in (2), it holds that x, m nsp Q'.

PROOF: It is an easy (slightly tedious) transition induction, both for (1) and for (2). □

Lemma C.0.25 *Suppose that* x, m_1 nsp Q, *that* $x, m_1 \notin fn(F)$ *and that* $m_2 \notin fn(Q, F)$. *Then*

$$Q\{m_1/x\}\ \{m_1 := F\} \simeq^{c}_{\mathrm{Ct}} Q\{m_2/x\}\ \{m_1 := F\}\ \{m_2 := F\}.$$

PROOF: Let us write $\boldsymbol{\nu}\,\widetilde{m}$, $!\,m\tilde{\star}F$ and $P\,\{\widetilde{m} := \widetilde{F}\}$ for $\boldsymbol{\nu}\,m_1\,m_2$, $!\,m_1 \star F \mid !\,m_2 \star F$ and $P\,\{m_1 := F\}\,\{m_2 := F\}$, respectively. Given the universal quantification on the processes in the assertion of the lemma, it is enough to prove the result for $\simeq_{\mathrm{Ct}}$. For this, since $Q\{m_1/x\}\ \{m_1 := F\} \simeq^{c}_{\mathrm{Ct}} Q\{m_1/x\}\ \{\widetilde{m} := \widetilde{F}\}$ (this follows from Lemma 4.4.4(2), for m_2 does not occur free in $Q\{m_1/x\}\ \{m_1 := F\}$), we show that

$$\mathcal{R} = \{(Q\{m_1/x\}\,\{\widetilde{m} := \widetilde{F}\}, Q\{m_2/x\}\,\{\widetilde{m} := \widetilde{F}\}) : \\ x, m_1 \text{ nsp } Q,\ m_1, x \notin fn(F) \text{ and } m_2 \notin fn(Q, F)\}$$

is a $\simeq_{\mathrm{Ct}}$-bisimulation up-to $\equiv$. Suppose $P_1\ \mathcal{R}\ P_2$, for

$$P_1 \stackrel{def}{=} Q\{m_1/x\}\,\{\widetilde{m} := \widetilde{F}\}, \qquad P_2 \stackrel{def}{=} Q\{m_2/x\}\,\{\widetilde{m} := \widetilde{F}\}.$$

We have to consider the actions of $Q\{m_1/x\}$ and $Q\{m_2/x\}$ which can cause an action of either P_1 or P_2. In the following we make use of Lemma C.0.24 which tells us how to infer the actions for $Q\{m_1/x\}$ and $Q\{m_2/x\}$ from those of Q.

Case (a) $Q \xrightarrow{\tau} Q'$.

We get the actions

$$P_1 \xrightarrow{\tau} Q'\{m_1/x\}\,\{\widetilde{m} := \widetilde{F}\}, \qquad P_2 \xrightarrow{\tau} Q'\{m_2/x\}\,\{\widetilde{m} := \widetilde{F}\}.$$

Now we are back in $\mathcal{R}$ because Lemma C.0.24 insures that Q' meets the side conditions of $\mathcal{R}$.

Case (b) $Q \xrightarrow{a\langle E\rangle} Q'$.

By the side conditions on x, m_1 it must be $a \notin \{x, m_1\}$. We can assume that $x, m_1, m_2 \notin (E)$ and then we get the actions:

$$P_1 \xrightarrow{a\langle E\rangle} Q'\{m_1/x\}\,\{\widetilde{m} := \widetilde{F}\} \quad \text{and } P_2 \xrightarrow{a\langle E\rangle} Q'\{m_2/x\}\,\{\widetilde{m} := \widetilde{F}\}$$

and as for case (a), we are back in $\mathcal{R}$.

Case (c) $Q \xrightarrow{(\nu\widetilde{b})\overline{a}\langle E\rangle} Q'$, with $a \notin \{x, m_1\}$.

The names x, m_1 might occur free in E. We consider only the case when both of them are free in E, as the others — modulo structural congruence — can be worked out in the same way. Since by α-convertibility we can assume $\widetilde{b} \cap fn(F) = \emptyset$, we get the actions

$$P_1 \xrightarrow{(\nu\, m_1\, \widetilde{b})\overline{a}\langle E\{m_1/x\}\rangle} \boldsymbol{\nu}\, m_2\, (Q'\{m_1/x\} \mid !\, m\tilde{\star}F) \stackrel{def}{=} P_3 \quad \text{and}$$
$$P_2 \xrightarrow{(\nu\, \widetilde{m}\, \widetilde{b})\overline{a}\langle E\{m_2/x\}\rangle} Q'\{m_2/x\} \mid !\, m\tilde{\star}F \stackrel{def}{=} P_4.$$

By definitions of $\simeq_{\mathrm{Ct}}$ and of $\mathcal{R}$, we have to insure that for every G with $fn(G) \cap (\widetilde{b} \cup \widetilde{m}) = \emptyset$, it holds that

$$P_5 \stackrel{def}{=} \boldsymbol{\nu}\, m_1\, \widetilde{b}\big(G\langle E\{m_1/x\}\rangle \mid P_3\big) \equiv \mathcal{R} \equiv \boldsymbol{\nu}\, \widetilde{m}\, \widetilde{b}\big(G\langle E\{m_2/x\}\rangle \mid P_4\big) \stackrel{def}{=} P_6.$$

With simple manipulations we get:

$$P_5 \equiv \big(\boldsymbol{\nu}\, \widetilde{b}(G\langle E\rangle \mid Q')\big)\{m_1/x\}\ \{\widetilde{m} := \widetilde{F}\}$$

and similarly,

$$P_6 \equiv \big(\boldsymbol{\nu}\, \widetilde{b}(G\langle E\rangle \mid Q')\big)\{m_2/x\}\ \{\widetilde{m} := \widetilde{F}\}.$$

Now we have that $P_5\ \mathcal{R}\ P_6$, if we can show that x, m_1 nsp $\boldsymbol{\nu}\, \widetilde{b}(G\langle E\rangle \mid Q')$ and $m_2 \notin fn\big(\boldsymbol{\nu}\, \widetilde{b}(G\langle E\rangle \mid Q')\big)$. This is trivially true because:

- by α-conversion, we can assume $x, m_1, m_2 \notin fn(G)$;
- being E a subcomponent of Q, we have x, m_1 nsp E, $m_2 \notin fn(E)$;
- from m_1 nsp Q and $m_2 \notin fn(Q)$, also $m_2 \notin fn(Q')$, and m_1 nsp Q'.

Case (d) $Q \xrightarrow{(\nu\, \widetilde{b})\overline{a}\langle E\rangle} Q'$, with $a \in \{x, m_1\}$.

We assume that $a = x$, as the case $a = m_1$ is similar. For this case we get a communication between $Q\{m_1/x\}$ and $!\, m_1 \star F$ in P_1, and a communication between $Q\{m_2/x\}$ and $!\, m_2 \star F$ in P_2:

$$P_1 \xrightarrow{\tau} \equiv \big(\boldsymbol{\nu}\, \widetilde{b}(F\langle E\{m_1/x\}\rangle \mid Q'\{m_1/x\})\big)\ \{\widetilde{m} := \widetilde{F}\} \quad \text{and}$$
$$P_2 \xrightarrow{\tau} \equiv \big(\boldsymbol{\nu}\, \widetilde{b}(F\langle E\{m_2/x\}\rangle \mid Q'\{m_2/x\})\big)\ \{\widetilde{m} := \widetilde{F}\}.$$

We now are back in $\mathcal{R}$, as its side conditions holds for $\boldsymbol{\nu}\, \widetilde{b}(F\langle E\rangle \mid Q')$. □

Theorem 4.3.3(1-5) *Suppose that* $m \notin fn(F)$ *and that* m nsp $\boldsymbol{P_i}, \boldsymbol{P}, \boldsymbol{Q}$. *Then*

1. $\left(\sum_{i\in I} \boldsymbol{P_i}\right)\{m := F\} \simeq^{c}_{\mathrm{Ct}} \sum_{i\in I}\left(\boldsymbol{P_i}\{m := F\}\right)$.
2. $\left(\boldsymbol{\nu}\, a\, \boldsymbol{P}\right)\{m := F\} \simeq^{c}_{\mathrm{Ct}} \boldsymbol{\nu}\, a\left(\boldsymbol{P}\{m := F\}\right)$, $a \neq m$.
3. $\left(\boldsymbol{P} \mid \boldsymbol{Q}\right)\{m := F\} \simeq^{c}_{\mathrm{Ct}} \boldsymbol{P}\{m := F\} \mid \boldsymbol{Q}\{m := F\}$.
4. $(!\boldsymbol{P})\{m := F\} \simeq^{c}_{\mathrm{Ct}} !(\boldsymbol{P}\{m := F\})$.
5. $([a = b]\boldsymbol{P})\{m := F\} \simeq^{c}_{\mathrm{Ct}} [a = b](\boldsymbol{P}\{m := F\})$.

PROOF: The hardest cases are (3) and (4), and we only consider these. It is enough to show the result for $\simeq_{\mathrm{Ct}}$ and w.r.t. closed agents. We start with (3).

Let $x \notin fn(P, Q)$ and $Q' \stackrel{def}{=} Q\{x/m\}$, i.e. in Q' all the free occurrences of m are substituted with the name x. Then we have $Q'\{m/x\} = Q$. Moreover, if $m' \notin fn(P, Q, F)$ we have

$$\begin{array}{rcl}
\left(P \mid Q\right)\{m := F\} & = & \\
(P \mid Q')\{m/x\}\{m := F\} & \simeq_{\mathrm{Ct}} & \text{(by Lemma C.0.25)} \\
(P \mid Q')\{m'/x\}\{m := F\}\{m' := F\} & = & \\
(P \mid Q'\{m'/x\})\{m := F\}\{m' := F\} & \equiv & \text{(as } m \notin fn(Q'\{m'/x\}, F) \\
 & & \text{and } m' \notin fn(P, F)) \\
\left(P\{m := F\}\right) \mid \left(Q'\{m'/x\}\{m' := F\}\right) & = & \text{(using } \alpha\text{-conversion)} \\
P\{m := F\} \mid Q\{m := F\} & &
\end{array}$$

Now the assertion (4). We show that

$$\mathcal{R} = \{(Q \mid (!P)\{m := F\},\ Q \mid !(P\{m := F\})) \ : \ m \notin fn(F), m \text{ nsp } P\}$$

is a $\simeq_{\mathrm{Ct}}$-bisimulation up-to $\simeq_{\mathrm{Ct}}$ and up-to restriction. An action of

$$Q \mid (!P)\{m := F\} \qquad \text{or} \qquad Q \mid !(P\{m := F\})$$

can come from an action of Q alone, an action of P alone, a communication between Q and P, or a communication between P and $\{m := F\}$. We analise the cases of output action by P and of communication between P and $\{m := F\}$. The other cases are easier and can be dealt with in a similar way. Suppose $P \stackrel{\nu \widetilde{b}\,\overline{a}\langle E\rangle}{\longrightarrow} P'$, with $a \neq m$. Since m nsp P, also m nsp E. Assuming m does indeed occur free in E, we have:

$$Q \mid (!\,P)\,\{m := F\} \stackrel{(\nu\, m,\widetilde{b})\overline{a}\langle E\rangle}{\longrightarrow} Q \mid P' \mid !\,P \mid !\,m \star F \stackrel{def}{=} H$$

and

$$Q \mid !\,(P\,\{m := F\}) \stackrel{(\nu\, m,\widetilde{b})\overline{a}\langle E\rangle}{\longrightarrow} Q \mid P' \mid !\,m \star F \mid (!\,P\,\{m := F\}) \stackrel{def}{=} K$$

By definition of $\simeq_{\mathrm{Ct}}$ and $\mathcal{R}$, to close up the bisimulation, we have to show that H and K, together with the recipient G of E, give processes in the relation $\mathcal{R}$, up-to $\simeq_{\mathrm{Ct}}$ and up-to restriction. By α-conversion, we can assume that $fn(G) \cap (m \cup \widetilde{b}) = \emptyset$. Then we have

$$\begin{aligned}
\boldsymbol{\nu}\,\widetilde{b}\,(G\langle E\rangle \mid H) \quad &= \\
\boldsymbol{\nu}\,m,\widetilde{b}\,\big(G\langle E\rangle \mid Q \mid P' \mid !\,P \mid !\,m \star F\big) \quad &\equiv \\
\boldsymbol{\nu}\,\widetilde{b}\,\big(Q \mid (G\langle E\rangle \mid P' \mid !\,P)\,\{m := F\}\big) \quad &\simeq_{\mathrm{Ct}} \quad \text{by Theorem 4.3.3(3)} \\
\boldsymbol{\nu}\,\widetilde{b}\,\big(Q \mid (G\langle E\rangle \mid P')\,\{m := F\} \mid (!\,P)\,\{m := F\}\big) &
\end{aligned}$$

and

$$\boldsymbol{\nu}\,m,\widetilde{b}\,(G\langle E\rangle \mid K) \;\equiv\; \boldsymbol{\nu}\,\widetilde{b}\,\big(Q \mid (G\langle E\rangle \mid P')\,\{m := F\} \mid !\,(P\,\{m := F\})\big)$$

This is enough, since

$$Q|(G\langle E\rangle|P')\{m := F\}|(!\,P)\{m := F\} \quad \text{and} \quad Q|(G\langle E\rangle|P')\{m := F\}|!\,(P\{m := F\})$$

are in the relation $\mathcal{R}$.

Now we look at the case in which the action of P occurs at the port m, and therefore P interacts with $\{m := F\}$. Suppose $P \stackrel{\nu \widetilde{b}\,\overline{m}\langle E\rangle}{\longrightarrow} P'$ is the action of P.

We have

$$Q \mid (!\,P)\,\{m := F\} \xrightarrow{\tau} \equiv$$
$$Q \mid \left(\boldsymbol{\nu}\,\tilde{b}(P' \mid F\langle E\rangle \mid !\,P)\right)\{m := F\} \stackrel{def}{=} H$$

and

$$Q \mid !(P\,\{m := F\}) \xrightarrow{\tau} \equiv$$
$$Q \mid \left(\boldsymbol{\nu}\,\tilde{b}(P' \mid F\langle E\rangle)\right)\{m := F\} \mid !(P\,\{m := F\}) \stackrel{def}{=} K$$

By the side condition of $\mathcal{R}$ on m, we have m nsp $\left(\boldsymbol{\nu}\,\tilde{b}(P' \mid F\langle E\rangle \mid !\,P)\right)$. Therefore, using Theorem 4.3.3(3),

$$H \simeq_{\mathrm{Ct}} Q \mid \left(\boldsymbol{\nu}\,\tilde{b}(P' \mid F\langle E\rangle)\right)\{m := F\} \mid (!\,P)\,\{m := F\} \stackrel{def}{=} H'$$

which finally yields $H'\ \mathcal{R}\ K$ and closes up the bisimulation. □

Proposition 4.3.2 *Suppose* $m \notin fn(\boldsymbol{F})$, *and* m nsp $\boldsymbol{A}, \boldsymbol{E}$. *Then*

$\boldsymbol{A}\{\boldsymbol{E}/X\}\,\{m := \boldsymbol{F}\} \simeq^{\mathrm{c}}_{\mathrm{Ct}} \boldsymbol{A}\,\{m := \boldsymbol{F}\}\{\boldsymbol{E}\,\{m := \boldsymbol{F}\}/X\}$.

PROOF: We use induction on

$$\mathcal{VD}(\boldsymbol{A}) = max\{\{sd(Y)\ :\ Y \in fv(\boldsymbol{A})\} \cup \{sd(\boldsymbol{A}) - 1\}\}$$

We write $\widetilde{E_F}$ and $\widetilde{X}$ as an abbreviations for the tuples $E, E\,\{m := F\}$ and X_1, X_2, respectively.

Basic case: $\mathcal{VD}(\boldsymbol{A}) = 0$. That is, $\boldsymbol{A}$ is a process or a first-order abstraction and $\boldsymbol{A}$ does not contain free variables. There is nothing to prove.

Inductive case: $\mathcal{VD}(\boldsymbol{A}) = n + 1$. By definition of $\simeq^{\mathrm{c}}_{\mathrm{Ct}}$ on open agents, it is enough to prove the result when $\boldsymbol{F}$ and $\boldsymbol{E}$ are closed; therefore in the following they will be written as F and E (i.e., not in bold). We first prove an auxiliary lemma which, intuitively, allows us to distribute $\{m := F\}$ on those occurrences of E which instantiate unguarded occurrences of X.

> **Lemma AUX** Suppose that $\mathcal{VD}(\boldsymbol{A_1}) \leq n+1$, that $X_1 \in fv(\boldsymbol{A_1})$ and that $X_2 \notin fv(\boldsymbol{A_1})$. Then there exists $\boldsymbol{A_2}$ such that
>
> 1. X_1 is guarded in $\boldsymbol{A_2}$;

2. $\boldsymbol{A_2}\{X_1/X_2\} = \boldsymbol{A_1}$;

3. for each E we have

$$\boldsymbol{A_1}\{E/X_1\}\ \{m := F\} \simeq^{\mathrm{c}}_{\mathrm{Ct}} \boldsymbol{A_2}\{\widetilde{E_F}/\widetilde{X}\}\ \{m := F\}.$$

PROOF: By induction on the structure of $\boldsymbol{A_1}$. By η-conversion, we can assume that if $\boldsymbol{A_1}$ is an abstraction, it is of the form $(X)\boldsymbol{P}$ or $(x)\boldsymbol{P}$. The basis of the induction occurs when $\boldsymbol{A_1} = \boldsymbol{\alpha}.\boldsymbol{P}$. For this, take $\boldsymbol{A_2} \stackrel{def}{=} \boldsymbol{A_1}$. For the inductive cases, as (1) and (2) are always be trivial to verify, we only examine (3).

When $\boldsymbol{A_1}$ is a process of the form $\boldsymbol{P_1}|\boldsymbol{P_2}$, $\sum_{i\in I}\boldsymbol{P_i}$, $\boldsymbol{\nu}\, a\, \boldsymbol{P}$, $[a = b]\boldsymbol{P}$ or $!\,\boldsymbol{P}$, use the inductive hypothesis and Theorem 4.3.3. As example, we consider the case of parallel composition. Suppose $\boldsymbol{A_1} = \boldsymbol{P_1} \mid \boldsymbol{P_2}$. By induction there exists $\boldsymbol{P_i'}$, $i = 1, 2$ such that $\boldsymbol{P_i'}$ and $\boldsymbol{P_i}$ satisfy the assertion of the lemma. Define $\boldsymbol{A_2} \stackrel{def}{=} \boldsymbol{P_1'} \mid \boldsymbol{P_2'}$. Then:

$$(\boldsymbol{P_1} \mid \boldsymbol{P_2})\{E/X_1\}\ \{m := F\} \quad =$$

(distributing the substitution)

$$\big(\boldsymbol{P_1}\{E/X_1\} \mid \boldsymbol{P_2}\{E/X_1\}\big)\ \{m := F\} \quad \simeq^{\mathrm{c}}_{\mathrm{Ct}}$$

(Theorem 4.3.3(3))

$$\boldsymbol{P_1}\{E/X_1\}\ \{m := F\} \mid \boldsymbol{P_2}\{E/X_1\}\ \{m := F\} \quad \simeq^{\mathrm{c}}_{\mathrm{Ct}} \quad \text{(induction)}$$

$$\boldsymbol{P_1'}\{\widetilde{E_F}/\widetilde{X}\}\ \{m := F\} \mid \boldsymbol{P_2'}\{\widetilde{E_F}/\widetilde{X}\}\ \{m := F\} \quad \simeq^{\mathrm{c}}_{\mathrm{Ct}}$$

(reversing the steps)

$$(\boldsymbol{P_1'} \mid \boldsymbol{P_2'})\{\widetilde{E_F}/\widetilde{X}\}\ \{m := F\}.$$

The case $\boldsymbol{A_1} = (a)\boldsymbol{P_1}$ is straightforward. The remaining cases are more delicate.

Suppose $\boldsymbol{A_1} = (Y)\boldsymbol{P_1}$. We have $\mathcal{VD}(\boldsymbol{P_1}) \leq n + 1$ (this can be proved reasoning as in the corresponding case for the Lemma Aux of Proposition 4.2.6). Hence, if $\boldsymbol{P_2}$ is the process obtained from $\boldsymbol{P_1}$ using the inductive hypothesis and $\boldsymbol{A_2} \stackrel{def}{=} (Y)\boldsymbol{P_2}$, the assertion (3) of the lemma is immediate.

Suppose now $\boldsymbol{A_1}$: () and defined by an expression which has a variable Y in head position. We have to distinguish the cases $sd(Y) = 1$ and $sd(Y) > 1$. If $sd(Y) = 1$, then $\boldsymbol{A_1} = Y\langle a\rangle$. For this, take $\boldsymbol{A_2} \stackrel{def}{=} X_2\langle a\rangle$ if $X_1 = Y$, and $\boldsymbol{A_2} \stackrel{def}{=} \boldsymbol{A_1}$ if $Y \neq X_1$.

If $sd(Y) > 1$, then $\boldsymbol{A_1} = Y\langle \boldsymbol{E_1}\rangle$, for some $\boldsymbol{E_1}$. We shall only show the proof for $Y = X_1$; the case $Y \neq X_1$ is similar. It is easy to check that $\mathcal{VD}(\boldsymbol{E_1}) \leq n + 1$. Hence, let $\boldsymbol{E_2}$ be the abstraction which the inductive hypothesis associates to $\boldsymbol{E_1}$. By the assertion (3) of the lemma, for

$$\boldsymbol{H_1} \stackrel{def}{=} \boldsymbol{E_1}\{E/X_1\}, \qquad \boldsymbol{H_2} \stackrel{def}{=} \boldsymbol{E_2}\{\widetilde{E_F}/\widetilde{X}\}$$

we have

$$\boldsymbol{H_1}\,\{m := F\} \simeq^{\mathrm{c}}_{\mathrm{Ct}} \boldsymbol{H_2}\,\{m := F\} \tag{C.1}$$

Define $\boldsymbol{A_2} \stackrel{def}{=} X_2\langle \boldsymbol{E_2}\rangle$. If we could prove that

$$\begin{aligned} \boldsymbol{Q_1} \stackrel{def}{=}\ & E\langle \boldsymbol{H_1}\rangle\,\{m := F\} \simeq^{\mathrm{c}}_{\mathrm{Ct}} \\ & E\,\{m := F\}\big\langle \boldsymbol{H_1}\,\{m := F\}\big\rangle \stackrel{def}{=} \boldsymbol{Q_2} \end{aligned} \tag{C.2}$$

and

$$\begin{aligned} \boldsymbol{Q_3} \stackrel{def}{=}\ & \big(E\,\{m := F\}\langle \boldsymbol{H_2}\rangle\big)\,\{m := F\} \simeq^{\mathrm{c}}_{\mathrm{Ct}} \\ & E\,\{m := F\}\big\langle \boldsymbol{H_2}\,\{m := F\}\big\rangle \stackrel{def}{=} \boldsymbol{Q_4} \end{aligned} \tag{C.3}$$

then we would have:

$$\begin{aligned} \boldsymbol{A_1}\,\{m := F\} = \boldsymbol{Q_1} \quad & \simeq^{\mathrm{c}}_{\mathrm{Ct}} \quad \text{(by (C.2))} \\ \boldsymbol{Q_2} \quad & \simeq^{\mathrm{c}}_{\mathrm{Ct}} \\ & \text{(using (C.1) and congruence of } \simeq^{\mathrm{c}}_{\mathrm{Ct}}) \\ E\,\{m := F\}\big\langle \boldsymbol{H_2}\,\{m := F\}\big\rangle \quad & = \\ \boldsymbol{Q_4} \quad & \simeq^{\mathrm{c}}_{\mathrm{Ct}} \quad \text{(by (C.3))} \\ \boldsymbol{Q_3} \quad & = \\ (X_2\langle \boldsymbol{E_2}\rangle)\{\widetilde{E_F}/\widetilde{X}\}\,\{m := F\} \quad & = \\ \boldsymbol{A_2}\{\widetilde{E_F}/\widetilde{X}\}\,\{m := F\} & \end{aligned}$$

which would prove the assertion (3) of the lemma. It remains (C.2) and (C.3). We consider only (C.2), as (C.3), can be worked out in a similar way.

We use the inductive hypothesis of Proposition 4.3.2. By η-conversion, we can assume that the higher-order abstraction E is be of the form $E = (X')P$, for some P; thus

$$\begin{aligned} \boldsymbol{Q_1} &= P\{\boldsymbol{H}_1/X'\}\,\{m := F\} \quad \text{and} \\ \boldsymbol{Q_2} &= P\,\{m := F\}\{\boldsymbol{H}_1\,\{m := F\}/X'\} \end{aligned}$$

Moreover, we have $\mathcal{VD}(P) \leq n$, since X' is the only variable which might occur free in P and $sd(X') \leq n$ follows from $sd(E) = sd(X_1) \leq n+1$. Therefore we conclude that $\boldsymbol{Q_1} \simeq^{\mathrm{c}}_{\mathrm{Ct}} \boldsymbol{Q_2}$ from the inductive hypothesis of Proposition 4.3.2 □

Now we can prove the induction case of Proposition 4.3.2. By definition of $\simeq^{\mathrm{c}}_{\mathrm{Ct}}$, and of $\simeq_{\mathrm{Ct}}$ on on open agents, it is enough to show the result when $fv(\boldsymbol{A}) = \{X\}$ and $\boldsymbol{A} : ()$, and in terms of $\simeq_{\mathrm{Ct}}$. To match the notation of Lemma Aux, let us set $X = X_1$ and $\boldsymbol{A} = P_1$. Thus, we have to show that:

$$P_1\{E/X_1\}\,\{m := F\} \simeq_{\mathrm{Ct}} P_1\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\} \qquad \text{(C.4)}$$

where $sd(X_1) \leq n+1$. For this, we shall need Lemma Aux together with the following equivalence result: The relation

$$\begin{aligned} \mathcal{R} = \{&\Big(P_1\{E/X_1\}\,\{m := F\}, P_1\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\}\Big) : \\ &X_1 \textit{ guarded in } P_1 \textit{ and } sd(X_1) \leq n+1\} \end{aligned}$$

is a bisimulation up-to $\simeq_{\mathrm{Ct}}$. Let $Q_1\,\mathcal{R}\,Q_2$, for

$$Q_1 \stackrel{def}{=} P_1\{E/X_1\}\,\{m := F\} \qquad Q_2 \stackrel{def}{=} P_1\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\}$$

Since X_1 is guarded in P_1 and $\{m := F\}$, due to the restriction on m, cannot alone give rise to a communication, it is enough to look at P_1 to derive the possible moves

for Q_1 and Q_2. With some abuse of language, as P_1 might contain free occurrences of X_1, we will write $P_1 \xrightarrow{\mu} P_1'$ to mean that whenever X_1 in P_1 is instantiated with an abstraction E', then $P_1\{E'/X_1\} \xrightarrow{\mu\{E'/X_1\}} P_1'\{E'/X_1\}$.

There are two cases to consider:

Case (a) $P_1 \xrightarrow{\mu} P_1'$ and μ is an input or a silent move.

Notice that if μ is an input, then the subject of m cannot be be m. Therefore we have:

$$Q_1 \xrightarrow{\mu} P_1'\{E/X_1\}\,\{m := F\},\quad Q_2 \xrightarrow{\mu} P_1'\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\}.$$

Being $\mathcal{R}$ a bisimulation up-to $\simeq_{\mathrm{Ct}}$, it is enough to show that

$$P_1'\{E/X_1\}\,\{m := F\} \simeq_{\mathrm{Ct}}\ \mathcal{R}\ \simeq_{\mathrm{Ct}} P_1'\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\} \quad \text{(C.5)}$$

As $\mathcal{VD}(P_1') \leq n+1$, we can apply Lemma Aux to P_1'; let P_2' be the abstraction returned. We have that

$$\begin{aligned} P_1'\{E/X_1\}\,\{m := F\} \quad &\simeq_{\mathrm{Ct}} \text{ (by Lemma Aux(3))} \\ P_2'\{\widetilde{E_F}/\widetilde{X}\}\,\{m := F\} \quad &= \end{aligned}$$

$$\begin{aligned} \Big(P_2'\{E\,\{m := F\}/X_2\}\Big)\{E/X_1\}\,\{m := F\} \quad &\mathcal{R} \\ &\text{(by Lemma Aux(1), } X_1 \text{ is guarded in } P_2') \\ P_2'\{E\,\{m := F\}/X_2\}\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\} \quad &= \\ P_2'\,\{m := F\}\Big\{E\,\{m := F\}, E\,\{m := F\}/X_1, X_2\Big\} \quad &= \\ &\text{(by Lemma Aux(2))} \\ P_1'\,\{m := F\}\{E\,\{m := F\}/X_1\} & \end{aligned}$$

i.e. summarising:

$$P_1'\{E/X_1\}\,\{m := F\} \simeq_{\mathrm{Ct}}\ \mathcal{R}\ P_1'\,\{m := F\}\Big\{E\,\{m := F\}/X_1\Big\}.$$

Case (b) $P_1 \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle K\rangle} P_1'$ (where K might contain free occurrences of X_1) .

First suppose $a \neq m$. We shall assume that the name m occurs free in $K\{E/X_1\}$ as the other cases — modulo structural congruence — can be worked out in the same way. We have to show is that for every G with $fn(G) \cap \widetilde{b} = \emptyset$,

$$\begin{array}{l} \big(\boldsymbol{\nu}\,\widetilde{b}\,(G\langle K\rangle \mid P_1')\big)\{E/X_1\}\ \{m := F\} \simeq_{\mathrm{Ct}}\ \mathcal{R}\ \simeq_{\mathrm{Ct}} \\ \big(\boldsymbol{\nu}\,\widetilde{b}\,(G\langle K\rangle \mid P_1')\big)\ \{m := F\}\big\{E\ \{m := F\}/X_1\big\} \end{array} \qquad \text{(C.6)}$$

This can be done exploiting Lemma Aux in the same way as for (C.5).

Suppose now that $a = m$. In this case μ gives rise to a communication with $!\,m \star F$. For this, we have to ensure that

$$\begin{array}{l} \big(\boldsymbol{\nu}\,\widetilde{b}\,(F\langle K\rangle \mid P_1')\big)\{E/X_1\}\ \{m := F\} \simeq_{\mathrm{Ct}}\ \mathcal{R}\ \simeq_{\mathrm{Ct}} \\ \big(\boldsymbol{\nu}\,\widetilde{b}\,(F\langle K\rangle \mid P_1')\big)\ \{m := F\}\big\{E\ \{m := F\}/X_1\big\} \end{array}$$

which is similar to (C.6).

Having proved Lemma Aux and $\mathcal{R} \subseteq \simeq_{\mathrm{Ct}}$, we can now derive (C.4). This can be done proceeding as we did to prove (C.5); the only difference is that now we can use $\simeq_{\mathrm{Ct}}$ instead of $\mathcal{R}$.

Appendix D

Proof of Theorem 4.7.5

The proof of Theorem 4.7.6 is obtained with a few adaptations form the proof of Theorem 3.3.3(2) in Appendix A, where we showed that in π-calculus weak barbed equivalence coincides with weak early bisimulation.

Theorem 4.7.6 The relations $\approx$ and $\cong_{\mathrm{Nr}}$ coincide.

PROOF: The inclusion $\cong_{\mathrm{Nr}} \subseteq \approx$ is easy: $\cong_{\mathrm{Nr}}$ implies weak barbed bisimulation; since $\cong_{\mathrm{Nr}}$ is a congruence over the static operators, then also $\cong_{\mathrm{Nr}} \subseteq \approx$.

For the opposite inclusion, let P and Q be the barbed equivalent processes which we want to show normal bisimilar. The only modification to do w.r.t. the proof of Theorem 3.3.3(2) is in the definition of the agent V, which, we remind, is an abstraction with formal parameters (H, Y). We need to add the appropriate summands of V with which the higher-order communications of P and Q are tested. As pointed out in the proof of Theorem 3.3.3(2), if names of different sorts appear in P and Q, then it is convenient to partition the sets H and Y into subsets, one for each different sort (each subset of Y so obtained should have infinite countable names). For notational simplicity, we assume that the names in P and Q belong to two only sorts: A sort of "first-order names", i.e. names which carry names, and a sort of "higher-order names", i.e. names which carry agents. Moreover, we assume that first-order names carry first-order names. Thus the

partitions are $H = H^1 \cup H^2$, and $Y = Y^1 \cup Y^2$, where H^1, Y^1 have the first-order names, and H^2, Y^2 have the higher-order names (H^1 should not be confused with H_1, which is the projection of the pairs of names in H on the first component). In the definition of V below we use the following conventions:

- y, y' are names from Y^1, and m, m' are names from Y^2;
- H_y^+ is $H \cup \{(y, y')\}$, and Y_y^- is $Y - \{y, y'\}$;
- H_m^+ is $H \cup \{(m, m')\}$, and Y_m^- is $Y - \{m, m'\}$;
- The agents W_1 and W_2 are defined as the agent W in Theorem 3.3.3(2); they only differ in the sorts of their triple of parameters (for the interested reader, the second parameter of W_2 could be omitted).

Thus, we set

$$
\begin{aligned}
V \stackrel{def}{=} (H,Y)\Big(& \\
& \textstyle\sum_{(a,a')\in H^1} \sum_{(b,b')\in H^1\cup(y,y')} \overline{a}\langle b\rangle.\overline{c}.\overline{c}.\left(\tau.W_1\langle a', b', in\rangle + \tau.V\langle H_y^+, Y_y^-\rangle\right) && (a1)\\
+ & \textstyle\sum_{(a,a')\in H^1} a(y).\overline{c}.\overline{c}. & \\
& \qquad \left(\tau.W_1\langle a', y', ou\rangle + \tau.V\langle H_y^+, Y_y^-\rangle + \tau.\textstyle\sum_{(b,b')\in H^1}[y=b]\, b'\right) && (a2)\\
+ & \ \tau.d_0 \ + \ \tau.d_1 && (a3)\\
+ & \textstyle\sum_{(a,a')\in H^2} \overline{a}\langle Tr_m\rangle.\overline{c}.\overline{c}.\left(\tau.W_2\langle a', m', in\rangle + \tau.V\langle H_m^+, Y_m^-\rangle\right) && (a4)\\
+ & \textstyle\sum_{(a,a')\in H^2} a(Y).\Big(!\, m \star Y \mid \overline{c}.\overline{c}.\left(\tau.W_2\langle a', m', ou\rangle + \tau.V\langle H_m^+, Y_m^-\rangle\right)\Big) \ \Big) && (a5)
\end{aligned}
$$

The summands (a1)-(a3) are the same as in the definition of V in Theorem 3.3.3(2), for H_1 and W_1 in place of H and W. (a4) and (a5) are the new summands, to deal with the higher-order communications. The use of these summands in the proof — including the new summands (a4) and (a5) — in entirely analogous to their use in Theorem 3.3.3(2). Note that the tests on P and Q's higher-order communications effected in (a4) and (a5) follow the definition of normal bisimulation: In (a4) P and Q's inputs are tested with a fresh trigger Tr_m; whereas in (a5) the agent received from P or Q's output is tested with the guarded-replication construct.

The bisimulation relation to exhibit is the same as in Theorem 3.3.3(2), namely

$$\begin{aligned}\mathcal{R} = \{(P,Q) \;:\; & n, H, Y, \tilde{x} \text{ exist s.t.} \\ & - fn(P,Q) \cup \{\tilde{x}\} \subseteq H_1 \\ & - n(H) \cap Y = \emptyset \\ & - \boldsymbol{\nu}\, \tilde{x}\, C_n^{H,Y}[P] \;\dot{\approx}\; \boldsymbol{\nu}\, \tilde{x}\, C_n^{H,Y}[Q] \qquad \}\end{aligned}$$

The relation $\mathcal{R}$ is a weak normal bisimulation up-to $\equiv$ (the use of $\equiv$ is useful to deal with a higher-order output of P or Q, say $P \xrightarrow{\boldsymbol{\nu}\, \tilde{b}\, \overline{a}\langle F\rangle} P'$, to allow us to reconstruct the term $\boldsymbol{\nu}\, \tilde{b}(P' \mid !m \star F)$ after the interaction of P with $V\langle H, Y\rangle$). $\square$

Bibliography

[AB84] D. Austry and G. Boudol. Algebre de processus et synchronisations. *Theoretical Computer Science*, 30(1):91–131, 1984.

[Abr87] S. Abramsky. *Domain Theory and the Logic of Observable Properties.* PhD thesis, University of London, 1987.

[Abr89] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

[AdBKR89] P. America, J. de Bakker, J. Kok, and J. Rutten. Denotational semantics of a parallel object-oriented language. *Information and Computation*, 83(2), 1989.

[AG88] E. Astesiano and A. Giovini. Generalized bisimulation in relational specifications. In *STACS '88*, volume 294 of *Lecture Notes in Computer Science*, pages 207–226. Springer Verlag, 1988.

[Agh86] G. Agha. *Actors: a Model of Concurrent Computation in Distributed Systems.* The MIT Press, 1986.

[Agh90] G. Agha. The structure and semantics of actor languages. In de Bakker J.W., W.P. de Rover, and G. Rozenberg, editors, *Foundations of Object Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 1–59. Springer Verlag, 1990.

[Ame89] P. America. Issues in the design of a parallel object-oriented language. *Formal Aspects of Computing*, 1(4):366–411, 1989.

[AO89] S. Abramsky and L. Ong. Full abstraction in the lazy lambda calculus. Technical report, Imperial College, 1989. To appear in *Information and Computation.*

[AR87] E. Astesiano and G. Reggio. SMoLCS-driven concurrent calculi. In *TAPSOFT '87*, volume 249 of *Lecture Notes in Computer Science*, pages 169–201. Springer Verlag, 1987.

[ARW86] E. Astesiano, G. Reggio, and M. Wirsing. Relational specifications and observational semantics. In *MFCS '86*, volume 233 of *Lecture Notes in Computer Science*, pages 209–217. Springer Verlag, 1986.

[AZ84] E. Astesiano and E. Zucca. Parametric channels via label expressions in CCS. *Theoretical Computer Science*, 33:45–64, 1984.

[Bar84] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic.* North Holland, 1984. Revised edition.

[BB89] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS.* North Holland, 1989.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[BCHK91] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. TR 13/91, University of Sussex, 1991. Submitted for publication.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[BK85] J.A. Bergstra and J.W. Klop. Algebra for communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[Bou89] G. Boudol. Towards a lambda calculus for concurrent and communicating systems. In *TAPSOFT '89*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161, 1989.

[Bou90] G. Boudol. A lambda calculus for parallel functions. Rapport de recherche 1231, INRIA-Sophia Antipolis, 1990.

[Bou91] G. Boudol. A lambda calculus for (strict) parallel functions. Rapport de recherche 1387, INRIA-Sophia Antipolis, 1991. To appear in Information and Computation.

[Bou92] G. Boudol. Asynchrony and the π-calculus. Submitted for publication, 1992.

[CF92] R. Cartwright and M. Felleisen. Observable sequentiality and full abstraction. In *Proc. POPL 92*, pages 328–342. Association for Computing Machinery, 1992.

[CH89] I. Castellani and M. Hennessy. Distributed bisimulation. *JACM*, 10(4):887–911, 1989.

[Cur92] P.-L. Curien. Observable algorithms on concrete data strucures. In *7th LICS Conf.*, pages 432–443. IEEE Computer Society Press, 1992.

[DD89] P. Degano and P. Darondeau. Causal trees. In *15th ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 234–248. Springer Verlag, 1989.

[DH84] R. De Nicola and R. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[DO91] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software - Practice and Experience*, 21(8):757–785, 1991.

[DS85] R. De Simone. Higher level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Ehr79] H. Ehrig. Introduction to the algebraic theory of graph grammars. In *International Workshop on Graph Grammars*, volume 73 of *Lecture Notes in Computer Science*, pages 229–273. Springer Verlag, 1979.

[ELR90a] J. Engelfriet, G. Leih, and G. Rozenberg. Net-based description of parallel object-based systems. In J.W. de Bakker, W.P. de Rover, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 229–273. Springer Verlag, 1990.

[ELR90b] J. Engelfriet, G. Leih, and G. Rozenberg. Parallel object-based systems and petri nets (Parts I and II). Tr 90-4 and 90-5, Department of Computer Science, Leiden University, 1990.

[EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label-passing. Report DAIMI PB-208, Computer Science Department, University of Aarhus, Denmark, 1986.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GMP89] A. Giacalone, P. Mishra, and S. Prasad. FACILE, a symmetric integration of concurrent and functional programming. In J. Diaz and F. Orejas, editors, *TAPSOFT '89*, volume 352 of *Lecture Notes in Computer Science*, pages 189–209. Springer Verlag, 1989.

[HB77] C. Hewitt and H. Baker. Laws for communicating parallel processes. In *1977 IFIP Congress Proceedings*, pages 987–992. IFIP, 1977.

[Hew77] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial intelligence*, 8(3):323–364, 1977.

[HL80] J.R. Hindley and G. Longo. Lambda calculus models and extensionality. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 26:289–310, 1980.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hol83] S. Holmstrom. PFL: A functional language for parallel programming. In *Declarative Programming Workshop,* Programming Methodology Group, Chalmers University of Technology, University of Goteborg, Sweden, 1983.

[HS86] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.

[HT] K. Honda and M. Tokoro. On asynchronous communication semantics. In M. Tokoro, O. Nierstrasz, P. Wegner, and A. Yonezawa., editors, *ECOOP '91 Workshop on Object Based Concurrent Programming,* Geneva, Switzerland, 1991. , volume 612 of *Lecture Notes in Computer Science.* Springer Verlag.

[JR89] D. Janssens and G. Rozenberg. Actor grammars. *Mathematical System Theory*, 22:75–107, 1989.

[JR90] D. Janssens and G. Rozenberg. Graph grammar-based description of object-based systems. In J.W. de Bakker, W.P. de Rover, and G. Rozenberg, editors, *Foundations of Object Oriented Languages*,

volume 489 of *Lecture Notes in Computer Science*, pages 341–404. Springer Verlag, 1990.

[Kre80] H.J. Kreowsky. A comparison between petri nets and graph grammars. In *Proceedings International Workshop WG '80*, volume 100 of *Lecture Notes in Computer Science*, pages 306–317. Springer Verlag, 1980.

[KS85] J.R. Kennaway and M.R. Sleep. Syntax and informal semantics of DyNe, a parallel language. In *Proceedings of the Workshop on the Analysis of Concurrent Systems 1983*, volume 207 of *Lecture Notes in Computer Science*, pages 222–230. Springer Verlag, 1985.

[Let92] L. Leth. *Functional Programs as Reconfigurable Networks of Communicating Processes.* PhD thesis, Imperial College, London University, 1992.

[Mil77] R. Milner. Fully abstract models of typed lambda calculus. *Theoretical Computer Science*, 4:1–22, 1977.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer Verlag, 1980.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:269–310, 1983.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mil90a] R. Milner. Functions as processes. Research Report 1154, INRIA, Sofia Antipolis, 1990. Final version in *Journal of Mathem. Structures in Computer Science* 2(2):119–141, 1992.

[Mil90b] R. Milner. Sorts and types in the π-calculus. Handwritten notes, Edinburgh, December 1990.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. To appear in the *Proceedings of the International Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991.

[MPW91] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. In *2nd CONCUR*, volume 527 of *Lecture Notes in Computer Science*, pages 45–60. Springer Verlag, 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Verlag, 1992.

[Nie] O. Nierstrasz. Towards an object calculus. In M. Tokoro, O. Nierstrasz, P. Wegner, and A. Yonezawa, editors, *ECOOP '91 Workshop on Object Based Concurrent Programming,* Geneva, Switzerland, 1991. , volume 612 of *Lecture Notes in Computer Science.* Springer Verlag.

[Nie89] F. Nielson. The typed λ-calculus with first-class processes. In *Proc. PARLE '89*, volume 366 of *Lecture Notes in Computer Science.* Springer Verlag, 1989.

[Ong88] L. Ong. *The Lazy Lambda Calculus: an Investigation into the Foundations of Functional Programming.* PhD thesis, University of London, 1988. Also Prize Fellowship Dissertation, Trinity College, Cambridge, 256 pp.

[Par81] D.M. Park. Concurrency on automata and infinite sequences. In

P. Deussen, editor, *Conf. on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*. Springer Verlag, 1981.

[Phi87] I.C.C. Phillips. Refusal testings. *Theoretical Computer Science*, 50:241–284, 1987.

[Plo75] G.D Plotkin. Call by name, call by value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Plo77] G.D Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo81] G.D Plotkin. A structural approach to operational semantics. DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[PMS90] G.M. Pinna and A. Maggiolo-Schettini. Transformations of Pr/T nets based on translation into structure grammars. In *Proceedings of the 11th International Conference on Application and Theory of Petri Nets*, pages 332–351, 1990. Paris, France.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, editor, *12th ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer Verlag, 1985.

[Rei85] W. Reisig. *Petri Nets: an Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.

[Sana] D. Sangiorgi. Bisimulations up to context (provisory title). In preparation. Edinburgh.

[Sanb] D. Sangiorgi. True-concurrency versus interleaving semantics in calculi for mobile processes: the case of location bisimulation (provisory title). In preparation. Edinburgh.

[San91] D. Sangiorgi. Most general sorts and types in HOπ. Handwritten notes, August, 1991.

[San92] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. In *7th LICS Conf.* IEEE Computer Society Press, 1992.

[SM92] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". To appear in the Proceedings of CONCUR '92 as Lecture Notes in Computer Science, 1992.

[SY85] R.E. Strom and S. Yemini. The NIL distributed systems programming language: a status report. In *Seminar on Concurrency 1984*, volume 197 of *Lecture Notes in Computer Science*, pages 512–523. Springer Verlag, 1985.

[Tho89] B. Thomsen. Plain CHOCS. Technical Report DOC 89/4, Department of Computing, Imperial College, 1989.

[Tho90] B. Thomsen. *Calculi for Higher Order Communicating Systems.* PhD thesis, Department of Computing, Imperial College, 1990.

[Tur] N.D. Turner. Forthcoming PhD thesis, Department of Computer Science, University of Edinburgh.

[Tur91] N.D. Turner. An intermediate semantics for the π-calculus. Unpublished notes, Edinburgh, June, 1991.

[Wal90] D. Walker. π-calculus semantics of object-oriented programming languages. ECS-LFCS-90-122, LFCS, Dept. of Comp. Sci., Edinburgh Univ., 1990. Also in *Proc. Conference on Theoretical Aspects of Computer Software*, Tohoku University, Japan, Sept. 1991.

[Wal91] D. Walker. Some results on the π-calculus. In A. Yonezawa and T. Ito, editors, *Concurrency: Theory, Language, and Architecture*,

volume 491 of *Lecture Notes in Computer Science*, pages 21–35. Springer Verlag, 1991.