Higher-Order Concurrency: Expressiveness and Decidability Results

Jorge A. Pérez P.

Technical Report UBLCS-2010-07

March 2010

Department of Computer Science

University of Bologna

Mura Anteo Zamboni 7
40127 Bologna (Italy)

Recent Titles from the UBLCS Technical Report Series

2008-18 *Lebesgue's Dominated Convergence Theorem in Bishop's Style*, Sacerdoti Coen, C., Zoli, E., November 2008.

2009-01 *A Note on Basic Implication*, Guidi, F., January 2009.

2009-02 *Algorithms for network design and routing problems (Ph.D. Thesis)*, Bartolini, E., February 2009.

2009-03 *Design and Performance Evaluation of Network on-Chip Communication Protocols and Architectures (Ph.D. Thesis)*, Concer, N., February 2009.

2009-04 *Kernel Methods for Tree Structured Data (Ph.D. Thesis)*, Da San Martino, G., February 2009.

2009-05 *Expressiveness of Concurrent Languages (Ph.D. Thesis)*, di Giusto, C., February 2009.

2009-06 *EXAM-S: an Analysis tool for Multi-Domain Policy Sets (Ph.D. Thesis)*, Ferrini, R., February 2009.

2009-07 *Self-Organizing Mechanisms for Task Allocation in a Knowledge-Based Economy (Ph.D. Thesis)*, Marcozzi, A., February 2009.

2009-08 *3-Dimensional Protein Reconstruction from Contact Maps: Complexity and Experimental Results (Ph.D. Thesis)*, Medri, F., February 2009.

2009-09 *A core calculus for the analysis and implementation of biologically inspired languages (Ph.D. Thesis)*, Versari, C., February 2009.

2009-10 *Probabilistic Data Integration*, Magnani, M., Montesi, D., March 2009.

2009-11 *Equilibrium Selection via Strategy Restriction in Multi-Stage Congestion Games for Real-time Streaming*, Rossi, G., Ferretti, S., D'Angelo, G., April 2009.

2009-12 *Natural deduction environment for Matita*, C. Sacerdoti Coen, E. Tassi, June 2009.

2009-13 *Hints in Unification*, Asperti, A., Ricciotti, W., Sacerdoti Coen, C., Tassi, E., June 2009.

2009-14 *A New Type for Tactics*, Asperti, A., Ricciotti, W., Sacerdoti Coen, C., Tassi, E., June 2009.

2009-15 *The k-Lattice: Decidability Boundaries for Qualitative Analysis in Biological Languages*, Delzanno, G., Di Giusto, C., Gabbrielli, M., Laneve, C., Zavattaro, G., June 2009.

2009-16 *Landau's "Grundlagen der Analysis" from Automath to lambda-delta*, Guidi, F., September 2009.

2010-01 *Fast overlapping of protein contact maps by alignment of eigenvectors*, Di Lena, P., Fariselli, P., Margara, L., Vassura, M., Casadio, R., January 2010.

2010-02 *Optimized Training of Support Vector Machines on the Cell Processor*, Marzolla, M., February 2010.

2010-03 *Modeling Self-Organizing, Faulty Peer-to-Peer Systems as Complex Networks* Ferretti, S., February 2010.

2010-04 *The qnetworks Toolbox: A Software Package for Queueing Networks Analysis*, Marzolla, M., February 2010.

2010-05 *QoS Analysis for Web Service Applications: a Survey of Performance-oriented Approaches from an Architectural Viewpoint*, Marzolla, M., Mirandola, R., February 2010.

2010-06 *The dark side of the board: advances in Kriegspiel Chess (Ph.D. Thesis)*, Favini, G.P., March 2010.

Dottorato di Ricerca in Informatica
Università di Bologna e Padova
INF/01 INFORMATICA
Ciclo XXII

Higher-Order Concurrency: Expressiveness and Decidability Results

Jorge A. Pérez P.

March 2010

Coordinatore:	Tutore:
Prof. Simone Martini	Prof. Davide Sangiorgi

Higher-Order Concurrency: Expressiveness and Decidability Results

Higher-order process calculi are formalisms for concurrency in which processes can be passed around in communications. Higher-order (or process-passing) concurrency is often presented as an alternative paradigm to the first order (or name-passing) concurrency of the π-calculus for the description of mobile systems. These calculi are inspired by, and formally close to, the λ-calculus, whose basic computational step —β-reduction— involves term instantiation.

The theory of higher-order process calculi is more complex than that of first-order process calculi. This shows up in, for instance, the definition of behavioral equivalences. A long-standing approach to overcome this burden is to define *encodings* of higher-order processes into a first-order setting, so as to transfer the theory of the first-order paradigm to the higher-order one. While satisfactory in the case of calculi with basic (higher-order) primitives, this indirect approach falls short in the case of higher-order process calculi featuring constructs for phenomena such as, e.g., localities and dynamic system reconfiguration, which are frequent in modern distributed systems. Indeed, for higher-order process calculi involving little more than traditional process communication, encodings into some first-order language are difficult to handle or do not exist. We then observe that foundational studies for higher-order process calculi must be carried out *directly* on them and exploit their peculiarities.

This dissertation contributes to such foundational studies for higher-order process calculi. We concentrate on two closely interwoven issues in process calculi: *expressiveness* and *decidability*. Surprisingly, these issues have been little explored in the higher-order setting. Our research is centered around a *core* calculus for higher-order concurrency in which only the operators strictly necessary to obtain higher-order communication are retained. We develop the basic theory of this core calculus and rely on it to study the expressive power of issues universally accepted as *basic* in process calculi, namely synchrony, forwarding, and polyadic communication.

Keywords: concurrency theory, process calculi, higher-order communication, expressiveness, decidability.

Acknowledgments

My greatest debt is to Davide Sangiorgi. Having him as supervisor has been truly inspiring. His careful supervision has influenced enormously my way of doing (and approaching) research. His continuous support and patience during these three years were fundamental to me. I am still amazed by the fact that Davide had always time for me, not only for scientific discussions but also for sorting out everyday issues. I am most grateful to him for his honest and direct advice, and for the liberty that he gave me during my studies.

I also owe much to Camilo Rueda and Frank D. Valencia. I do not forget that it was Camilo who introduced me to research, thus giving me an opportunity that most people in his position would have refused. Even if my PhD studies were not directly related to his research interests, Camilo was always there, interested in my progresses, encouraging me with his support and friendship. Frank not only introduced me to concurrency theory; he also gave me constant advise and support during my PhD studies and long before. Frank had a lot to do with me coming to Bologna, and that I will never forget.

There is no way in which I could have completed this dissertation by myself. It has been a pleasure to collaborate with extremely talented people, to whom I am deeply grateful: Cinzia Di Giusto, Ivan Lanese, Alan Schmitt, Gianluigi Zavattaro. Thank you for your kindness, generosity and, above all, for your patience.

Many thanks to Uwe Nestmann and Nobuko Yoshida for having accepted to review this dissertation. Thanks also to the members of my internal committee (*commissione*), Cosimo Laneve and Claudio Sacerdoti-Coen. I am indebted to Simone Martini, the coordinator of the PhD program, for all his constant availability and kindness.

Many people proof-read parts of this dissertation, and provided me with constructive criticisms. I am grateful to all of them for their time and availability: Jesús Aranda, Alberto Delgado, Cinzia Di Giusto, Daniele Gorla, Julián Gutiérrez, Hugo A. López, Claudio Mezzina, Margarida Piriquito, Frank D. Valencia. A special thanks goes to Daniele Varacca, who suffered an early draft of the whole document and provided me with insightful remarks. Along these years I have benefited a lot from discussions with/comments from a lot of people. I am most grateful for their positive attitude towards my work: Jesús Aranda, Ahmed Bouajjani, Gérard Boudol, Santiago Cortés, Rocco De Nicola, Daniele Gorla, Matthew Hennessy, Thomas Hildebrandt, Kohei Honda, Roland Meyer, Fabrizio Montesi, Camilo Rueda, Jean-Bernard Stefani, Frank D. Valencia, Daniele Varacca, Nobuko Yoshida.

During 2009 I spent some months visiting Alan Schmitt in the SARDES team at INRIA Grenoble - Rhône-Alpes. The period in Grenoble was very enriching and productive; a substantial part of this dissertation was written there. I am grateful to Alan and to Jean-Bernard Stefani for the opportunity of working with them and for treating me as another member of the team. I would like to thank Diane Courtiol for her patient help with all the administrative issues during my stay, and to Claudio Mezzina (or the "tiny little Italian with a pony tail", as he requested to be acknowledged) for being such a friendly office mate. I also thank the INRIA Equipe Associée BACON for partially supporting my visit.

I would like to express my appreciation to the University of Bologna - MIUR for supporting my studies through a full scholarship. Thanks also to the administrative staff in the Department of Computer Science, for their help and kindness in everyday issues.

I am most proud to be part of a small group of Colombians doing research abroad. We all share many things: we started in the same research group, have similar backgrounds, and came to Europe more or less at the same time. With most of them I even shared an office for a long time. Many thanks to: Jesús Aranda, for his inherent kindness; Alejandro Arbelaez, for the good times while working in Colombia and his hospitality during trips to Paris; Andrés Aristizábal, for the constant support in spite of our favorite football teams; Alberto Delgado, with whom I started doing research back in 2002 and has always been there ever since; Gustavo Gutiérrez, for the old, good times when he was my first boss, and for the sincere support during all these years; Julian Gutiérrez, for all the discussions on life and research, during our PhDs and even way before; Hugo A. López, for sharing with me the experience of living in Italy, several trips, and a plenty of discussions on concurrency theory and life at large; Carlos Olarte, for all the good times in Paris and hospitality in the great city of Bourg-la-Reine; Luis O. Quesada, for his exceptional kindness and hospitality during a visit to Ireland (despite of the fact that my visit brought historical floodings to the Cork region). Above all, I would like to thank all of them for being my friends.

Perhaps the most significant achievement of my PhD studies is all the people I have meet along the way. A special thanks goes to: Cinzia Di Giusto, for her constant support and friendship, and for being the most enthusiastic partner in research one could imagine; Antonio Vitale, for the several trips and for sharing with me bits of PhD frustration and pizzas of varying quality; Ivan Lanese, the loyal friend, the reasonable flat mate, and the talented co-author. Thanks also to: Stefano Arteconi, for insightful and enjoyable discussions on Italy, movies, and music; Ferdinanda Camporesi, for the many chats and the movies we watched together; Marco Di Felice, for being the most welcoming and friendly office mate in underground and being worse than me in *calcetto*; Ebbe Elsborg (and family) —the most loyal reader of my blog— for his kind hospitality during the most splendid vacation in Denmark I could have imagined, and for plenty of discussions on pretty much every aspect of life; Elena Giachino and Luis Pérez, for all the fun we had together at summer schools, and for dinners and parties at Pisa; Zeynep Kiziltan, for the chats over lunch that *didn't deal* about work; Flavio S. Mendes, for the many trips we did together around Italy, the constant support and friendship, and the many times I stayed at his place; Margarida Piriquito, for the most unexpected friendship I can remember; Sylvain Pradalier, or the coolest French guy I could have shared an office with; Alan Schmitt (and family), for the several great dinners at his place (in Grenoble, but also in Casalecchio) and the clever games in which I would always lose no matter how hard I would try.

Last but not least, I would like to thank my family for their unconditional, constant support. There are no words to thank my parents, my sisters, my brother, and my grandmother. Their love gave me strength to overcome the difficult times. I would also like to thank Andrés F. Monsalve, who is more like a brother than a friend to me. Thanks also to the rest of my family, the many cousins, uncles, and aunts for their continued support towards me.

Contents

List of Figures

Chapter 1

Introduction

This dissertation studies calculi for *higher-order concurrency*, and focuses on their *expressive power* and *decidable properties*. Our thesis is that a *direct* and *minimal* approach to the expressiveness and decidability of higher-order concurrency is both *necessary* and *relevant*, given the emergence of higher-order process calculi with *specialized constructs* and the inconvenience (or non-existence) of first-order representations for such constructs.

1.1 Context and Motivation

The challenging nature of *concurrent systems* is no longer a novelty for computer science. In fact, by now there is a consolidated understanding on how concurrent behavior departs from sequential computation. Based on pioneering developments by Hewitt, Milner, Hoare, and others, the last three decades have witnessed a remarkable progress on the formulation of foundational theories of concurrent processes; notions such as *interaction* and *communication* are widely accepted to be intimately related to *computing* at large. Given the wealth of abstract languages, theories, and application areas that have emerged from this progress, it is fair to say that *concurrency theory* is no longer in its infancy.

This development of concurrency theory coincides with the transition towards *global ubiquitous computing* we witness nowadays. Supported by a number of technological advances —most notably, the availability of cheaper and more powerful processors, the increase in flexibility and power of communication networks, and the widespread consolidation of the Internet— global ubiquitous computing (GUC, in the sequel) is a broad term that refers to computing over massively networked, dynamically reconfigurable infrastructures that interconnect heterogeneous collections of computing devices. As such, systems in GUC represent the natural evolution of traditional distributed systems, and distinguish from these in aspects such as *mobility*, *network-awareness*, and *openness* on which we comment next.

Nowadays we find *mobility* in *devices* that move in our physical world while performing diverse kinds of computation (mobile phones, laptops, PDAs), as well as in *objects* travelling across communication networks (SMSs, structured data as XML files, snippets of runnable code, software agents). Sustained advances in bandwidth growth and network connectivity have broaden the range of feasible communications; as a result, communication objects not only exhibit now an increasingly complex structure but also an autonomous nature. This evolution in the nature of communication objects can be seen in a number of applications these days:

Distribution of digital content. It is becoming increasingly popular to buy the right to download digital content (music, video, books) from online stores directly to personal computers or mobile devices. Here the communication objects are the (pieces of) multimedia files that are transmitted from the online store to the customer; these are files in standardized media formats and hence self-contained to a large extent.

***Plug-ins* (or *add-ons*).** Plug-ins are self-contained programs that integrate within applications (e.g. web browsers, email clients) with the purpose of inserting, removing, or updating functionalities at runtime. For instance, plug-ins in web browsers have made possible a transition from data mobility to code mobility: rather than submitting data to a web service and getting results, the model is to *download* the required behavior (e.g. a snippet of JavaScript code) and *apply* it to data which may be local or remote. Similarly, most tools for software update are in fact small helper applications available online, ready to be downloaded; once installed, they obtain information on the current configuration of the system and use it to retrieve the most appropriate update from some application server.

Service-oriented Computing. *Services* are software artifacts which can be accessed, manipulated, structured into complex architectures, and distributed in wide area networks such as the Internet. Services are the building blocks in service-oriented computing, an approach to distributed applications that has received much attention in recent years. Forms of *service mobility* are most natural to service-oriented architectures that define workflows involving services which cannot be determined statically before execution. As such, these services must be found and integrated at run time. The behavior of such architectures thus depends on correct, reliable forms of service/code mobility.

In general, mobility cannot abstract from the *locations* of the moving entities (computing devices, communication objects). For instance, in the service-oriented computing scenario just sketched, it is crucial to be able to tell *where* a requested service is (e.g. in the service provider, in the requester, in transit) as such information entails a different behavior for the system. A location can be as concrete as the wireless network a PDA connects to, or as abstract as the administrative domains in which wide area networks are usually partitioned. A commonality here is the reciprocal relationship between locations and mobility, as (the behavior of) a mobile entity and its surrounding environment (determined by its location) might have direct influence on each other. This can be seen, for instance, in the relationship between network bandwidth and the quality of service available to mobile devices; in the websites that change depending on the country in which they are accessed; in the actions of network reconfiguration triggered by high peaks of user activity. This phenomenon is sometimes referred to as *network-awareness*: it can be seen to embody a notion of *structure* that not only underlies mobile behavior but that often determines it.

The *openness* of modern computing environments results from the understanding that systems in GUC are built as very large collections of loosely coupled, heterogeneous components. These components might not be known a priori; unknown or partially specified components could enter and leave the system at will. In general, an open system should allow to add, suspend, update, relocate, and remove entire components transparently. From a global point of view, open systems are seldom meant to terminate; as such, their overall behavior must abstract from changes on the local state of its components, and in particular from their malfunction. Hence, forms of *dynamic system reconfiguration*, with varying levels of autonomy, are most natural within models of open systems. It is worth pointing that openness is closely related to mobility and network-awareness in that not only complete components might move across the predefined structure of the system, but also it might occur that such a structure is reconfigured as a result of the interactions of mobile components. This is the case of, for instance, a running component which disconnects from one location and later on reconnects to some other location.

Systems in GUC therefore represent a challenge for computer science in general, and for concurrency theory in particular. As we have seen, such environments feature complex forms of concurrent behavior that go way beyond the (already complex) interaction patterns present in traditional distributed systems. The challenge therefore consists in the formulation of foundational theories to cope with the features of modern computing environments.

We believe that in this context *higher-order concurrency* has much to offer. In fact, process-passing communication as available in higher-order process calculi is closely related to the aspects of mobility, network-awareness, and openness discussed for GUC. The communication of objects with complex structure can be neatly represented in higher-order process calculi by the communication of *terms* of the language. As in the first-order case, extensions of higher-order process calculi with constructs for network-awareness are natural; process communication adds the possibility of describing richer and more realistic interaction patterns between different computation loci. Furthermore, higher-order communication allows to consider autonomous, self-contained software artifacts —such as components, services, or agents— as *first-class objects* which can be moved, executed, manipulated. This allows for clean and modular descriptions of open systems and their behavior.

At this point it might be clear that higher-order communication arises in abstract languages for GUC in the form of *specialized constructs* that go beyond mere process communication. Instances of such constructs include forms of *localities* that lead to involved process hierarchies featuring complex communication patterns; operators for *reflection* that allow to observe and/or modify process execution at runtime; sophisticated forms of *pattern matching* or *cryptographic operations* used over terms representing messages or semi-structured data.

The wide range and inherent complexity of the higher-order interactions that underlie these specialized operators cast serious doubts on the convenience of studying the theory of higher-order concurrent languages featuring such operators by means of first-order representations. Based on this insight, in this dissertation we shall argue that foundational studies for higher-order process calculi must be undertaken *directly* on them and exploit their peculiarities. This is particularly critical for those issues that have remained unexplored in the theory of higher-order concurrency. We shall concentrate on two of such issues, namely *expressiveness* and *decidability*, two closely interwoven concerns in process calculi at large.

1.2 First-Order and Higher-Order Concurrency

In this section we first comment on the relationship between first-order and higher-order concurrency. Then, we give intuitions on Sangiorgi's representability result of higher-order into first-order concurrency, and argue that it does not carry over to higher-order languages with specialized constructs. As compelling example, we illustrate the case of a higher-order process calculus with a very basic form of localities.

Two Kinds of Mobility. Broadly speaking, mobility has arisen in calculi for concurrency in essentially two kinds: *link* and *process* mobility. In the first kind it is *links* that move in an abstract space of linked processes, whilst in the second kind it is *processes* that move (Sangiorgi and Walker, 2001). By far, link mobility has attracted most of the attention of the research community in process calculi. In the π-calculus (Milner et al., 1992; Sangiorgi and Walker, 2001) —arguably the most influential process calculus— link mobility is achieved by means of *name-passing*. While the impact of the π-calculus can be appreciated in the numerous efforts devoted to study its theory, variants, and applications, its significance is strongly related to the unifying view it provides to explain otherwise unrelated models and paradigms such as, e.g., the λ-calculus (Milner, 1992; Sangiorgi, 1992), concurrent object-oriented programming (Walker, 1995), and structured communication (Honda et al., 1998). It is therefore no surprise that *first-order concurrency* based on the communication of links is the predominant paradigm in process calculi for mobility.

In comparison, process calculi for *higher-order concurrency* have attained much less attention. Higher-order process calculi emerged first as concurrent extensions of functional languages (see, e.g., (Boudol, 1989; Nielson, 1989)). As a matter of fact, higher-order process calculi are inspired by, and formally close to, the λ-calculus, whose basic computational step — β-reduction — involves term instantiation.[1] Later on, as a way of studying forms of code mobility and mobile agents, a number of process calculi extended with process-passing features were put forward; examples include CHOCS (Thomsen, 1989), Plain CHOCS (Thomsen, 1993), and the Higher-Order π-calculus (Sangiorgi, 1992), which were intensely studied in the early 1990s. Although that period witnessed remarkable progresses on the theory of higher-order process calculi (most notably, on the development of their behavioral theory), a number of fundamental issues were not addressed. Some of such issues still remain unexplored; this is the case of expressiveness and decidability, central to this dissertation.

The contrast in the attention that each paradigm has received is certainly not a coincidence. We believe it can be explained by the introduction of what is probably the most prominent result for higher-order process calculi: in the context of the π-calculus, Sangiorgi (1992) showed that the higher-order paradigm is *representable* into the first-order one by means of a rather elegant translation, in which the communication of a process is modeled as the communication of a pointer that can activate as many copies of such a process as needed. Crucially, such a translation is *fully-abstract* with respect to barbed congruence, the form of contextual equivalence used in concurrency theory. Hence, the behavioral theory from the first-order setting can be readily transferred to the higher-order one. By demonstrating that the higher-order paradigm only adds modeling convenience, this result greatly contributed to consolidate the π-calculus as a basic formalism for concurrency. It also appears to have contributed to a decline of interest in formalisms for higher-order concurrency. In our view, Sangiorgi's representability result was so conclusive at that time that it indirectly put forward the idea that his translation could be adapted to represent *every* kind of higher-order interaction. This misconception seems to persist nowadays, even if, as we shall see, it has been shown that for higher-order process calculi with little more than process communication, translations into some first-order language —as in Sangiorgi's representability result— are unsatisfactory or do not exist.

[1] Probably as a consequence of this, the appellation *higher-order* is often used to refer to the exchange of values that might contain terms of the language, i.e., processes. Also intrinsically related with the appellation higher-order is the *non-linear* character of process mobility in higher-order process calculi: upon reception, received processes can be freely copied, or even discarded. This is one of the points of contrast between higher-order process calculi and calculi for mobility such as Ambients (Cardelli and Gordon, 2000) and its several variants, in which processes can move around but *cannot* be copied or discarded, i.e., they feature *linear* process mobility. For this reason, in what follows we *do not* consider calculi such as Ambients as higher-order process calculi.

Sangiorgi's Representability Result. Let us give an intuitive overview of Sangiorgi's representability result of higher-order π-calculus into the (first-order) π-calculus, as presented in (Sangiorgi, 1992). The discussion here will be informal: our focus will be on rough intuitions rather than on technicalities. Formal details and extended explanations are deferred to Chapter 2.

Sangiorgi's translation of higher-order into first-order π-calculus can be presented as follows. Let us use $P, Q, R, M, N, \ldots$ to range over processes. Assume that $\overline{a}\langle P\rangle.\, Q$ represents the output of process P on name (or channel) a, with continuation Q. The higher-order input action $a(x).\, P$ expects a process value on name a and, upon reception of a process R in the bound variable x, it behaves as the process P in which all free occurrences of x have been substituted with R. Constructs for parallel composition , non-deterministic choice $+$, name restriction $\nu r\, P$, process replication $!P$, and inaction $\mathbf{0}$ are assumed as expected. The *reaction rule*

$$(a(x).\, M + M') \;\; (\overline{a}\langle R\rangle.\, N + N') \longrightarrow M\{{}^{R}\!/\!{}_{x}\} \;\; N\,.$$

defines the behavior of higher-order processes independently of its environment.

As an example, consider the higher-order process

$$P \overset{\text{def}}{=} \overline{a}\langle \overline{b}\langle R\rangle.\, \mathbf{0}\rangle.\, \mathbf{0} \;\; a(x).\, x \;\; b(y).\, y \tag{1.1}$$

for which it holds that

$$\begin{array}{rcl} P & \longrightarrow & \overline{b}\langle R\rangle.\, \mathbf{0} \;\; b(y).\, y \\ & \longrightarrow & R\,. \end{array}$$

We consider now the translation of higher-order processes into the π-calculus. As mentioned before, it represents process passing by means of *reference passing*. Let $[\cdot]$ be the mapping from the higher-order π-calculus into the π-calculus defined as

$$\begin{array}{rcl} [\overline{a}\langle P\rangle.\, Q] & = & (\nu m)\, \overline{a}\langle m\rangle.\, ([Q] \;\; !m.\, [P]) \quad \text{with } m \notin \mathrm{fn}(P, Q) \\ [a(x).\, R] & = & a(x).\, [R] \\ [x] & = & \overline{x} \end{array}$$

and that is a homomorphism for the other constructs. Intuitively, the communication of a process P is represented by the communication of a unique name m that is used by the recipient to *trigger* as many copies of P as required. Now consider the translation of P in (1.1); it is given as follows, with m, n fresh in R,

$$\begin{array}{rcl} [P] & = & (\nu m)\, \overline{a}\langle \overline{m}\rangle.\, (\mathbf{0} \;\; !m.\, [\overline{b}\langle R\rangle.\, \mathbf{0}]) \;\; a(x).\, \overline{x} \;\; b(y).\, \overline{y} \\ & = & (\nu m)\, \overline{a}\langle \overline{m}\rangle.\, (\mathbf{0} \;\; !m.\, (\nu n)\, \overline{b}\langle n\rangle.\, (\mathbf{0} \;\; !n.\, [R])) \;\; a(x).\, \overline{x} \;\; b(y).\, \overline{y} \end{array}$$

we then have

$$\begin{aligned}
[P] &\longrightarrow (\nu m)\,(!m.\,(\nu n)\,\overline{b}\langle n\rangle.\,(\mathbf{0}\;\;!n.[R])\;\;\overline{m})\;\;b(y).\,\overline{y} \\
&\longrightarrow (\nu m)(\nu n)\,(\overline{b}\langle n\rangle.\,(\mathbf{0}\;\;!n.[R])\;\;!m.\,(\nu n)\,\overline{b}\langle n\rangle.\,(\mathbf{0}\;\;!n.[R]))\;\;b(y).\,\overline{y} \\
&\longrightarrow (\nu m)(\nu n)\,(!n.[R]\;\;!m.\,(\nu n)\,\overline{b}\langle n\rangle.\,(\mathbf{0}\;\;!n.[R])\;\;\overline{n}) \\
&\longrightarrow (\nu m)(\nu n)\,([R]\;\;!n.[R]\;\;!m.\,(\nu n)\,\overline{b}\langle n\rangle.\,(\mathbf{0}\;\;!n.[R])) \\
&\sim [R]
\end{aligned}$$

where $\sim$ stands for a relation that allows to disregard behaviorally irrelevant processes.

When First-Order Is Not Higher-Order. The above example should be sufficient to understand how process mobility is realized by means of reference passing in Sangiorgi's translation. Indeed, the movement of processes is represented as the movement of names that *refer* to processes. At this point it is useful to quote Cardelli and Gordon (2000) who, when introducing the Ambient calculus, criticize a reference-based approach to mobility:

> There is no clear indication that processes themselves move. For example, if a channel crosses a firewall (that is, if it is communicated to a process meant to represent a firewall), there is no clear sense in which the process has also crossed the firewall. In fact, the channel may cross several independent firewalls, but a process could not be in all those places at once.

As a matter of fact, what this remark reveals is the following: when process mobility is to be considered in conjunction with notions of observable behavior that explicitly account for the location in which behavior takes place, the reference-passing approach for mobility is *inadequate* to capture process movement. Translations such as Sangiorgi's are therefore *not robust enough* in the context of explicit notions of locality, such as the required by in the modelling of network-aware systems.

Let us elaborate on this point by means of an example. Consider the higher-order process

$$P \overset{\text{def}}{=} \overline{a}\langle T\rangle.\,Q\;\;a(x).\,(x\;\;x)\,. \tag{1.2}$$

It is easy to see that via a synchronization on a, P is able to produce two copies of T, running in parallel with the continuation Q, i.e.

$$P \longrightarrow Q\;\;T\;\;T\,.$$

Now suppose we extend our higher-order calculus with a basic form of localities. More precisely, let us assume that processes are of the form $\{P\}_l$ which intuitively represents the process P executing in the computation locus l. The reaction rule given before is extended accordingly; it allows interactions between complementary actions in two —possibly different— localities:

$$\{a(x).\, M + M'\}_m \;\; \{\overline{a}\langle R\rangle.\, N + N'\}_n \longrightarrow \{M\{^R\!/\!_x\}\}_m \;\; \{N\}_n\,.$$

Let us consider P', the located version of P in (1.2). Process P' involves two different localities s and r for sender and receiver processes, respectively:

$$P' \overset{\text{def}}{=} \{\overline{a}\langle T\rangle.\, Q\}_s \;\; \{a(x).\,(x \;\; x)\}_r\,.$$

The behavior of P' is essentially the same of P, except for the fact that T is associated to location r. Intuitively, this represents the *movement* of T in the space of locations both s and r belong to:

$$P' \longrightarrow \{Q\}_s \;\; \{T \;\; T\}_r\,.$$

Indeed, we now have an observable behavior of the system that is finer in that we are now able to tell not only that Q executes in parallel with two copies of T, but also that Q executes in location s whereas that $T \;\; T$ executes in location r. Let us consider the first-order representation of P' given by the extension of Sangiorgi's translation to the located case. (Without loss of generality we can assume that the translation $[\![\cdot]\!]$ is homomorphic also with respect to locations, i.e. $[\![\{P\}_l]\!] = \{[\![P]\!]\}_l$.) This way, we have

$$\begin{array}{rcl}
[\![P']\!] & = & \{(\nu m)\,\overline{a}\langle m\rangle.\,([\![Q]\!] \;\; !m.[\![T]\!])\}_s \;\; \{a(x).\,(\overline{x} \;\; \overline{x})\}_r \\
 & \longrightarrow & (\nu m)\,(\{[\![Q]\!] \;\; !m.[\![T]\!]\}_s \;\; \{\overline{m} \;\; \overline{m}\}_r) \\
 & \longrightarrow & (\nu m)\,(\{[\![Q]\!] \;\; [\![T]\!] \;\; !m.[\![T]\!]\}_s \;\; \{\overline{m}\}_r) \\
 & \longrightarrow\sim & \{[\![Q]\!] \;\; [\![T]\!] \;\; [\![T]\!]\}_s \;\; \{\mathbf{0}\}_r
\end{array}$$

which is certainly unsatisfactory under any reasonable notion of behavioral equivalence with explicit locations since, unlike the source term, process $[\![T \;\; T]\!]$ is executed in location s. It is clear that what moved in the translation was a pointer to the copies, rather than the processes themselves.

The morale of this example is that while translations such as Sangiorgi's are satisfactory in the case of "basic" higher-order languages, this is not necessarily the case for higher-order process calculi with specialized constructs, such as the ones required in global and ubiquitous computing scenarios. It is in this sense that we claim that Sangiorgi's representability result induced a generalized misconception, both on the nature of higher-order communication and on the applicability of the translation. This is certainly not an original insight; as a matter of fact, Sangiorgi and Walker (2001) comment on this issue, remarking on the potentially dangerous effects some other operators could have in Sangiorgi's translation. Vivas and Dam (1998) and Vivas and Yoshida (2002) have studied such effects in the case of higher-order languages involving dynamic binding. Also, the nature of the *passivation* operators introduced in

(Hildebrandt et al., 2004; Schmitt and Stefani, 2004) to represent the *suspension* of executing processes —as required in, e.g., forms of dynamic system reconfiguration— strongly suggests that they are not representable into some first-order setting. All these works thus provide compelling evidence of the need of developing the theory of higher-order process calculi *directly on them*, without going through intermediate translations.

1.3 This Dissertation

This dissertation studies expressiveness and decidability issues in higher-order concurrency. The research is centered around a *core calculus* for higher-order concurrency in which only the operators strictly necessary to obtain higher-order communication are retained. Next, we give an overview to expressiveness and decidability in concurrent languages in general, and in higher-order concurrency in particular. Then, we elaborate on the approach we shall follow in our research. Finally, we comment on the contributions and structure of the dissertation.

1.3.1 Expressiveness and Decidability in Higher-Order Concurrency

An important criterion for assessing the significance of a paradigm is its *expressiveness*. Expressiveness studies are concerned with formal assessments of the *expressive power* of a language or family of languages. The precise meaning of "expressive power" depends on the purpose, and several suitable definitions are possible. At the heart of all of them, however, is the notion of *encoding*: a map from the terms of a *source language* into those of a *target language*, subject to a set of *correctness criteria*.

The quest for a unified definition of encoding —in particular, a set of correctness criteria that a *good encoding* should enforce— has been a matter of research for some time now, and concrete proposals have been put forward. In spite of this, there is yet no general agreement on such a definition. In our view, a single, all-embracing definition of encoding is unlikely to exist, essentially because expressiveness studies may have many different purposes, and may be carried out over concurrent languages of a very diverse nature. This way, the set of criteria required in the definition of a taxonomy aimed at *relating* different process calculi should be different from, for instance, the criteria required when the interest is on *transferring* reasoning principles from one language to another. Indeed, whereas in the latter case the definition of encoding should impose rather strict criteria on the relationship between equivalent terms in both source and target languages, in the former case the adopted definition could well enforce milder forms of correspondence between equivalent terms, and/or consider criteria oriented at capturing precise aspects of the relationships of interest. Hence, differences between the two sets of criteria do not mean one is better than the other; they just reflect the different motivations underlying the respective expressiveness studies. Nevertheless, considering the "quality" of an encoding is still interesting because, as we shall see, there is a direct relationship between the precise definition of encoding and the significance of the results obtained with it. We treat this issue in length in Chapter 2.

As hinted at above, expressiveness has been little studied for higher-order process calculi. Most previous works address issues of *relative* expressiveness: higher-order calculi (both sequential and concurrent) have been compared with first-order calculi, but mainly as a way of investigating the expressiveness of the π-calculus and similar formalisms. In addition to the representability result in (Sangiorgi, 1992), the expressiveness of higher-order process calculi was studied in (Sangiorgi, 1996b), where variants of the π-calculus with different degrees of *internal mobility* are related to typed variants of the Higher-Order π-calculus. Interestingly, this work presents encodings of (variants of) the π-calculus into *strictly* higher-order process calculi, i.e., calculi in which only pure process passing is allowed and no name-passing is present. The only other result on the expressiveness of pure process passing we are aware of is (Bundgaard et al., 2006), where an encoding of the π-calculus into Homer —a higher-order process calculi with locations (Hildebrandt et al., 2004)— is presented. Encodings of variants of the π-calculus into the Higher-Order π-calculus were first given in (Sangiorgi, 1996b) and later consolidated in (Sangiorgi and Walker, 2001), where the abstraction mechanism of the higher-order π-calculus is exploited. Thomsen (1990) and Xu (2007) have proposed encodings of π-calculus into Plain CHOCS. These encodings make essential use of the relabeling operator of Plain CHOCS.

The expressiveness of concurrent languages is closely related to *decidability issues*. Given a concurrent language, it is legitimate to ask whether or not its expressive power is related to the decidability of some property of interest. Examples include properties related with *behavioral equivalences* (e.g. strong bisimilarity), *termination of processes* (e.g. convergence), and graph-like structures (e.g. reachability and coverability). An appealing question here is "what is the most expressive fragment of the language in which the property is decidable?" There is a trade-off between expressiveness and decidability: most interesting decision problems are generally undecidable for very expressive languages. Hence, given a process calculus and some property of interest, a common research direction is identifying the largest sub-calculus for which the property is decidable. Studies dealing with the interplay of expressiveness and decidability are relevant in that they provide support for *verification*: they might pave the way for the implementation of tools, or provide insights on the aspects that might be sensible for verification purposes.

Studies of decidable properties for higher-order process calculi are scarce. The only work we are aware of is (Bundgaard et al., 2009), in which the interest is on the decidability of barbed bisimilarity in the context of Homer.

1.3.2 Approach

We shall follow a *direct* and *minimal* approach for investigating the expressive power and decidability of higher-order process calculi.

Our approach is *direct* in that we abandon the idea of studying the foundations of higher-order concurrency by means of translations into first-order languages. Based on the inadequacy of studying higher-order concurrency through first-order translations (as discussed in the previous section), we advocate that foundational studies for higher-order process calculi must be carried out directly on them and exploit their peculiarities. While we concentrate on expressiveness and decidability issues, this direct approach is in concordance with that advocated by recent works on other aspects of the theory of higher-order process calculi, such as behavioral theory (see, e.g., (Lenglet et al., 2008; Sato and Sumii, 2009)) and type systems (Demangeon et al., 2010).

On the other hand, our approach is *minimal* in that the research shall be centered around a *core calculus* for higher-order concurrency in which only the operators strictly necessary to obtain higher-order communication are retained. The calculus, called HOcore, aims to be the simplest, non-trivial process calculus featuring higher-order concurrency. In particular:

- HOcore has *no name-passing*, so processes are the only kind of values that can be passed around in communications. This is in sharp contrast to most higher-order process calculi in the literature, in which *both* name-passing and process-passing are present.

- HOcore has *no restriction operator*, thus all channels are global, and dynamic creation of new channels is impossible. As such, the behavior of a concurrent system described in HOcore is completely *exposed*. Also, it is worth noticing that the syntax of higher-order process calculi (including HOcore) usually omits primitive operators for infinite behavior (as replication), as they can be encoded by mimicking the structure of fixed-point combinators in the λ-calculus. Known encodings of fixed-point combinators require restriction; therefore, the lack of restriction in HOcore is directly related to its ability of expressing infinite behavior. While in most of the dissertation we consider HOcore (or variants of it) without restriction, we shall find it useful to consider an extension with restriction useful when examining synchronous and polyadic communication.

- HOcore has *no output prefix* so it is an *asynchronous* calculus. It is well-known that asynchronous communication is easier to implement and maintain that synchronous communication. As such, it appears as the *most elemental* communication discipline one could adopt. Asynchrony represented as the absence of continuations after output actions is the main feature of the *asynchronous π-calculus*, which was proposed in seminal papers by Boudol (1992) and Honda and Tokoro (1991), and thoroughly studied since then. Within concurrency theory, the expressive power of asynchrony has been studied by Palamidessi (2003) (see also (Cacciagrano et al., 2007; Beauxis et al., 2008)), who showed that in the π-calculus with choice synchronous communication is more expressive than asynchronous one. Even if the same phenomenon should not necessarily carry over to a higher-order setting —we shall address this issue in this dissertation—, Palamidessi's result ought be taken as an additional evidence of the simplicity that asynchrony might embody in process calculi.

The minimality of HOcore is convenient in that it allows us to focus on higher-order communication and its associated phenomena, without being shadowed by complex constructs nor by first-order interactions; studies of expressiveness and decidability for HOcore will therefore reflect the inherent to *pure* process passing and shed light on their intrinsic nature.

1.3.3 Contributions and Structure

The dissertation contributes to the theory of higher-order concurrency with several original results on the expressiveness and decidability of HOcore and a number of selected variants of it. Our results complement the few ones in the literature, and deepen and strengthen our understanding of the theory core higher-order process calculi as a whole. More precisely, our contributions are structured as follows.

Chapter 2: Preliminaries. This chapter provides the theoretical background for the thesis. We introduce fundamental concepts on process calculi, higher-order process calculi, and expressiveness of concurrent languages.

Chapter 3: HOcore and its Expressiveness. We introduce HOcore, a core calculus for higher-order concurrency. We study the expressive power by encoding basic forms of choice and input-guarded replication. Such derived constructs are then used to define an encoding of Minsky machines into HOcore, which demonstrates that the language is Turing complete. The encoding is deterministic and termination preserving; as such, properties such as *termination* (i.e. the absence of divergent computation) and *convergence* (i.e. the existence of a non-diverging computation) are immediately shown to be undecidable.

Chapter 4: Behavioral Theory of HOcore. We show that in HOcore strong bisimilarity is *decidable*. To the best of our knowledge, HOcore is the first concurrent formalism that is Turing complete *and* for which bisimilarity is decidable. Furthermore, strong bisimilarity is shown to be a congruence, and to coincide with other well-established behavioral equivalences for higher-order calculi. A sound and complete axiomatization of strong bisimilarity is given, and used to obtain complexity bounds for bisimilarity checking. The limits of decidability are explored by considering an extension of HOcore with static (top-level) restrictions. For the extension with four of such restrictions, bisimilarity is shown to be *undecidable*. This result is obtained through an encoding of the Post correspondence problem (PCP).

Chapter 5: Expressiveness of Forwarding and Suspension. We study $\text{Ho}^{-\text{f}}$, the fragment of HOcore that results from forbidding *nested output actions* in communication objects. This represents a limitation of the *forwarding capabilities* of HOcore. The expressiveness of $\text{Ho}^{-\text{f}}$ is analyzed using decidability of termination and convergence as a yardstick. As in HOcore, in $\text{Ho}^{-\text{f}}$ convergence is still *undecidable*, a result obtained by exhibiting an unfaithful encoding of Minsky machines. In contrast, termination is shown to be *decidable*. This result is obtained by appealing to the theory of well-structured transition systems. To the best of our knowledge, this is the first time such a theory is used in the higher-order setting.

Decidability of termination suggests a loss of expressive power when passing from HOcore to $\text{Ho}^{-\text{f}}$. Then, as a way of recovering such power, we consider $\text{HoP}^{-\text{f}}$, the extension of $\text{Ho}^{-\text{f}}$ with a *passivation* construct that allows for process *suspension* at run time. We show that in $\text{HoP}^{-\text{f}}$, a faithful encoding of Minsky machines becomes possible. This implies that in $\text{HoP}^{-\text{f}}$ both convergence and termination are *undecidable*. To the best of our knowledge, ours is the first result on the expressiveness and decidability of passivation operators in the higher-order setting.

Chapter 6: Expressiveness of Synchronous and Polyadic Communication. We study the expressive power of extensions of HOcore with restriction. We call such an extension AHO. As a first encodability result, we show that AHO is expressive enough to encode *synchronous communication*. We then move to study the expressiveness of SHO^{n}, the extension of HOcore with name restriction, synchronous communication, and polyadic communication of arity n. We consider the family of higher-order process calculi given by varying the polyadicity of such an extension. The main result is that polyadicity induces a hierarchy of strictly increasing expressiveness: polyadic communication of arity n (as in SHO^{n}) cannot be encoded into polyadic communication of arity $n-1$ (as in SHO^{n-1}). Furthermore, we show that SHO^{n}_{a} —the extension of SHO^{n} with *abstraction-passing*— cannot be encoded into SHO^{n}.

Chapter 7: Conclusions and Perspectives. We draw conclusions from the research and discuss perspectives of future work.

The calculi studied in the dissertation are depicted in Figure 1.1.

Origin of the Chapters. Most of the material in this dissertation has been previously presented in international conferences and appear in the respective proceedings. Even if many

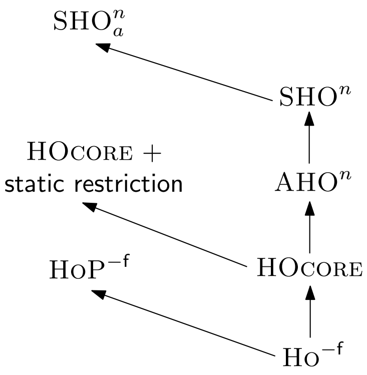


Figure 1.1: The higher-order process calculi studied in this dissertation. An arrow indicates language inclusion.

improvements have been made with respect to the published material, we think that the basic ideas behind the results remain the same.

- HOcore and its behavioral theory as presented in Chapters 3 and 4 has been published as the paper (Lanese et al., 2008).

- The expressiveness of forwarding in HOcore, as presented in Chapter 5, is based on results first published in the paper (Di Giusto et al., 2009a).

- The expressiveness of polyadic communication as discussed in Chapter 6 is based on results published as the extended abstract (Lanese et al., 2009).

There are some results original to this dissertation; this unpublished material will be explicitly mentioned in the corresponding chapter.

Chapter 2

Preliminaries

This chapter provides the theoretical background for the dissertation. It is in three sections. In Section 2.1 we introduce the basic terminology and concepts used in the dissertation. In order to do so, we present a description of CCS (Milner, 1989) and of the π-calculus (Milner et al., 1992). In Section 2.2 we introduce *higher-order process calculi*: we review their origins and behavioral theory. The higher-order π-calculus, as well as Sangiorgi's representability result, are detailed there. Section 2.3 introduces main issues in the analysis of the expressiveness of concurrent languages. We give an overview to the most common kinds of expressiveness studies and the techniques used to carry them out. Furthermore, previous efforts on studying the expressiveness of higher-order languages are reviewed.

2.1 Technical Background

2.1.1 Bisimilarity

Broadly speaking, *behavioral equivalences* allow to determine when the *behavior* of two concurrent system can be considered as *equal*. There are many plausible motivations for aiming at definitions of behavioral equivalences. For instance, one would like the behavior of the implementation of system to be behaviorally equivalent to that of its specification; similarly, in a component-based system it is generally desirable to replace a component with a new one that features *at least* the same possibilities for behavior. Accordingly, many definitions of behavioral equivalences for concurrent systems have been proposed; notable notions include *trace equivalence* —which equates two processes if they can perform the same finite sequences of transitions— and the *testing framework* (De Nicola and Hennessy, 1984), in which the behavior of two processes is deemed as equal if they *pass the same tests* provided by an external *observer*. In this context, *bisimilarity* is widely accepted as the finest behavioral equivalence one would like to impose on processes. Following (Sangiorgi, 2009), we now define bisimilarity and state a few of its fundamental properties.

A fundamental notion is that of *Labeled Transition System* (LTS in the sequel).

Definition 2.1. *A* Labelled Transition System *(LTS) is a triple* $(S, T, \{\xrightarrow{t}: t \in T\})$ *where* S *is a set of* states, T *is a set of* (transition) labels, *and* $\xrightarrow{t} \subseteq S \times S$ *for each* $t \in T$ *is the* transition relation*.*

It is customary to write $P \xrightarrow{\alpha} Q$ to denote the fact that $(P, Q) \subseteq \xrightarrow{\alpha}$. In the context of concurrency theory, it is natural to relate states and *processes*, and labels as the *actions* processes can perform. This way, $P \xrightarrow{\alpha} Q$ is indeed a *transition* which represents that process P can perform α and evolve into Q. The transition relation for a process language is generally defined by means of a set of *transition rules* which realize the intended behavior of each construct of the language. In what follows, we say that a *process relation* is a binary relation on the states of an LTS.

Definition 2.2 (Bisimilarity)**.** *A process relation* $\mathcal{R}$ *is a* bisimulation *if, whenever* $P\mathcal{R}Q$*, for all* α *we have that:*

1. *for all* P' *with* $P \xrightarrow{\alpha} P'$*, there is* Q' *such that* $Q \xrightarrow{\alpha} Q'$ *and* $P'\mathcal{R}Q'$*;*
2. *the converse, on the transitions emanating from* Q*: for all* Q' *with* $Q \xrightarrow{\alpha} Q'$*, there is* P' *such that* $P \xrightarrow{\alpha} P'$ *and* $P'\mathcal{R}Q'$*.*

Bisimilarity*, written* $\sim$*, is the union of all bisimulations; thus* $P \sim Q$ *if there is a bisimulation* $\mathcal{R}$ *with* $P\mathcal{R}Q$*.*

Given this definition, the *bisimulation proof method* naturally follows: to determine that two processes P and Q are bisimilar, it is sufficient to exhibit a bisimulation relation containing (P, Q). It is useful to state a few fundamental properties of bisimilarity.

Theorem 2.1 (Basic Properties of Bisimilarity)**.** *Given* $\sim$*, it holds that:*

1. $\sim$ *is an* equivalence relation*, i.e., it is reflexive, symmetric, and transitive.*
2. $\sim$ *is itself a* bisimulation*.*

Item (2) is insightful in that it allows to grasp the *circular* flavor of bisimilarity: bisimilarity itself is a bisimulation, and is part of the union on which it is defined. Hence, the following theorem holds.

Theorem 2.2. *Bisimilarity is the largest bisimulation.*

2.1.2 A Calculus of Communicating Systems

We introduce a number of relevant concepts of CCS, following the presentation in Milner (1989).

CCS departs from theories of sequential computation by focusing on the notion of *interaction*: a concurrent system *interacts* with its environment which realizes the behavior of the system through *observations*. In CCS —like in other process calculi such as ACP and CSP— the overall behavior of a system is entirely determined by the *atomic actions* it performs. The distinguishing principle in CCS is that the notion of interaction is equated to that of observation: not only actions are *observable*, but we observe an action produced by the system by *interacting* with it, that is, by performing its complementary action, or *coaction*. We then say that the two participants, system and observer, have *synchronized* in the action by means of this mutual observation.

Syntax. We shall assume a set of *names* $\mathcal{N} = \{a, b, c, \dots\}$, as well as a disjoint set of *co-names* defined as $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$. There is a set of *labels* defined as $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$; we let $l, l', \dots$ range over $\mathcal{L}$. Labels give an account of the observable behavior of the system. We shall use K, L for subsets of $\mathcal{L}$; $\overline{L}$ stands for the set of complements of the labels in L. We consider the distinguished symbol τ representing the *internal* or *silent* action that results from synchronizations. We then define $\mathcal{A} = \mathcal{L} \cup \tau$ to be the set of *actions*; α, β range over $\mathcal{A}$. In the spirit of the above discussion, actions a and $\overline{a}$ are thought of as complementary; this way, $\overline{\overline{a}} = a$ and $\overline{\tau} = \tau$. The set of CCS processes expressing finite behavior is given as follows:

Definition 2.3. *The set of finite CCS processes is given by the following syntax:*

$$P, Q, \dots ::= \sum_{i \in I} \alpha_i.\, P_i \;\Big|\; P \backslash a \;\Big|\; P_1 \;\; P_2$$

where I is an indexing set.

The *summation* $\sum_{i \in I} \alpha_i.\, P_i$ represents the process that is able to perform one and only one of its actions α_i, and then behaves as its associated P_i. It is customary to write $\mathbf{0}$ —nil, the process that does nothing— in case $\mid I \mid = 0$, $\alpha.\, P$ if $\mid I \mid = 1$, and “+” for binary sum. The restriction $P \backslash a$ behaves exactly as P but it cannot offer neither a or $\overline{a}$ to its surrounding environment. Both a and $\overline{a}$ are then said to be *bound* in P; we shall use $\mathrm{fn}(P)$ to denote the set of *free names*, i.e., not bound, in P; the *bound names* of P, $\mathrm{bn}(P)$, are those with a bound occurrence in P. The *parallel composition* $P \;\; Q$ allows P and Q to run concurrently: either P or Q may perform an action, or they can synchronize by performing complementary actions.

$$\textsc{Sum}\quad \sum_{i\in I} \alpha_i.\,P_i \xrightarrow{\alpha_j} P_j \;\; \text{if } j \in I \qquad \textsc{Res}\quad \frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P\backslash a} \;\; \text{if } a \notin \{\alpha, \overline{\alpha}\,\}$$

$$\textsc{Par1}\quad \frac{P \xrightarrow{\alpha} P'}{P \;\; Q \xrightarrow{\alpha} P' \;\; Q} \qquad \textsc{Tau}\quad \frac{P \xrightarrow{l} P' \qquad Q \xrightarrow{\overline{l}} Q'}{P \;\; Q \xrightarrow{\tau} P' \;\; Q'}$$

Figure 2.1: An LTS for CCS. Rule Par2, the symmetric of Par1, is omitted.

Semantics and Infinite Behavior. The operational semantics of CCS is given by an LTS in which the set of processes is the set of states, and the set of labels is taken to be $\mathcal{A}$, the set of actions in CCS. The transition relation is given by the set of transition rules in Figure 2.1.

Let us move now to the different ways of expressing *infinite behavior*. We consider *recursion* and *replication*. In order to represent *recursion* a denumerable set of *constants*, ranged over by D, is assumed. It is also assumed that each constant D has associated a (possibly recursive) defining equation of the form $D \overset{\text{def}}{=} P$. The extension of (finite) CCS with recursion is then is obtained by adding the production $P ::= D$ to the grammar in Definition 2.3, and by extending the operational semantics in Figure 2.1 with the following transition rule

$$\textsc{Cons}\quad \frac{P \xrightarrow{\alpha} P' \qquad D \overset{\text{def}}{=} P}{D \xrightarrow{\alpha} P'}\,.$$

As Busi et al. (2009) remark, recursive behavior defined by means of constants can be intuitively assimilated to infinite behavior "in depth", in that process copies can be nested at an arbitrary depth by using constant application. This is in sharp contrast to the kind of infinite behavior provided by *replication*: by means of the replication operator $!P$ it is possible to obtain an unbounded number of copies of P; such copies, however, are all at the same level, thus defining infinite behavior "in width". The extension of (finite) CCS with replication is obtained by adding the production $P ::= \,!P$ to the grammar in Definition 2.3, and by extending the operational semantics in Figure 2.1 with the following transition rule

$$\textsc{Repl}\quad \frac{P \;\; !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}\,.$$

A word on *proof techniques* is most convenient at this point. Defining the semantics in terms of a LTS provides us automatically with two basic proof techniques, both of which are forms of *induction*: one on the structure of process terms (*structural induction*), and one on the transition rules (*transition induction*). The finitary character of inductive proof techniques is in contrast with the infinite behavior concurrent systems generally exhibit. As a result, when addressing the issue of *equality* of concurrent systems, one needs to appeal to *coinductive* proof techniques. Bisimilarity as introduced in Section 2.1.1, is probably the most representative coinductive proof-technique.

2.1.3 More on Behavioral Equivalences

Having introduced the notion of bisimilarity, and some basic notions of CCS, we find it useful to informally present some additional concepts on behavioral equivalences. The discussion here is intended to introduce useful terminology; technical accounts of the concepts mentioned here can be found elsewhere (see, e.g., (Sangiorgi, 2009; Milner, 1989)).

It is desirable to require bisimilarity to be preserved by all process contexts. This allows to replace, in any process expression, a subterm with a bisimilar one. An equivalence relation with this property is said to be a *congruence*. Proofs of congruence combine inductive and coinductive arguments: the former are necessary as the syntax of the processes is defined inductively, whereas the latter are required in that bisimilarity is a coinductive definition. In the case of CCS we have the following.

Theorem 2.3. *In CCS, $\sim$ is a congruence relation.*

When bisimilarity is decidable, it may be possible to give an algebraic characterization of it, or *axiomatization*. The axiomatization of an equivalence on a set of terms consists essentially of some equational axioms that suffice for proving all and only the equations among the terms that are valid for the given equivalence. These axioms are used together with rules of equational reasoning, which include reflexivity, symmetry, transitivity, and congruence rules that allow to replace any subterm of a process with an equivalent one. A bit more formally, given a set of axioms $\mathcal{S}$, it is usual to write $\mathcal{S} \vdash P = Q$ if one can derive $P = Q$ using the axioms in $\mathcal{S}$ and the laws of equational reasoning. The objective is then to show that the axiomatization is a full characterization of bisimilarity, i.e., that it is both sound and complete with respect to bisimilarity:

$$P \sim Q \text{ if and only if } \mathcal{S} \vdash P = Q\,. \tag{2.1}$$

While establishing soundness (i.e., the backward direction in (2.1)) is in general easy, establishing completeness (i.e., the forward direction in (2.1)) often involves defining some standard syntactic form for processes and requires more effort. This the case of, e.g., finite-state CCS processes as studied by Milner (1989).

We have seen that CCS considers the special action τ as a form of internal activity. Often it is useful to describe concurrent behavior by abstracting from such internal actions. This gives rise to *weak* transition relations, denoted $\Rightarrow$ and $\xRightarrow{\alpha}$. While $P \Rightarrow Q$ is used to mean that P can evolve to Q by performing any number of internal actions (even zero), $P \xRightarrow{\alpha} Q$ means that P can evolve to Q as a result of an evolution that includes an action α, but may involve any number of internal actions before and after α. As such, $\xRightarrow{\tau}$ is different from $\Rightarrow$ as the former guarantees that *at least* one internal action has been performed. More formally, we have the following.

Definition 2.4 (Weak transitions). .

- *Relation $\Rightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$. That is, $P \Rightarrow P'$ holds if there if there is $n \geq 0$ and processes $P_1, \ldots, P_n$ with $P_n = P'$ such that $P \xrightarrow{\tau} P_1 \cdots \xrightarrow{\tau} P_n$. (Notice that $P \Rightarrow P$ holds for all processes.)*
- *For all $\alpha \in T$, relation $\xRightarrow{\alpha}$ is the composition of the relations $\Rightarrow$, $\xrightarrow{\alpha}$, and $\Rightarrow$. That is, $P \xRightarrow{\alpha} P'$ holds if there are P_1, P_2 such that $P \Rightarrow P_1 \xrightarrow{\alpha} P_2 \Rightarrow P'$.*

With the aid of weak transitions, it is possible to define *weak bisimulation* and *weak bisimilarity*, as in the following definition.

Definition 2.5. *A process relation $\mathcal{R}$ is a* weak bisimulation *if, whenever $P\mathcal{R}Q$, for all α we have:*

1. *for all P' with $P \xRightarrow{\alpha} P'$ there is a Q' such that $Q \xRightarrow{\alpha} Q'$ and $P'\mathcal{R}Q'$;*
2. *for all P' with $P \xRightarrow{\tau} P'$ there is a Q' such that $Q \Longrightarrow Q'$ and $P'\mathcal{R}Q'$;*
3. *the converse of (1) and (2), on the actions from Q.*

P and Q are weakly bisimilar, *written $P \approx Q$, if $P\mathcal{R}Q$ for some weak bisimulation $\mathcal{R}$.*

We now discuss the ideas behind *barbed bisimilarity* (Milner and Sangiorgi, 1992). A transition $P \xrightarrow{\alpha} P'$ of an LTS intuitively describes a pure synchronization between P and its external environment along a port a mentioned in α. This is but one particular of concurrent interaction; a natural question that arises is how to adapt the idea of bisimulation to other kinds of interaction. The idea is to set a bisimulation in which the observer has a *minimal* ability to observe actions and/or process states. This yields a bisimilarity, namely indistinguishability under such observations, which in turns yields a congruence over terms, namely bisimilarity in all contexts. The bisimilarity is called *barbed bisimilarity*; the congruence is called *barbed congruence*.

The main assumption in the barbed setting is the existence of a *reduction relation* in the language. Such a relation is intended to express an evolution step of a term in which no intervention from the environment is required. In CCS, such a relation is $\xrightarrow{\tau}$. The reduction relation represents the most fundamental notion in the operational semantics of a language. The *reduction semantics* of a language is then an approach to operational semantics in which the meaning is only attached to reductions; it explains how a system can evolve independently of its environment. This approach is then in clear contrast to that underlying a labeled transition system.

In barbed bisimilarity the clauses involve challenges only on reductions. In addition, equal processes should exhibit the same *barbs*—i.e., predicates representing basic observables of the states. Barbs are of the essence to obtain an adequate discriminating power. Barbed congruence is a contextual equivalence: it is the closure of barbed bisimilarity over contexts. The definition of barbs we shall be interested in is as follows.

Definition 2.6. *Given a visible action α, the* observability predicate $\downarrow_\alpha$ *holds for a process P if, for some P', $P \xrightarrow{\alpha} P'$.*

We now define strong barbed bisimulation.

Definition 2.7 (Barbed bisimilarity)**.** *A process relation $\mathcal{R}$ is said to be a* barbed bisimulation *if whenever $P \stackrel{\centerdot}{\sim} Q$ it implies:*

1. *whenever $P \longrightarrow P'$ then $Q \longrightarrow Q'$ and $P'\mathcal{R}Q'$;*

2. *for each visible action α, if $P \downarrow_\alpha$ then $P \downarrow_{\overline{\alpha}}$.*

Barbed bisimilarity*, written $\stackrel{\centerdot}{\sim}$, is the union of all barbed bisimulations.*

The weak version of Definition 2.7 is obtained in the standard way. Let $\Longrightarrow$ be the reflexive and transitive closure of $\longrightarrow$ and $\Downarrow_a$ be defined as $\Longrightarrow\downarrow_a$. Then, *weak barbed bisimulation*, written $\stackrel{\centerdot}{\approx}$, is defined by replacing the reduction $Q \Longrightarrow Q'$ with $Q \Longrightarrow Q'$ and the predicate $Q \downarrow_\mu$ with $Q \Downarrow \mu$. As mentioned before, by quantifying over contexts, we obtain *barbed congruence*:

Definition 2.8. *Two processes P and Q are said to be* strongly barbed congruent*, written $P \simeq Q$, if for every context $C[\cdot]$, it holds that $C[P] \stackrel{\centerdot}{\sim} C[Q]$.*

We obtain *weak barbed congruence*, written $\cong^c$, by replacing $\stackrel{\centerdot}{\sim}$ with $\stackrel{\centerdot}{\approx}$ in the definition above.

A main drawback of the notion of barbed congruence is that the universal quantification on contexts, which can make it impractical to use in proofs. The challenge is then to find tractable characterizations of barbed congruence. A well-established approach here is to use (labeled) bisimilarities: the objective is to find a bisimilarity that is both *sound* and *complete* with respect to barbed congruence. That is, a notion of bisimilarity that both *includes* and *contains* barbed congruence. While for the case of CCS and the π-calculus effective characterizations of barbed congruence have been thoroughly studied (see, e.g., (Sangiorgi and Walker, 2001)), we shall see that this is not quite the case for higher-order process calculi, in which the situation is much less clear.

2.1.4 A Calculus of Mobile Processes

We introduce the (polyadic) π-calculus following the presentation given in (Sangiorgi, 1992, 1993); this will make the introduction of the higher-order π-calculus easier. The reader is referred to (Milner et al., 1992; Sangiorgi and Walker, 2001) for complete references on the π-calculus.

The π-calculus departs from CCS with the capability of sending (first-order) values along communication channels. Its significance derives from the fact that such values include the set of communication channels; new communication channels can be created dynamically, and shared among processes, possibly in a restricted way. This is most useful to represent dynamic communication topologies.

Syntax. We use $a, b, c, \ldots, x, y, z, \ldots$ to range over *names* (or *channels*) and $P, Q, R, T, \ldots$ to range over processes. We use a tilde to represent *tuples* of elements; this way, given a name y, $\widetilde{y}$ stands for a tuple of names. The set of π-calculus processes is given by the following definition.

Definition 2.9. *The set of π-calculus processes is given by the following syntax:*

$$P, Q, \ldots ::= \sum_{i \in I} \alpha_i.\, P_i \;\Big|\; P_1 \;\; P_2 \;\Big|\; (\nu x)\, P \;\Big|\; [x = y]P \;\Big|\; D\langle \widetilde{x} \rangle$$

where I is any finite indexing set. The set of prefixes *is given by*

$$\alpha ::= x(\widetilde{y}) \;\Big|\; \overline{x}\langle \widetilde{y} \rangle\,.$$

As in CCS, we assume that each constant D has a defining equation of the form $D \overset{\text{def}}{=} (\widetilde{x})P$, where the parameters $\widetilde{x}$ collect all names which may occur free in P. Some constraints to tuples in input and output prefixes are in order. In an input prefix $x(\widetilde{y})$, tuple $\widetilde{y}$ is required to be made of pairwise distinct elements. We omit brackets $(\,)$ and $\langle\,\rangle$ when the tuple is empty. Also, tuple $\widetilde{y}$ is required to be finite in both input and output prefixes. This is not the case for the tuple $\widetilde{x}$ in constant definitions and applications; hence, it can be infinite.

The intuitive semantics of processes is as expected. An input-prefixed process $x(\widetilde{y}).\,P$ waits for a tuple $\widetilde{z}$ to be transmitted along name x; once this occurs, the process P in which $\widetilde{y}$ has been instantiated by $\widetilde{z}$ executes. An output-prefixed process $\overline{x}\langle\widetilde{y}\rangle.\,P$ sends tuple $\widetilde{y}$ along x and then behaves like P. The *matching* operator $[x = y]P$ is used to test for equality of the names x and y. The intuition behind the restriction operator is somewhat similar to that in CCS: $(\nu x)\, P$ makes name x local to P; thus x becomes a new, unique name, distinct from all those external to P. We often write $(\nu\widetilde{x})\, P$ to stand for the process $(\nu x_1)(\nu x_2)\ldots(\nu x_n)\, P$. The semantics and notation for (guarded) summation follow those in CCS. In particular, we shall use $+$ to represent binary sum.

We have already commented on the use of constants to represent infinite behavior. Notice that it is possible to encode replication using constants. It is worth noticing that, given $D = \langle\widetilde{x}\rangle P$, in an *application* $D(\widetilde{y})$ tuple $\widetilde{y}$ must be of the same length as $\widetilde{x}$. This kind of potential disagreements on the arities of tuples, as well as some other aspects of the name-passing discipline, are enforced by the use of appropriate *type systems* on names.[1] For the sake of conciseness, we do not elaborate on the definitions and properties of sorts. As such, along the chapter we always assume well-sorted processes; we use notation $x : y$ to mean that names x and y have the same sort. If $D \overset{\text{def}}{=} (\widetilde{x})P$ and $\widetilde{x}$ is not empty then D and $(\widetilde{x})P$ are called *abstractions*. Abstractions and processes are *agents*. We use $F, E, \ldots$ and A to range over abstractions and agents, respectively.

Notions of free and bound names are as expected: in $a(\widetilde{b}).\, P$, $(\nu \widetilde{b})\, P$, and $(\widetilde{b})P$ all free occurrences of names $\widetilde{b}$ in P are *bound*. The sets of free and bound names of an agent A are denoted $\mathrm{fn}(A)$ and $\mathrm{bn}(A)$, respectively. Notice that if $A = D\langle \widetilde{x} \rangle$ then $\mathrm{fn}(A) = \widetilde{x}$ and $\mathrm{bn}(A) = \emptyset$. Name substitution is a function from names to names. Given a vector of distinct names $\widetilde{x}$, we write $\{\widetilde{y}/\widetilde{x}\}$ for the substitution that maps the x_i-th name in $\widetilde{x}$ to the y_i-th name in $\widetilde{y}$, and maps all names not in $\widetilde{x}$ to themselves. We assume standard definitions of substitution and α-conversion on processes, with possible renamings so as to avoid capture of free names. In what follows we shall be working modulo α-conversion, and hence we decree two processes as equal if one is α-convertible into the other.

Operational Semantics. We present now a reduction semantics and an LTS for the π-calculus. As argued before, the reduction semantics is intended to capture the behavior that is intrinsic to a process, that is, the behavior that does not include the potential interactions between the process and its environment. Central to the reduction semantics is the notion of *structural congruence* that allows flexibility in the syntactic structure of the process, thus promoting interactions to occur.

Structural congruence, denoted $\equiv$, is the smallest congruence over the set of π-calculus processes that satisfies the following rules:

1. $P \equiv Q$ if P is α-convertible to Q;
2. abelian monoid laws for $+$: $P + \mathbf{0} \equiv P$, $P + Q \equiv Q + P$, $(P + Q) + R \equiv P + (Q + R)$;
3. abelian monoid laws for : $P \;\; \mathbf{0} \equiv P$, $P \;\; Q \equiv Q \;\; P$, $(P \;\; Q) \;\; R \equiv P \;\; (Q \;\; R)$;
4. laws for restriction: $\nu x \mathbf{0} \equiv \mathbf{0}$, $\nu x\, \nu y P \equiv \nu y\, \nu x P$, $(\nu x P) \;\; Q \equiv \nu x(P \;\; Q)$ if $x \notin \mathrm{fn}(Q)$;
5. law for match: $[x = x]P \equiv P$;
6. law for constants: if $D \overset{\text{def}}{=} (\widetilde{x})P$ and $\widetilde{x} : \widetilde{y}$ then $D\langle \widetilde{y} \rangle \equiv P\{\widetilde{y}/\widetilde{x}\}$. (In case of replication is used: $!P \equiv P \;\; !P$.)

[1] In early proposals of the π-calculus (see, e.g., (Milner, 1991)) discipline on names was enforced by the notion of *sorting*. The presentation of the first- and higher-order π-calculus in (Sangiorgi, 1992) relies on sorts. In (Pierce and Sangiorgi, 1996) the notion of sort was refined into the notion of typing for processes.

$$\textsc{Com}\quad (\cdots + x(\widetilde{y}).\,P)\;\;(\cdots + \overline{x}\langle\widetilde{z}\rangle.\,Q) \rightarrow P\{\widetilde{z}/\widetilde{y}\}\;\;Q$$

$$\textsc{Par}\quad \frac{P \rightarrow P'}{P\;\;Q \rightarrow P'\;\;Q} \qquad \textsc{Res}\quad \frac{P \rightarrow P'}{\nu x P \rightarrow \nu x P}$$

$$\textsc{Struct}\quad \frac{P \equiv Q \qquad Q \rightarrow Q' \qquad Q' \equiv P'}{P \rightarrow P'}$$

Figure 2.2: Reduction semantics for the π-calculus.

The notion of interaction is formalized by the *reduction rules* given in Figure 2.2.

We now present the semantics in terms of a labelled transition system. It is actually the *early* semantics for the π-calculus: the bound names of an input are instantiated as soon as possible, namely in the rule for input. (This is contrast to the *late* semantics, in which such an instantiation takes place later, in the rule for communication.) Actions can take three possible forms. In addition to the silent action τ that represents interaction, we have the following:

$P \xrightarrow{x\langle\widetilde{y}\rangle} P'$ which stands for an *input action*: x is the name at which it occurs, while $\widetilde{y}$ is the tuple of names which are received.

$P \xrightarrow{(\nu\widetilde{y'})\,\overline{x}\langle\widetilde{y}\rangle} P'$ which stands for an *output action*, namely the output of names $\widetilde{y}$ at x. It always holds that $\widetilde{y'} \subseteq \widetilde{y} - x$. Tuple $\widetilde{y'}$ represents those private names that are emitted from P, carried out of their current scope. This is commonly known as *scope extrusion*.

In both cases, x is the *subject* and $\widetilde{y}$ is the *object* part of the action. There is a difference in the brackets of input *prefixes* and input *actions*: they are round in the former and angled in the latter. This is meant to emphasize the fact that in the input prefix $x(\widetilde{y})$ names in $\widetilde{y}$ are binders (i.e. placeholders waiting to be instantiated), whereas in the input action $x\langle\widetilde{y}\rangle$ they represent values (i.e. binders already instantiated).

We use μ to represent the label of a generic action. Given an action μ, the bound and free names of μ, denoted bn(μ) and fn(μ), respectively, is as follows:

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

The set of names of μ is defined as n(μ) = fn(μ) ∪ bn(μ). The labeled transition system is given in Figure 2.3. To conclude this introduction to the π-calculus, it is worth mentioning that, up to structural congruence, the reduction semantics $\rightarrow$ is exactly the relation $\xrightarrow{\tau}$ of the

$$\textsc{Alp}\quad \frac{P' \xrightarrow{\mu} Q \qquad P \text{ and } P' \text{ are } \alpha\text{-convertible}}{P \xrightarrow{\mu} Q}$$

$$\textsc{Inp}\quad x(\widetilde{y}).\,P \xrightarrow{x\langle\widetilde{z}\rangle} P\{^{\widetilde{z}}/_{\widetilde{y}}\},\ \text{if } \mid \widetilde{z} \mid = \mid \widetilde{y} \mid \qquad\qquad \textsc{Out}\quad \overline{x}\langle\widetilde{y}\rangle.\,P \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} P$$

$$\textsc{Sum}\quad \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \textsc{Par}\quad \frac{P \xrightarrow{\mu} P'}{P\ \ Q \xrightarrow{\mu} P'\ \ Q}\ \ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\textsc{Com}\quad \frac{P_1 \xrightarrow{(\nu\widetilde{y}')\overline{x}\langle\widetilde{y}\rangle} P' \qquad Q \xrightarrow{x\langle\widetilde{y}\rangle} Q' \qquad \widetilde{y}' \cap \mathrm{fn}(Q) = \emptyset}{P\ \ Q \xrightarrow{\tau} \nu\widetilde{y}'(P'\ \ Q')}$$

$$\textsc{Res}\quad \frac{P \xrightarrow{\mu} P' \qquad r \notin \mathrm{n}(\mu)}{\nu r\, P \xrightarrow{\mu} \nu r\, P'} \qquad \textsc{Const}\quad \frac{P\{^{\widetilde{y}}/_{\widetilde{x}}\} \xrightarrow{\mu} P'}{D\langle\widetilde{y}\rangle \xrightarrow{\mu} P'}\ \ \text{if }\ D \overset{\text{def}}{=} (\widetilde{x})P$$

$$\textsc{Open}\quad \frac{P \xrightarrow{(\nu\widetilde{y'})\overline{z}\langle\widetilde{y}\rangle} P'}{\nu x\, P \xrightarrow{(\nu x,\widetilde{y'})\overline{z}\langle\widetilde{y}\rangle} P'}\ \ x \neq z,\ x \in \mathrm{fn}(\widetilde{y}) - \widetilde{y'} \qquad \textsc{Match}\quad \frac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'}$$

Figure 2.3: The (early) labeled transition system for the π-calculus. Rules Act2 and Tau2, the symmetric counterparts of rules Act1 and Tau1, are omitted.

labeled transition semantics. This result is sometimes referred to as the *harmony lemma* (see, e.g., Sangiorgi and Walker (2001)).

2.2 Higher-Order Process Calculi

Higher-order process calculi are calculi in which processes (more generally, values containing processes) can be communicated. Thus a computation step involves the instantiation of a variable with a term, which is then copied as many times as there are occurrences of the variable. If there are multiple occurrences, the size of a system may grow. Higher-order process calculi have been put forward in the early 90s, with CHOCS (Thomsen, 1989) and Plain CHOCS (its variant with static binding) (Thomsen, 1990), and with the Higher-Order π-calculus (Sangiorgi, 1992). The basic operators are those of CCS (Milner, 1989).

The appearance of processes inside values usually has strong consequences on the semantics: namely on labeled transition systems (notions of alpha conversion, higher-order substitutions, scope extrusions) and, especially, on behavioral equivalences (e.g. bisimulation). Higher-order, or process-passing, concurrency is often presented as an alternative paradigm to the first order, or name-passing, concurrency of the π-calculus for the description of mobile systems, i.e. concurrent systems whose communication topology may change dynamically. Higher-order calculi are formally closer to the λ-calculus, whose basic computational step — β-reduction — involves term instantiation. As in the λ-calculus, a computational step in higher-order calculi results in the instantiation of a variable with a term, which is then copied as many times as there are occurrences of the variable, resulting in potentially larger terms.

The remainder of this section is structured as follows. In Section 2.2.1 we present the higher-order π-calculus; this is necessary to introduce Sangiorgi's representability result in Section 2.2.2. Then, in Section 2.2.3 we review several proposals of higher-order languages in the literature. Finally, in Section 2.2.4, we report on previous works on the behavioral theory for languages in the higher-order setting.

2.2.1 The Higher-Order π-calculus

Here we present the higher-order π-calculus, abbreviated HOπ. We introduce the language by building on the notations presented for the π-calculus in Section 2.1.4.

Let Var be a set of agent-variables, ranged over X, Y. In order to obtain HOπ, the syntax of the π-calculus (cf. Section 2.1.4) is modified in two ways. First, variable application is allowed, so that an abstraction received as input can be provided with appropriate arguments. Second, tuples in inputs, outputs, applications, and abstraction may also contain agent or agent-variables. To simplify the notation in the grammar below we use K to stand for an agent or a name and U to stand for a variable or a name.

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

Recall that K may be an agent: hence, it may be a process, but also an abstraction of arbitrary high order. The grammar of agents is the following:

$$A ::= (\widetilde{U})P \;\big|\; (\widetilde{U})X\langle \widetilde{K} \rangle \;\big|\; (\widetilde{U})D\langle \widetilde{K} \rangle$$

(Notice also that a variable X and a constant D are agents, corresponding to the cases in which $\widetilde{U}$ and $\widetilde{K}$ are empty. We make the assumptions regarding finiteness of tuples as in the π-calculus. A variable X which is not underneath some input prefix $x(\widetilde{U})$ or an abstraction $(\widetilde{U})$ with $X \in \widetilde{U}$ is said to be *free*. An agent containing free variables is said to be *open*. We use fv(A) to denote the set of free variables of agent A.

In HOπ the notions of types and type systems are more involved than in the π-calculus. For the sake of conciseness, we do not present such details here, and assume well-sorted expressions. The reader is referred to (Sangiorgi, 1992, 1996b) for details.

Now we present reduction and labeled transition semantics for HOπ. Let us introduce the reduction semantics first. The structural congruence rules and the reduction rules for HOπ are the same as for the π-calculus. We only have to generalize the structural congruence rule

$$\textsc{Alp}\quad \frac{P' \xrightarrow{\mu} Q \qquad P \text{ and } P' \text{ are } \alpha\text{-convertible}}{P \xrightarrow{\mu} Q}$$

$$\textsc{Inp}\quad x(\widetilde{U}).\,P \xrightarrow{x\langle\widetilde{K}\rangle} P\{^{\widetilde{K}}/_{\widetilde{U}}\},\ \text{if} \mid \widetilde{z} \mid = \mid \widetilde{y} \mid \qquad\qquad \textsc{Out}\quad \overline{x}\langle\widetilde{K}\rangle.\,P \xrightarrow{\overline{x}\langle\widetilde{K}\rangle} P$$

$$\textsc{Sum}\quad \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \textsc{Par}\quad \frac{P \xrightarrow{\mu} P'}{P \;\; Q \xrightarrow{\mu} P' \;\; Q}\ \ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\textsc{Com}\quad \frac{P_1 \xrightarrow{(\nu\widetilde{y})\overline{x}\langle\widetilde{K}\rangle} P' \qquad Q \xrightarrow{x\langle\widetilde{K}\rangle} Q' \qquad \widetilde{y}' \cap \mathrm{fn}(Q) = \emptyset}{P \;\; Q \xrightarrow{\tau} \nu\widetilde{y}(P' \;\; Q')}$$

$$\textsc{Res}\quad \frac{P \xrightarrow{\mu} P' \qquad r \notin \mathrm{n}(\mu)}{\nu r\, P \xrightarrow{\mu} \nu r\, P'} \qquad \textsc{Const}\quad \frac{P\{^{\widetilde{K}}/_{\widetilde{U}}\} \xrightarrow{\mu} P'}{D\langle\widetilde{K}\rangle \xrightarrow{\mu} P'}\ \ \text{if}\ \ D \overset{\text{def}}{=} (\widetilde{U})P$$

$$\textsc{Open}\quad \frac{P \xrightarrow{(\nu\widetilde{y})\overline{z}\langle\widetilde{K}\rangle} P'}{\nu x\, P \xrightarrow{(\nu x,\widetilde{y})\overline{z}\langle\widetilde{K}\rangle} P'}\ \ x \neq z,\, x \in \mathrm{fn}(\widetilde{K}) - \widetilde{y} \qquad \textsc{Match}\quad \frac{P \xrightarrow{\mu} P'}{[x = x]P \xrightarrow{\mu} P'}$$

Figure 2.4: The labeled transition system for HOπ. Rules Act2 and Tau2, the symmetric counterparts of rules Act1 and Tau1, are omitted.

(6) and the rule Com, so that the tuples involved may contain agents. This way, these rules become, respectively:

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

and

$$\textsc{Com}\quad (\cdots + x(\widetilde{U}).\,P) \;\; (\cdots + \overline{x}\langle\widetilde{K}\rangle) \longrightarrow P\{^{\widetilde{K}}/_{\widetilde{U}}\} \;\; Q\,.$$

The labeled transition semantics for HOπ is given in Figure 2.4. It arises as a generalization of that for the π-calculus given in Figure 2.3. While input actions take the form $x\langle\widetilde{K}\rangle$, output actions are of the form $(\nu\widetilde{y})\overline{x}\langle\widetilde{K}\rangle$; for the latter it holds that $\widetilde{y} \subseteq \mathrm{fn}(\widetilde{K}) - x$. The correspondence between reduction and labeled transition semantics mentioned for the π-calculus holds for HOπ as well.

2.2.2 Sangiorgi's Representability Result

We now introduce $\mathcal{C}$, the compilation of HOπ into the π-calculus, following the presentation in (Sangiorgi, 1993). Hence, for readability purposes, only the monadic calculus is considered. Also, it is assumed that agent definitions use a finite number of constants; this allows the use of replication in place of constants. Alternative presentations of the representability result —focused on asynchronous calculi— can be found in (Sangiorgi, 2001; Sangiorgi and Walker, 2001).

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

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

$$\mathcal{C}[X\langle F\rangle] \overset{\text{def}}{=} (\overline{x}\langle m\rangle.\,\mathbf{0})\{m := \mathcal{C}[F]\} \qquad \mathcal{C}[X\langle b\rangle] \overset{\text{def}}{=} \overline{x}\langle b\rangle.\,\mathbf{0}$$

$$\mathcal{C}[P\quad Q] \overset{\text{def}}{=} \mathcal{C}[P]\ \mathcal{C}[Q] \qquad \mathcal{C}[P+Q] \overset{\text{def}}{=} \mathcal{C}[P]+\mathcal{C}[Q]$$

$$\mathcal{C}[!P] \overset{\text{def}}{=} !\,\mathcal{C}[P] \qquad \mathcal{C}[\nu a\,P] \overset{\text{def}}{=} \nu a\,\mathcal{C}[P] \qquad \mathcal{C}[\,[a=b]P\,] \overset{\text{def}}{=} [a=b]\mathcal{C}[P]$$

$$\mathcal{C}[(X)P] \overset{\text{def}}{=} (x)\mathcal{C}[P] \qquad \mathcal{C}[(a)P] \overset{\text{def}}{=} (a)\mathcal{C}[P]$$

Figure 2.5: The compilation $\mathcal{C}$ from higher-order into first-order π-calculus

The translation uses notation $P\{m := F\}$ to stand for $\nu m(P \quad !m(U).\,F\langle U\rangle)$, where U is a name or a variable. The intuition is that $\mathcal{C}$ replaces the communication of an agent with the communication of the access to that agent. This way, $P_1 \overset{\text{def}}{=} \overline{a}\langle F\rangle.\,Q$ is replaced by $P_2 \overset{\text{def}}{=} (\overline{a}\langle m\rangle.\,Q)\{m := F\}$. While an agent interacting with P_1 may use F directly with, e.g., argument b, an agent interacting with P_2 uses m to *activate* F and provide it with b. The name m is called *name-trigger* or simply *trigger*.

The definition of $\mathcal{C}$ is presented in Figure 2.5. Notice that a variable X is translated into a name x. The correctness of $\mathcal{C}$ is studied in depth in Sangiorgi (1992). There, $\mathcal{C}$ is derived in two steps. The first is a mapping $\mathcal{T}$ which transforms an agent into a *triggered* agent. These are HOπ agents in which every agent emitted in an output or expected in an input has the same structure of a trigger. This gives homogeneity to higher-order communications and simplifies the reasoning over agents. The agent $\mathcal{T}[A]$ has the same structure as $\mathcal{C}[A]$ and maintains agents in the higher-order setting. A complementary mapping, denoted $\mathcal{F}$, transforms triggered agents into first-order processes. The compilation $\mathcal{C}$ is *fully-abstract* with respect to (weak) barbed congruence, i.e., for each pair of agents A_1 and A_2,

$$A_1 \cong^{c} A_2 \text{ if and only if } \mathcal{C}[A_1] \cong^{c} \mathcal{C}[A_2].$$

This result is then complemented by a statement of *operational correspondence* between P and $\mathcal{C}[P]$ which reveals the way the latter simulates the behavior of the former.

2.2.3 Other Higher-Order Languages

We review a number higher-order languages with concurrency, following the way they appeared in theoretical computer science:

- As a way of studying the foundations of programming languages integrating functional and concurrent paradigms. This is represented by variants of the λ-calculus enriched with forms of parallelism (see, e.g., (Boudol, 1989; Nielson, 1989)).
- As a way of studying forms of code mobility and mobile agents. This is represented by process calculi with process-passing (see, e.g., (Thomsen, 1990; Sangiorgi, 1992; Schmitt and Stefani, 2004; Hildebrandt et al., 2004)).

In what follows we give an overview of both research strands, with an emphasis on the efforts concerning process calculi.

2.2.3.1 Functional Languages with Concurrency

A number of works advocated that a formal model for concurrent, communicating processes should contain the λ-calculus as a simple sub-calculus.

Boudol (1989) proposes the γ-calculus, a strict extension of lambda calculus with CCS-like communication. The γ-calculus is a *direct generalization* of the λ-calculus, in that a β-reduction is formally defined to be a particular instance of the communication rule (and not something that is representable by a series of communications, for instance). The calculus is parametrized on a set of ports, and the parallel composition operator of CCS is splitted in two constructs: *interleaving* and *cooperation*, which represent concurrency and communication, respectively. The cooperation operator is not associative, so λ-calculus application is represented by cooperation and output constructs. This way the desired (tight) relationship between communication and application is achieved. Interestingly, in the γ-calculus the cooperation operator is "dynamic" in that reduction *behind* prefixes are allowed; much like as active outputs in certain modern process calculi.

In a similar spirit, Nielson (1989) proposes an extension of the *typed* λ-calculus with process communication. Here the rôle of types is indeed prominent, as they are meant to record the communication possibilities of processes, while retaining the usual information about functions and tuples that is provided by the typed λ-calculus. Here "communication possibilities" refers essentially to the channels over which communication can occur and the types of the entities that can be communicated over these. The type system then results from generalizing the notion of *sort* as defined in CCS; it guarantees that, given some expression e with type t, everything e evaluates to will also have type t, provided that e reads values of permissible types.

The FACILE language framework (Giacalone et al., 1989; Prasad et al., 1990) is an integration of functional and concurrent programming. Unlike other proposals, which enrich one of the programming styles with features of the other, FACILE intends to be *symmetric* in that a full functional language is integrated with a full concurrent language. In FACILE concurrent processes communicate through synchronous message passing; processes manipulate data in functional style. While the operational definition of FACILE first given in (Giacalone et al., 1989) consisted of a translation into a concurrent functional abstract machine, in (Prasad et al., 1990) the semantic foundations of the framework are given in terms of a notion of program behavior that combines the observable behavior of processes and the evaluation of expressions. Program equivalence is based on a form of contexts called *windows*, which are meant to index families of bisimulation relations. Roughly speaking, two expressions are equivalent if they reduce to equivalent values while producing equivalent behavior. The higher-order nature of the language is reflected in the observational equivalence by developing the notion of equivalent actions to be based on the idea of equivalent values. This is like higher-order bisimilarity defined in CHOCS.

Finally, CML is a concurrent extension of Standard ML (Reppy, 1991, 1992) in which synchronous operations are treated as first-class values. Synchronous operations are represented in the set of values by *events*; the language provides combinators to construct complex events from event values. This way a wide range of synchronization abstractions can be constructed, which in turn allows to support different concurrency idioms.

2.2.3.2 Process Calculi with Higher-Order Features

Here we review a number of concurrent languages which rely on a process calculi basis in order to implement higher-order features.

CHOCS and Plain CHOCS. Thomsen (1995, 1990) introduces and develops the basic theory of CHOCS, an extension of CCS with process passing. Probably the most distinctive feature of CHOCS is the treatment of the restriction operator as a dynamic binder. This simplifies significantly several aspects of the theory, such as the definition of an algebraic theory and denotational semantics. Similarly as in (Boudol, 1989), the behavioral equivalence defined for CHOCS (higher-order bisimilarity) considers the bisimilarity of the values of the actions, rather than their equality. Higher-order bisimilarity is shown to be a congruence, and an algebraic theory for it is also developed. Main ideas of the type system developed by Nielson (1989) are adapted to CHOCS so as to define a notion of sorts. As for the observational equivalence, internal actions are abstracted in the delayed style: an arbitrary number of internal actions are only allowed before a visible one. Using this definition, and similarly as in CCS, the observational equivalence is proven to be a congruence for sum-free processes.

Finally, an extension of the Hennessy-Milner logic is proposed for characterizing higher-order bisimilarity. The higher-order nature of the calculus is captured by enriching modalities with (i) formulas representing the processes sent or received, and (ii) the state after the transition. The expressiveness of the calculus is demonstrated by exhibiting encodings of the lambda calculus (with several evaluation strategies) into CHOCS, as well as an interpretation of a simple imperative programming language into CHOCS.

Inspired in the π-calculus, Plain CHOCS (Thomsen, 1993, 1990) results from considering the restriction operator in CHOCS as a static binder. The transition system for Plain CHOCS and the bisimilarity developed upon it follow closely those defined for the lazy λ-calculus (Abramsky, 1989). The relationship between Plain CHOCS and the π-calculus is explored by means of encodings in the two directions. The encoding of Plain CHOCS into the π-calculus follows the strategy in Milner et al. (1992): the communication of a process in Plain CHOCS is represented in the π-calculus as the communication of a link to a trigger construct that provides copies of the communicated process. This encoding is defined for the fragment of Plain CHOCS without renaming. An alternative encoding that considers renaming by admitting a set of names as parameter is also proposed. The encoding of the π-calculus into Plain CHOCS is more involved: the communication of a name in the π-calculus is represented by the communication of a Plain CHOCS process that contains the (input, output) capabilities of a name. This process, so-called *wire*, is meant to be "plugged" into the context of a given receiver by renaming operations that allow to "localize" the capabilities of the name. The encoding is then formalized as a two-level translation. In the first step, all free names and input-bound names are translated into process variables. Names bound by restriction are simply translated as names in Plain CHOCS. In the second step of the translation, the process variables corresponding to free names in the π-calculus process are translated into names in Plain CHOCS.

Bloom (1994) proposes a meta-theory for higher-order process calculi that generalizes CHOCS by considering constructs for broadcasting communication and interruption of process execution. Computation is of two kinds: process algebraic and functional so the meta-theory subsumes the name passing capabilities of the π-calculus and reduction as in the λ-calculus. The metatheory is typed; types of channels recognize input and output capabilities. The behavioral equivalence investigated is a generalization of higher-order bisimulation as proposed by Boudol (1989) and Thomsen (1990), and is shown to be a congruence.

The Blue Calculus. The Blue calculus (Boudol, 1998) provides an integration of the λ-calculus with the π-calculus with the objective of obtaining a direct model for higher-order concurrency with the same expressive power of the π-calculus while offering a more convenient programming notation. The Blue calculus is endowed with a type system that encompasses both Curry's type inference for the λ-calculus and Milner's sorting type system for the π-calculus. In a nutshell, the computational model of Blue is built around of a *name-passing* λ-calculus in which asynchronous messages might call for resources (or services) available in the form of linear and "inexhaustible" declarations. Because of the unified rationale of Blue, programs in the λ-calculus and processes in the π-calculus have both a Blue interpretation; this way, the Blue calculus is useful to formalize a number of intuitions on the relationship between the λ-calculus and the π-calculus as well as encodings of evaluation strategies of the former into the latter.

The M-calculus. Schmitt and Stefani (2003, 2002) propose the M-calculus, a higher-order distributed calculus that provides the notion of *hierarchical, programmable locality* as a way of representing those distributed systems in which localities can be of different kinds and exhibit different kinds of behaviors (e.g. with respect to access control or to failures). This is in sharp contrast with other calculi for distributed programming (such as the Ambient Calculus (Cardelli and Gordon, 2000)) in which localities are homogeneous, i.e. they are all of the same kind and have the same pre-defined behavior. This kind of distributed localities with explicit, programmable behavior are called *cells*; in combination with higher-order communication and dynamic binding features they allow to give a unified view of process migration and communication. The design of the M-calculus retains features from the Blue Calculus (Boudol, 1998) and the Join calculus (Fournet et al., 1996); the M-calculus features a functional character in messages (as in Blue), message patterns within definitions, and named cells so as to form a tree-like hierarchy (as in Join). As a novelty, the M-calculus introduces a *passivation* operator, which can "freeze" running processes, and a type system for guaranteeing the unicity of names of active cells are also introduced.

The Kell calculus. The Kell calculus (Schmitt and Stefani, 2004) arises as a generalization of the M-calculus, defined as a *family* of calculi intended to serve as a basis for component-based distributed programming. Built around a π-calculus core, the main features of the Kell calculus are hierarchical, programmable localities and local actions. While the former are inherited from the M-calculus and allow to express different semantics for containment and movement, the latter embody a principle under which atomic actions should occur within a locality, or at the boundary between a locality and its enclosing environment. Also as in the M-calculus, in Kell the execution of a process within a locality can be controlled through its *passivation*. The Kell calculus can be instantiated by means of *input pattern languages*, i.e. the language allowed in input constructs; this is most useful in defining a generic behavioral theory for the calculus. As a matter of fact, under sufficient conditions on substitution properties of such pattern languages, a co-inductive characterization of contextual equivalence is provided in terms of a form of higher-order bisimulation termed strong context bisimulation.

Homer. Homer (Hildebrandt et al., 2004) is a higher-order calculus for mobile embedded resources. Its main features are active code mobility, explicit nested locations, and local names. Given a resource (i.e. a process) inside a location, active process mobility refers to the fact that the resource might be *taken* (or *pulled*) by a suitable complementary prefix. This kind of movement —sometimes referred to as *objective mobility*— is a feature Homer shares with the M-calculus and Kell. Crucially, and similarly as Boudol's γ-calculus reviewed above, the resource has the capability of performing internal computations inside locations, that is, the resource can evolve on its own before being moved. This is in sharp contrast to usual process passing and substitution. Location addresses are defined by nested names; interactions between resources at arbitrarily nested resources are allowed. These nested locations come with an involved treatment of local names, scope extension and extrusion. In Homer, barbed congruence is shown to be characterized by a labeled transition bisimulation congruence.

HOPLA. HOPLA (Nygaard and Winskel, 2002) is a higher-order language for non-deterministic processes that arises from a proposal for domain theory for concurrency; that is, from a denotational/categorical approach for giving meaning to concurrent computation. Roughly speaking, HOPLA is an extension of the typed λ-calculus in which a process is typed with a collection of its possible *computation paths*. The notion of *prefix-sum type* is introduced for this purpose. The denotation of a process then relies on set-based operations on paths and their extensions. HOPLA has a developed operational semantics and behavioral theory; sensible equivalences have been defined for it, including ordinary bisimilarity, applicative bisimilarity, and higher-order bisimilarity. An advantage of the domain-theoretical approach for concurrency is that it is general by definition, and naturally leads to metalanguages for process description. This is evidenced in HOPLA, which can encode directly languages such as CCS, CCS with process passing, and mobile ambients. An extension of HOPLA that incorporates name generation has been introduced in (Winskel and Zappa Nardelli, 2004).

KLAIM. KLAIM (De Nicola et al., 1998) is a *process description language*: as such, it falls between a programming language and a process calculi (see (De Nicola, 2006) for a short survey on this distinction). In its process calculus dimension, processes and data can be moved from one computing environment to another. KLAIM builds on the Linda tuple space model, and can be seen as an asynchronous higher-order process calculus whose basic actions are the original Linda primitives enriched with explicit information about the location of the nodes where processes and tuples are allocated. The behavioral theory for sub-languages of KLAIM focuses on barbed congruence and may testing; it has been studied by Boreale et al. (1999).

Other proposals. In addition to the above mentioned calculi, other proposals of calculi for higher-order concurrency can be found in the literature. For the sake of conciseness, we only mention them without expanding in their details. Radestock and Eisenbach (1996) put forward a higher-order process calculus for coordination in environments with distributed components. Ostrovsky et al. (2002) propose a higher-order process calculus with broadcasting communication, and study its semantic theory in depth. Meredith and Radestock (2005b,a) propose a *reflective* higher-order process calculus in which names as in the π-calculus are obtaining by *quoting* processes. Hennessy et al. (2005) have proposed sophisticated type systems with dependent and existential types for a distributed version of the π-calculus with higher-order communication of parametrized code. Mostrous and Yoshida (2007, 2009) have studied calculi for structured communication with higher-order communication and type disciplines for them. Higher-order process calculi oriented towards security issues have been put forward by Maffeis et al. (2008), who propose a higher-order spi-calculus (Abadi and Gordon, 1999) for code-carrying authorization, and by Sato and Sumii (2009), who define a higher-order calculus with cryptographic-like operations over terms such as decomposition. They rely in environmental bisimilarities —to be reviewed later on— for developing the behavioral theory of their calculus.

2.2.4 Behavioral Theory

Here we review some works that have addressed the behavioral theory for higher-order languages. We concentrate on works for higher-order process calculi.

Sangiorgi (1994) studies equivalences for versions of the λ-calculus possibly involving parallelism. The objective is to find the finest behavioral equivalence on terms (i.e. the one that discriminates the most). The starting point is Abramsky's applicative bisimilarity for the lazy λ-calculus (Abramsky, 1989). Two approaches are followed. In the first one, the equivalence induced by the encoding of the lazy λ-calculus into the π-calculus (so-called lambda observational equivalence) is studied. Such an equivalence is shown to be a congruence (using a direct proof, i.e. without using the encoding into pi), and fully-abstract with respect to Levy-Longo trees, the tree-like model for lazy λ-calculus terms. The second approach considers extensions of the pure λ-calculus. A rule format for *well-formed operators* is proposed for that purpose; intuitively, the rule format generates operators whose behavior only depends on their semantics, and not on their syntax. The most discriminating congruence is obtained when all well-formed operators are admitted; such a congruence (so called rich applicative congruence) is shown to coincide with lambda observational equivalence. Non determinism is shown to be the essential component for obtaining maximal discrimination.

The definition of a satisfactory notion of bisimilarity is a hard problem for a higher-order process language. In ordinary bisimilarity, as e.g. in CCS, two processes are bisimilar if any action by one of them can be matched by an equal action from the other in such a way that the resulting derivatives are again bisimilar. The two matching actions must be syntactically *identical*. This condition is unacceptable in higher-order concurrency; for instance it breaks vital algebraic laws such as the commutativity of parallel composition. The approach taken by Thomsen (1990), following earlier ideas by Astesiano et al. (1988) and Boudol (1989), is to require *bisimilarity* rather than *identity* of the processes emitted in a higher-order output action. This weakening is natural for higher-order calculi and the bisimulation checks involved are simple. Sangiorgi (1992) then argued that this form of bisimulation, called *higher-order bisimilarity*, is in general troublesome or over-discriminating as a behavioral equivalence, and basic properties, such as congruence, may be very hard to establish. He then proposed *context bisimilarity* (1996a), a form of bisimilarity that avoids the separation between object part and continuation of an output action by explicitly taking into account the context in which the emitted agent is supposed to go. Context bisimilarity yields more satisfactory process equalities, and coincides with contextual equivalence (i.e., barbed congruence). However, it has the drawback of a universal quantifications over contexts, which can make it hard, in practice, to check equivalences.

Normal bisimilarity (Sangiorgi, 1992; Jeffrey and Rathke, 2005; Cao, 2006) is a simplification of context bisimilarity without universal quantifications in the output clause. The input clause is simpler too: normal bisimilarity can indeed be viewed as a form of *open bisimilarity* (Sangiorgi, 1996c), where the formal parameter of an input is not substituted in the input clause, and free variables of terms are observable during the bisimulation game. However, the definition of the bisimilarity may depend on the operators in the calculus, and the correspondence with context bisimilarity may be hard to prove. The characterization of context bisimilarity using normal bisimilarity in (Sangiorgi, 1996a) exploits *triggered bisimilarity*, an intermediate characterization of bisimilarity defined over *triggered processes*, i.e. a set of processes in which every communication takes place by the exchange of a trigger. Sangiorgi (1996a) obtains this characterization for the *weak* case; however, the proof technique based on going through triggered agents does not carry over to the strong case, as it adds extra internal actions. Recently, Cao (2006) showed that strong/weak context bisimulation and strong/weak normal bisimulation coincide in higher-order π-calculus. To do so, he goes through *indexed bisimilarity*, an equivalence defined over a variant of the calculus in which every prefix is indexed. Cao then uses indices to distinguish "internal" tau actions (those originating inside a component) from those "external" ones (those taking place among different components). The first —which are essentially the kind of actions added by the encoding into triggered processes— are neglected in the indexed versions of the bisimulation games. Apart from settling the issue of the coincidence between normal and context bisimilarities in the strong case, the work in (Cao, 2006) provides a uniform setting for proving the coincidence of bisimilarities:

in the index-based proof technique the coincidence for the weak case results as a particular instance.

A drawback of the characterization of context bisimilarity with normal bisimilarity in (Sangiorgi, 1996a) is that it is restricted to languages with finite types. Jeffrey and Rathke (2005) extends such a characterization to a language with recursive types. Their approach is based on an enriched labeled transition system in which special operators representing references to triggers are included in the labels. As a result, a direct proof of soundness is possible, i.e., bisimilarity based on this enriched labeled transition system implies context bisimilarity. Completeness also holds; for the proof the original approach based on triggers is necessary.

In addition to the definition of a suitable notion of bisimilarity, a related hard problem is the proof that the bisimilarity is a congruence. In fact, for higher-order languages the "term-copying" feature inherited from the λ-calculus can make it hard to prove that bisimilarity is a congruence. A classical method for proving congruence of higher-order bisimulations is that of Howe (1996). Originally introduced for (lazy) functional programming languages, this method was first adapted to higher-order process calculi by Baldamus (1998) and Thomsen (1989, 1993) who used it for (variants of) CHOCS and Plain CHOCS. More recently, it has been used by Ferreira et al. (1998) –for a concurrent version of ML– and by Hildebrandt et al. (2004; 2005), to show that late and input early bisimilarities are congruences in untyped and typed versions of Homer.

Recently, as a means of alleviating some of the problems Howe's method entails when used for concurrent languages (most notably, its lack of flexibility), Sangiorgi et al. (2007) proposed *environmental bisimulations*, a method for higher-order languages that aims at make proofs of congruence easier and compatible with the so-called up-to techniques (Sangiorgi, 1998). Roughly, an environmental bisimulation makes a clear distinction between the terms tested in the bisimulation clauses and the environment, that is, an observer's current knowledge. As such, for instance, in the output clause of the environmental bisimulation for HOπ, the emitted processes become part of the environment; the extruded names also receive special treatment inside the clause. This is a more robust technique than previous approaches; it has been applied to both functional languages (pure λ-calculus and λ-calculus with information hiding) and to concurrent ones (Higher-Order π-calculus).

Two very recent works develop further the theory of environmental bisimulations. Sato and Sumii (2009) adapt and extend it in the setting of a higher-order, applied π-calculus featuring cryptographic operations such as encryption and decryption. Koutavas and Hennessy (2009) propose a first-order behavioral theory for higher-order processes based on the combination of the principles of environmental bisimulations and the improvements to normal bisimilarity proposed by Jeffrey and Rathke (2005). At the heart of the proposed theory is a novel treatment of name extrusions, which is formalized as an LTS in which configurations not only contain the current knowledge of the environment and a process, but also information on the names extruded by the process. As a consequence, the labels of such an LTS have a very simple structure. The weak bisimilarity derived from this LTS is shown to be a congruence, fully abstract with respect to contextual equivalence, and to have a logic characterization using a very simple Hennessy-Milner logic.

Lenglet et al. (2009b, 2008, 2009a) have studied the behavioral theory of variants of higher-order π-calculi with restriction and/or passivation constructs. In (Lenglet et al., 2009b, 2008) they show that in a higher-order calculus with a passivation operator (such as Kell and Homer), the presence of a restriction operator disallows the characterization of barbed congruence by means of strong and normal bisimilarities. They use Howe's method to prove congruence of a weak higher-order bisimilarity for a calculus with passivation but without restriction. This result is improved in (Lenglet et al., 2009a) where barbed congruence is characterized for a higher-order process calculus with both passivation and restriction. To that end, they exploit Howe's method with the aid of so-called *complementary semantics*, which coincide with contextual semantics and allow the use of Howe's method to prove soundness of weak bisimilarities.

The congruence of bisimilarity can also be approached by means of (syntactic) rule formats (see, e.g., (Mousavi et al., 2007), for a survey on formats and metatheory of structural operational semantics). These are formats that induce congruence for any given notion of bisimilarity once the rules of the operational semantics adhere to the formats. Bernstein (1998) proposes a rule format (promoted tyft/tyxy) for languages with higher-order features. The paper shows that for any language defined in the format, strong bisimulation is a congruence. The approach is applied to the lazy λ-calculus, the π-calculus, and CHOCS. In all cases, the studied equivalence is bisimilarity; other behavioral equivalences, such as applicative bisimulation or higher-order bisimulation, are not considered. Also, the format imposes a number of restrictions on labels. In (Mousavi et al., 2005) both these shortcomings are studied. They build on Bernstein's work and propose a more general and relaxed rule format which induces congruence of (strong) higher-order bisimilarity. They use CHOCS to illustrate their rule format. The definition of suitable rule formats for other, more sensible, notions of bisimilarity (say, normal and context bisimilarity) is left in (Mousavi et al., 2005) as an open question.

2.3 Expressiveness of Concurrent Languages

In this section we give a broad overview of the main approaches to the *expressiveness* of concurrent languages. We focus on the issues and techniques we shall use in this dissertation; the reader is referred to, e.g., (Parrow, 2008), for a recent survey on the area.

We discuss on general issues in expressiveness in Section 2.3.1. Then, in Section 2.3.2, we briefly review some of the notions of encoding that have been proposed in the literature. A classification of the main kinds of expressiveness results and the approaches to obtain them is presented in Section 2.3.3. Finally, we report on previous efforts on the expressiveness of higher-order concurrent languages (Section 2.3.4).

Along the section, we shall follow a few notational conventions. We use $\mathcal{L}_1, \mathcal{L}_2, \ldots$ to range over languages; we use $\approx$ (possibly decorated) to denote a suitable behavioral equivalence. Also, $\longrightarrow$ and $\Longrightarrow$ denote some (reduction) semantics and its reflexive, transitive closure, respectively.

2.3.1 Generalities

An important criterion for assessing the significance of a paradigm is its *expressiveness*. While in other areas of computer science (most notably, automata theory), the notion of expressiveness is well-understood and settled, in concurrency theory there is yet no agreement on a formal characterization of the *expressive power* of a language, possibly with respect to that of some other language or model. While such a unified theory would be certainly desirable, the wide variety of existing models for concurrency (and consequently, of the expressiveness issues inherent to them) strongly suggests that a single theory for language comparison embracing them all does not exist.

The crux of expressiveness studies is the notion of *encoding*, i.e., a function (or map) $[\![\cdot]\!]$ from the terms of a *source language* into the terms of a *target language* that satisfies certain *correctness criteria*. These criteria enforce both *syntactic* and *semantic* conditions on the nature of $[\![\cdot]\!]$. It is not difficult to see that the main source of difficulty in defining a unified theory for language comparison lies precisely in the exact definition of these criteria: depending on the purpose and on the given language(s), the set of applicable criteria might vary and/or there might be criteria more adequate than others.

From the point of view of their *purpose*, expressiveness studies can be broadly seen to be aimed at two kinds of results: *encodability* and *non-encodability* (or *impossibility*) results. As their name suggests, the former are concerned with the *existence* of an encoding, whereas the latter address the opposite issue. These two kinds of questions are intimately related as, given two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, in order to assert that $\mathcal{L}_1$ is *more expressive* than $\mathcal{L}_2$, one needs to provide instances of both kinds of results: one should exhibit an encoding $[\![\cdot]\!] : \mathcal{L}_2 \to \mathcal{L}_1$ and, at the same time, one should provide a formal argument ensuring that an encoding $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ does not exist. That is, it should be made clear that while $\mathcal{L}_1$ is able to express all the behaviors of $\mathcal{L}_2$, there are some behaviors in $\mathcal{L}_1$ that $\mathcal{L}_2$ is unable to represent. It might then appear clear that the correctness criteria for an encodability result should be different from those for an impossibility result. Indeed, for encodability results one would like to exhibit the best encoding possible, i.e., one satisfying the most demanding correctness criteria possible; in contrast, for impossibility results one would like to rely on the most general formal argument, i.e. one satisfying the least demanding correctness criteria possible. Not surprisingly, the proof techniques involved and the ingenuity required to obtain either result can be quite different.

Another broad classification of expressiveness studies takes into account whether or not the expressive power of a given language is analyzed with respect to another language. In other words, whether one is interested in *absolute* or in *relative* expressiveness.

In studies of absolute expressiveness the interest is therefore in assessing the expressive power that is intrinsic to the language and its associated semantics: as Parrow (2008) explains, this question entails determining exactly the transition systems —as well as the operators on them— that are expressible in a given language. That is, the focus is on the expressiveness of the terms of the language, and on the kind of operators that are expressible in it. These questions depend on suitable denotations of labeled transition systems, which explains the fact that expressiveness results of this kind have been reported only for *basic* process calculi, with relatively simple labels (Parrow, 2008). A pioneering work in this direction is De Simone's study of the expressive power of the MEIJE process algebra (de Simone, 1985). A seemingly widespread approach to absolute expressiveness relies on some standard model of computation —rather than on the semantic machinery of the language— to assess the expressive power of a language. A common yardstick here is Turing completeness, which is generally shown by exhibiting an encoding of some Turing-equivalent model into the given language. While this approach to absolute expressiveness —sometimes referred to as *computational expressiveness* (see, e.g., Aranda (2009); Busi and Zandron (2009))— takes some external model as reference (and as such, it is not entirely "absolute"), the fact that such reference models are widely known and/or understood often constitutes a satisfactory measure of the intrinsic expressive power of a language.

In relative expressiveness one measures the expressive power of a given language $\mathcal{L}_1$ by taking some other language $\mathcal{L}_2$ as a reference. This is particularly appealing when, for instance, one wants to show that $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same expressive power. In this case, the objective is to obtain two encodability results, one in each direction. Another common situation is when one wishes to determine the influence a particular operator or construct has on the expressiveness of a language $\mathcal{L}_1$. In this case, the reference language $\mathcal{L}_2$ is the fragment of $\mathcal{L}_1$ without the operator(s) of interest. In this case, one aims at showing that $\mathcal{L}_1$ cannot be encoded into $\mathcal{L}_2$. If this can be done then the difference in the expressive power between the two languages has been singled out: it is in the operators that $\mathcal{L}_1$ has but that $\mathcal{L}_2$ lacks. This is sometimes referred to as a *separation result*, as the analyzed construct *separates* the world with it from the world without it (Yoshida, 2002).

2.3.2 The Notion of Encoding

We present a historical account of the evolution of definition of *encoding*, starting from proposals within programming languages at large and concluding with the most relevant proposals for concurrent languages.

2.3.2.1 Early Attempts to Expressiveness

It is instructive to examine the origin of the notions of expressiveness and expressive power in the realm of programming languages at large. The earliest attempts towards a formal notion of "expressive power" can be traced back to the late 1960s, when a proliferation of programming languages was first noticed. Perhaps the most influential work of that period is due to Landin (1966), who proposed a unified framework aimed at describing *families* of programming languages from which particular languages can be derived by an appropriate choice of primitives. Main concerns in Landin's formal framework are conventions about user-defined names and functional relationships.

Later on, in the early 1970s, the question of the expressive power was studied by representing families of programs by means of *program schemas*, i.e., abstract representations of programming features with uninterpreted constant and function symbols (see, e.g., (Chandra and Manna, 1976)). This line of research —sometimes referred to as *comparative schematology*— is mainly concerned about the expressiveness of single constructs.

Felleisen (1991) developed a framework for expressiveness studies in the context of functional languages. His framework is suited for comparing a language and some *extension* of it; hence, it is suited for studies of relative expressiveness as introduced before. The framework departs from the idea of *eliminable* syntactic symbols as proposed in logic by Kleene and others. More concretely, given two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 \subseteq \mathcal{L}_2$, if the additional symbols/constructs of $\mathcal{L}_2$ are eliminable (with respect to $\mathcal{L}_1$) then $\mathcal{L}_2$ is said to be a *definitional extension* of $\mathcal{L}_1$. Several notions and concepts that we shall encounter in "modern" studies of expressiveness of concurrent languages can be found already in Felleisen's work. For instance, the crucial observation that the key to (programming) language comparison is a restriction on the set of admissible translations between (programming) languages. This observation is represented by structural (syntactic) and semantic conditions; while the former include notions such as compositionality of translations and homomorphism of a translation with respect to some operator, the latter is represented by the preservation of terminating behavior, a natural requirement in a functional setting. In Felleisen's view, the expressiveness of a programming language is closely related to the programming discipline since, intuitively, programs written in the extension of some core language can be more readable than the programs written in the core language.

Mitchell (1993) compares (functional) languages according to the ability of making sections of a program "abstract" by hiding some details of the internal functioning of the code. He defines so-called *abstraction-preserving reductions*, which are compositional translations that preserve observational equivalence. Perhaps the simplest reduction of this kind is the one translating program blocks into function declaration and calls. Proofs showing that more involved reductions are abstraction-preserving might involve appealing to the operational and denotational semantics of the languages in question. Riecke (1993) uses and extends the notion of abstraction-preserving reductions in the study of the expressive power of different evaluation strategies in the functional language PCF. He shows that call-by-value and lazy PCF are equally expressive, and that both are more expressive than call-by-name PCF.

2.3.2.2 Encodings Among Concurrent Languages: The Early Days

Shapiro (1989) was the first to study expressiveness issues for concurrent languages. He proposed the notion of *embedding* as a way of comparing concurrent logic programming languages; considered languages are thus relatively similar and it is easy to focus on their differences. An *embedding* is composed of a *compiler* and a *viewer* (or *decoder*). Given two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, the compiler is a function c from programs of $\mathcal{L}_1$ into programs of $\mathcal{L}_2$, whereas the viewer is a function v from observables of $\mathcal{L}_2$ into observables of $\mathcal{L}_1$. Both c and v form an embedding of $\mathcal{L}_1$ into $\mathcal{L}_2$ if the observables of every program P in $\mathcal{L}_1$ correspond to the observables of the program obtained by compiling P using c and viewing (or decoding) its behavior using v. In order to define a hierarchy of concurrent logic programming languages, this notion of embedding is tailored to the logic programming setting by requiring *natural embeddings*, i.e., embeddings in which (a) the unification mechanism of one language is implemented in the unification mechanism of the other, and (b) logical variables of one language are mapped into logical variables of the other. This proposal for language comparison was refined by Shapiro (1991) and by de Boer and Palamidessi (1990, 1994). We comment on both refinements next.

Shapiro (1991) claims that no method similar to program schemas exists for comparison of concurrent languages. He then proposes a *general framework* for language comparison, which relies on the (non) existence of *mappings* that preserve the syntactic and semantic structure of the languages. Those mappings adhering to such preservation conditions are called *embeddings*. The framework is expressed in categorical terms, and is general enough so as to work for any family of languages with syntactic operations and a semantic equivalence. Shapiro identifies three categories of embeddings that provide an incremental notion on the preservation of the semantic structure of languages: *sound embeddings*, i.e. mappings that preserve observable distinctions; *faithful embeddings*, i.e. sound embeddings that preserve the semantic equivalence; *fully-abstract embeddings*, i.e. embeddings that are faithful with respect to the congruence induced by the semantic equivalence. The work concentrates in the formalization of separation results; a so-called *separation schema* arises from considering parallel composition as the sole composition operation and by considering three properties: *compositionality*, i.e. the coincidence of the semantic equivalence with its induced congruence; *interference-freedom*, which disallows the parallel composition of a program with itself; *hiding*, i.e. the existence of programs that are semantically different from the trivial program, but whose composition is semantically equivalent to the trivial program. The framework for language comparison is used to provide a number of separation results among several concurrent languages and models, including Input/Output Automata, Actors, concurrent Prolog, and (variants of) CCS and CSP. In (Shapiro, 1992), the general framework is also shown to be useful for formalizing positive (i.e. encodability) results.

After observing that the notion of embedding introduced by Shapiro fell short for formalizing certain separation results among concurrent constraint languages, de Boer and Palamidessi (1994) introduced the refined notion of *modular embedding*. A modular embedding is an embedding that satisfies the following three restrictions. First, since in the presence of non-determinism the domain of the observables of a language is a powerset, the decoder of the embedding is required to be defined elementwise on the elements on the set of observables. Second, the compiler is required to be *compositional* with respect to the parallel composition and the non-deterministic choice operators. Third, the embedding must be *termination invariant*: a success (resp. deadlock or failure) in the target language must correspond to a success (resp. deadlock or failure) in the source language. The notion of modular encoding is then used to derive separation results in the context of concurrent constraint languages with different communication primitives in guarded-choice operators. The key idea to achieve separation relies on a semantic argument: two variants are separated by showing that a certain closure property is satisfied by the semantics of one variant but not by the semantics of the other. The notion of modular embedding was also used in (de Boer and Palamidessi, 1991) to show separation results for variants of CSP with different communication primitives in the guards. Indeed, it is shown that asynchronous CSP is strictly less expressive than CSP, thus confirming results obtained by Bougé (1988), who exploited the capability each variant have of expressing symmetric solutions to the leader election problem.

2.3.2.3 Encodings Among Concurrent Languages: Towards "Modern" Criteria

The introduction of the π-calculus in the early 1990s gave a significant momentum to the study of expressiveness issues in process calculi. Indeed, the simplicity and flexibility of name-passing as embodied in the π-calculus triggered many works proposing variants or extensions of it. Such works addressed a wide variety of concerns, including, e.g., polyadic communication (Milner, 1991), asynchronous communication (Boudol, 1992; Honda and Tokoro, 1991), higher-order communication (Thomsen, 1990; Sangiorgi, 1992), stochastic behavior (Priami, 1995), structured communication (Honda et al., 1998), security protocols (Abadi and Gordon, 1999; Abadi and Fournet, 2001). While some of these variants were mainly only of theoretical interest, some others (e.g., (Priami, 1995; Abadi and Gordon, 1999)) were aimed at exploiting working analogies between the behavior of mobile systems as in the π-calculus and that of systems in areas such as systems biology and security.

In this context, expressiveness studies for the π-calculus were then indispensable to understand its fundamental properties, to identify the intrinsic sources of its expressive power, and to discern about the relationships between its many variants. As representative examples of works in these directions, we find studies on the properties of the translation of polyadic into monadic π-calculus (Yoshida, 1996; Quaglia and Walker, 2005), on the relationship between point-to-point and broadcasting communication (Ene and Muntean, 1999), on the different kinds of choice operators (Nestmann, 2000; Nestmann and Pierce, 2000) and, closely related, on mechanisms for synchronous and asynchronous communication (Palamidessi, 2003; Cacciagrano et al., 2007). Probably as a consequence of the different motivations for approaching expressiveness, each of these works advocated its own definition of encoding, one in which the set of correctness criteria is defined in accordance to some specific working intuition or necessity. In what follows we review some of those proposals and comment on their main features. For the sake of conciseness, we focus on a few, representative proposals —namely those by Sangiorgi (1992), Nestmann (1996), Palamidessi (2003), and Gorla (2008)— in order to give a broad overview to the area and to contrast certain aspects that we judge relevant.

As part of his study on the relationship between first-order and higher-order π-calculus, Sangiorgi (1992) identifies three phases in determining that a given source language can be *representable* into some target language:

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 correctness of the encoding with respect to the semantics given.

Concerning the properties of (2), the only requirement is *compositionality*, that is, that the definition of the encoding of a term should only depend on the definition of its immediate constituents. Given source and target languages $\mathcal{L}_s$ and $\mathcal{L}_t$, an encoding $[\cdot] : \mathcal{L}_s \to \mathcal{L}_t$, and an n-adic construct op of $\mathcal{L}_s$, compositionality can be expressed as follows:

$$[\mathrm{op}(P_1, \ldots, P_n)] = C^{\mathrm{op}}[[P_1], \ldots, [P_n]] \tag{2.2}$$

where C^{op} is a valid process context in $\mathcal{L}_t$. As for correctness criteria, the main criteria adopted is *full-abstraction*, i.e., two terms in the source language should be equivalent if and only if their translations are equivalent:

$$S_1 \approx_s S_2 \text{ if and only } [S_1] \approx_t [S_2]. \tag{2.3}$$

That is, full-abstraction enforces both *preservation* and *reflection* of the equivalence of source terms. Sangiorgi admits that full-abstraction represents a strong approach to representability. As we shall elaborate later, the purpose of Sangiorgi is to *transfer* reasoning techniques from the first-order setting to the higher-order one. In this sense, requiring full abstraction turns out to be necessary, given that target terms should be usable in any context, and the indistinguishability of two source terms should imply that of their translations in order to switch from one language to another. He also acknowledges that full-abstraction alone is not informative enough with respect to the relationship between source and target terms. To that end, he argues that full-abstraction should be complemented with some form of *operational correspondence* relating a term and its translation.

Based on his works on the encodability of choice operators into the (choice-free) π-calculus, Nestmann (1996) collects a number of desirable correctness criteria for encodings. As for full-abstraction, Nestmann comments that it might not applicable in those cases in which the source language is not equipped with a notion of equivalence. Then, a suitable notion of operational correspondence gains relevance. Operational correspondence is usually expressed as two complementary criteria. The first one, *completeness*, ensures the preservation of execution steps, i.e., that the translation is able to simulate all the computations of the source term:

$$S_1 \longrightarrow_s S_2 \text{ implies } [\![S_1]\!] \Longrightarrow_t [\![S_2]\!] . \tag{2.4}$$

The second criteria, *soundness*, ensures the reflection of execution steps, i.e., that the behavior of a term in the target language can be related to the behavior of its corresponding term in the source language:

$$[\![S_1]\!] \Longrightarrow_t [\![S_2]\!] \text{ implies } S_1 \Longrightarrow_s S_2 . \tag{2.5}$$

However, soundness as in (2.5) is not satisfactory as it disregards the intermediate processes the translation of a source term might need to go through in order to simulate its behavior. A refinement that considers such intermediate steps is the following:

$$\text{if } [\![S]\!] \longrightarrow_t [\![T]\!] \text{ then there is } S \longrightarrow_s S' \text{ such that } [\![T]\!] \approx_t [\![S']\!] . \tag{2.6}$$

A further refinement to soundness is the one that takes into account the *administrative steps* that an encoding might have to perform *before* simulating a step of the source term:

$$\text{if } [\![S]\!] \Longrightarrow_t [\![T]\!] \text{ then there is } S \Longrightarrow_s S' \text{ such that } [\![T]\!] \Longrightarrow_t [\![S']\!] . \tag{2.7}$$

In addition to full-abstraction and operational correspondence, Nestmann (1996) considers two further correctness criteria *effectiveness/efficiency* and preservation/reflection of *deadlocks and divergence*. Let us elaborate only the latter criterion. Nestmann regards as interesting to consider both the reflection and preservation of deadlocks. The former is quite natural: the translation of a term should not deadlock if the given source term does not deadlock.

Preservation of deadlocks is also reasonable as long as potential administrative steps in the target side that might precede deadlock are taken into account. As for divergence, Nestmann distinguishes between the kind of translation performed by compilers and that performed by encodings. Indeed, while a compiler is not expect to add divergent behavior, Nestmann finds an encoding that adds divergence perfectly acceptable. To put this position into context, it is worth noticing that the issue of divergence is central to the work in (Nestmann and Pierce, 2000) where a trade-off between atomicity of committing a choice and divergence is discovered. In fact, Nestmann and Pierce (2000) propose two encodings of the π-calculus with input-guarded choice into the choice-free fragment: one encoding is atomic with respect to choice but introduces divergence; the other encoding is divergence-free but replaces the atomic commitment of choice with gradual commitment. Therefore, there could be scenarios in which correct encodings that add divergence might still be worth having.

A well-known definition of encoding is the one proposed by Palamidessi (2003) as part of a comparison of the expressive power of synchronous and asynchronous communication in the π-calculus. In short, she showed that there is no encoding of the synchronous π-calculus with mixed-choice into the asynchronous π-calculus without choice. This separation result holds under a notion of encoding in which syntactic criteria are captured by the notion of *uniformity*, which is given by the following two conditions:

1. homomorphism with respect to parallel composition, i.e., $[P \quad Q] = [P] \quad [Q]$;

2. preservation of renaming, i.e. for any permutation of names σ in the domain of the target language, there exists a permutation θ in the domain of the target language such that, for all name i, $\sigma(i) = \theta(i)$ and $[\sigma(P)] = \theta([P])$.

Palamidessi argues that uniformity is tailored for the representations of distributed systems, in which issues such as connectivity and coordination should be taken into account by any notion of encoding. This is particularly evident in requiring homomorphism with respect to parallel composition rather than generic compositionality as in (2.2) above. This can be considered as a strong syntactic criterion. However, as Palamidessi claims, in the context of distributed systems homomorphism with respect to parallel composition finds justification as it is essential to ensure that the encoding preserves the degree of distribution of the system, i.e. the encoding of a distributed system does not add coordinating processes (or sites).

Furthermore, in Palamidessi's expressiveness results, encodings are required to be *semantically reasonable*. Quoting Palamidessi (2003), encodings are required to preserve

> a semantics which distinguishes two processes P and Q whenever there exists a (finite or infinite) computation of P in which the intended observables (some visible actions) are different from the observables in any (maximal) computation of Q.

It is worth noticing that this is quite a liberal way of capturing requirements such as operational correspondence and the reflection/preservation of deadlocks and divergence, discussed above. Nestmann (2000) has studied the results in (Palamidessi, 2003) by taking correctness criteria more precise than "preservation of a reasonable semantics". Indeed, he shows that while the π-calculus with mixed-choice can be translated into into the asynchronous π-calculus, a trade-off between divergence and the exact notion of compositionality arises: there are encodings that are uniform but that introduce divergence, whereas encodings that do not introduce divergence only respect generic compositionality.

Recent works have questioned the rôle of full-abstraction as a correctness criteria in encodings of concurrent languages (see Beauxis et al. (2008) for an insightful discussion). Their motivation is that when one is interested in relative expressiveness —rather than in, for instance, the transference of reasoning tools from one language to another— full-abstraction is of little significance, as it is too focused on the actual equivalences considered. This is precisely the motivation for a unified approach to correctness criteria in encodings recently proposed by Gorla (2008).

Gorla's proposal defines a kind of meta-theory for relative expressiveness, based on a set of encodability criteria formulated in abstract terms. As in (Felleisen, 1991), the criteria are divided into *structural* (i.e., syntactic) and *semantic*. The former include a form of compositionality as in (2.2) but where the context is parametrized by the set of free names of the source terms, and a condition on the independence from the actual names used in source terms that generalizes condition (2) in the definition of uniform encoding given by Palamidessi. Semantic criteria include a form of operational correspondence that is defined up to the "garbage terms" that an encoding might produce; divergence reflection, that is, that the encoding does not add divergence; and *success sensitiveness*, i.e., a criteria that requires that based on some notion of "success computation" ensures that a successful source term is mapped into a successful target term. Sensible notions of success include observables such as barbs (Milner and Sangiorgi, 1992) or the outcomes from tests as in behavioral equivalences/preorders based on testing (De Nicola and Hennessy, 1984). A significant advantage of the proposal in (Gorla, 2008) is that it can be exploited by diverse concurrent languages (with different behavioral equivalences) and, to a certain extent, it can be used to reason abstractly about encodings and their properties. In order to illustrate its relevance, the proposal has been instantiated so as to obtain results previously proposed in the literature (Gorla, 2006), and to offer more straightforward proofs for other results.

To conclude, these different proposals for the definition of encoding and its associated correctness criteria only reinforce the idea that a unified notion of encoding is unlikely to exist. In fact, we have seen how the definitions vary depending on the final purpose of the expressiveness study. Hence, a particular definition of encoding should not be judged solely on the basis of its differences with respect to other notions of encoding, which will most likely be aimed at different purposes. A current debate concerns the rôle of full-abstraction as advocated by, e.g., Sangiorgi (1992). In our view, the crucial insight here is to understand that (i) the transference of reasoning techniques from one language to another and (ii) the study of issues of relative expressiveness are essentially two *different goals* that expressiveness results can aim at. As such, one cannot expect correctness criteria aimed at (i) to make sense in settings in which the interest is in (ii), and viceversa.

2.3.3 Main Approaches to Expressiveness

Having reviewed some representative definitions of encoding, here we propose a very broad classification of approaches for obtaining expressiveness results. Our classification does not intend to be exhaustive or conclusive; it provides us with a way of presenting certain used techniques and to emphasize on their differences.

2.3.3.1 Encodability of Computational Models

This is a rather widespread approach to studies of absolute expressiveness. The objective is to to demonstrate the (full) computational expressiveness of a language or model by means of the encodability of a Turing complete model. Notice that, under certain conditions, such an encoding is enough to demonstrate that most relevant decision problems are undecidable.

Examples of Turing complete models used in expressiveness studies are Random Access Machines (RAMs) (Shepherdson and Sturgis, 1963), Minsky machines (Minsky, 1967), and Turing machines. Roughly speaking, both RAMs and Minsky machines are models composed of *registers* (or *counters*) that hold natural numbers, a set of labeled *instructions*, and a *program counter* indicating the instruction currently in execution. The main difference between the two is that while a RAM considers a finite set of registers, a Minsky machine requires only two of them to ensure Turing completeness.

One of the first works that have used this approach is (Busi et al., 2000) in which RAMs are encoded into variants of the coordination language Linda. In turn, such work has served as inspiration for a number of works addressing similar concerns (see, e.g., (Busi and Zavattaro, 2000, 2004; Busi et al., 2009; Maffeis and Phillips, 2005)). The use of complete Turing machines (i.e. with a ribbon or tape, a transition relation, initial and accepting states) has been reported by Hirschkoff et al. (2002) in their study of the expressiveness of the Ambient logic. Similarly, Cardelli and Gordon (2000) have reported an encoding of Turing machines in the Ambient calculus. In addition to Turing complete formalisms, models of computability strictly less expressive than Turing machines have been considered for expressiveness purposes. Christensen (1993) shows that the class of languages generated by Basic Parallel Processes (BPP, a fragment of CCS without communication nor restriction) is contained in the class of context-sensitive languages. In the realm of (process) rewrite systems, efforts towards a general Chomsky-like hierarchy of process languages have been made by Moller (1996) and by Mayr (2000). More recently, Aranda et al. (2007) study fragments of CCS with replication are studied with respect to context-sensitive, context-free, and regular languages.

The fact that several works have appealed to encodings of Turing complete models has raised the question as to what criteria such encodings should satisfy. That is, the issue of the notion of encoding that is crucial to studies of relative expressiveness arises in issues of absolute expressiveness as well. In this case, the criteria are oriented towards determining how *faithful* such encodings are with respect to the behavior of a Turing machine. In fact, notions of Turing completeness that are "weaker" than the classical one have been put forward for explaining the computational expressiveness of certain process calculi. Maffeis and Phillips (2005) and Bravetti and Zavattaro (2009) have analyzed and defined precisely these weaker notions. Let us recall such criteria, as identified by Bravetti and Zavattaro (2009).

Definition 2.10 (Turing completeness for process calculi, (Bravetti and Zavattaro, 2009))**.** *A language $\mathcal{L}$ is said to be* Turing complete*, if given a partial recursive function with a given input, there is a process (i.e., a term of the language) in $\mathcal{L}$ such that*

1. *If the function* is defined *for the given input, then* every *computation of the process* terminates *and make the corresponding output available;*

2. *If the function* is not defined *for the given input, then* every computation *of the process* does not *terminate.*

There are process calculi in which Turing complete models can be encoded in such a way that at least the terminating computations respect the computations of the considered model. Such calculi satisfy the following weaker criterion.

Definition 2.11 (Weak Turing completeness for process calculi, (Bravetti and Zavattaro, 2009))**.** *A language $\mathcal{L}$ is said to be* weakly Turing complete*, if given a partial recursive function with a given input, there is a process (i.e., a term of the language) in $\mathcal{L}$ such that*

1. *If the function* is defined *for the given input, then there exists* at least *one computation of the process that* terminates *and make the corresponding output available;*

2. *If the function* is not defined *for the given input, then* every computation *of the process* does not *terminate.*

Notice that the difference between the two notions is then in the first item. Indeed, if the function is defined according to the first notion every computation of the corresponding process terminates; in the second notion, the corresponding process may have computations that do not terminate. While encodings used to show Turing completeness for process calculi as in

$$\textsc{M-Inc}\ \frac{i : \mathrm{INC}(r_j) \qquad m'_j = m_j + 1 \qquad m'_{1-j} = m_{1-j}}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i+1, m'_0, m'_1)}$$

$$\textsc{M-Dec}\ \frac{i : \mathrm{DECJ}(r_j, k) \qquad m_j \neq 0 \qquad m'_j = m_j - 1 \qquad m'_{1-j} = m_{1-j}}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i+1, m'_0, m'_1)}$$

$$\textsc{M-Jmp}\ \frac{i : \mathrm{DECJ}(r_j, k) \quad m_j = 0}{(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (k, m_0, m_1)}$$

Figure 2.6: Reduction of Minsky machines

Definition 2.10 are sometimes called *deterministic* or *faithful* (see, e.g., (Busi et al., 2009)). In contrast, encodings used to show weak Turing completeness for process calculi as in Definition 2.11 are called *non-deterministic* or *not faithful* (see, e.g., (Aranda, 2009)).

In this dissertation we will consider calculi that satisfy the criteria given by Definition 2.10, as well as calculi that satisfy the criterion given by Definition 2.11. In all cases, we shall exploit encodings of such calculi into Minsky machines. We therefore find it convenient to introduce such a model here.

Minsky machines A Minsky machine (Minsky, 1967) is a Turing complete model composed of a set of sequential, labeled instructions, and two registers. Registers r_j ($j \in \{0, 1\}$) can hold arbitrarily large natural numbers. Instructions $(1 : I_1), \ldots, (n : I_n)$ can be of two kinds: $\mathrm{INC}(r_j)$ adds 1 to register r_j and proceeds to the next instruction; $\mathrm{DECJ}(r_j, k)$ jumps to instruction k if r_j is zero, otherwise it decreases register r_j by 1 and proceeds to the next instruction.

A Minsky machine includes a program counter p indicating the label of the instruction being executed. In its initial state, the machine has both registers set to 0 and the program counter p set to the first instruction. The Minsky machine stops whenever the program counter is set to a non-existent instruction, i.e. $p > n$.

A *configuration* of a Minsky machine is a tuple (i, m_0, m_1); it consists of the current program counter and the values of the registers. Formally, the reduction relation over configurations of a Minsky machine, denoted $\longrightarrow_{\mathrm{M}}$, is defined in Figure 2.6.

2.3.3.2 Decision/Representative Problems

This is an approach to separation results. As argued by Zavattaro (2009), the idea is to discriminate the expressiveness of two variants of the same computational model by investigating the decidability of some decision problem in the two different settings. This allows one to prove that a different interpretation for a given concurrent computational model, or a simple extension of one concurrent computational model, strictly increases the expressive power.

An example of this line of research is (Dufourd et al., 1998) in which separation results for Petri nets with Reset arcs are obtained from the (un)decidability of decision problems such as reachability, termination, coverability, and boundness. In process calculi, this approach has been pioneered by the already cited work on the expressiveness of variants of Linda (Busi et al., 2000) where the decidability of termination is used to prove a separation result between two semantics of the language. Such a decidability result is obtained by endowing the language with a net semantics (in terms of contextual Place/Transition nets) and by defining a deadlock-preserving mapping into finite Place/Transition nets. Another significant application of such approach is (Busi et al., 2009), in which separation results for variants of CCS with different constructs for infinite behavior are reported. In (Busi et al., 2009) the focus is on the (un)decidability of termination and convergence of processes. It is shown that while both properties are undecidable for the variant of CCS with recursion, termination is decidable for the variant for replication. While undecidability results are obtained by exhibiting (termination-preserving) encodings of RAMs (as described above), decidability results are obtained by appealing to the theory of well-structured transition systems (Abdulla et al., 2000; Finkel, 1990; Finkel and Schnoebelen, 2001). In Chapter 5 we shall apply the approach to separation in (Busi et al., 2009) in the context of a higher-order process calculus.

A somewhat related approach to separation results is the one that distinguishes two models based on their capability of solving some well-established problem. That is, a language $\mathcal{L}_1$ is considered to be more expressive than $\mathcal{L}_2$ if the problem can be solved in $\mathcal{L}_1$ but not in $\mathcal{L}_2$. This is a natural approach to follow when the languages at hand are both known to be Turing complete and hence a separation result based on the decidability of some property (as discussed before) is not an option.

Inspired in results by Bougé (1988) in the context of CSP, this approach was used by Palamidessi (2003) for showing the separation between the π calculus with mixed-choice and the asynchronous π-calculus with separate choice. The separation is demonstrated by the fact that, under certain conditions, the *leader election problem* —a problem of distributed consensus in the realm of distributed computing— can be solved in the former but not in the latter. This approach has been rather successful for it has been applied to a number of very diverse calculi (see, e.g., Bougé (1988); Ene and Muntean (1999); Palamidessi (2003); Vigliotti et al. (2007)). More recently, the approach based on leader election has been intensively studied by Vigliotti (2004) in the context of the Ambient calculus. An excellent reference to this approach (and to separation results in general) is (Vigliotti et al., 2007).

Furthermore, while the use of widely known problems is a sensible option for separation results, new problems have been also proposed. This way, for instance, Carbone and Maffeis (2003) have introduced *matching systems* so as to define an expressiveness hierarchy of variants of the π-calculus with polyadic synchronization. Also, Versari et al. (2009) have proposed the *last man standing problem* in order to assess the expressive power of variants of CCS with global and local prorities.

2.3.3.3 By Combinators

This is a less studied approach to the expressiveness of concurrent languages. It aims at the assessing the expressive power of a language by identifying its set of *combinators*, i.e., the elements of the language that are indispensable to represent the *whole* behavior realizable in the language. This is similar to the notion of combinators in the λ-calculus (Barendregt, 1984). Hence, each the combinators of a language is said to be *essential* for in the absence of one of them it is not possible to express the whole language (possibly up to semantic equivalences). Studying the expressiveness of a language based on combinators then appears as a useful method to analyze and categorize its behavior.

The earliest attempt in this direction is by Parrow (1990), where the focus is on the expressiveness of two forms of parallel composition (called *disjoint parallelism* and *linking*) in the context of a small process calculus with synchronization primitives. Parrow identifies three "units" which are responsible for generating all the finite-state behavior that can be expressed in the language. He also establishes conditions under which operators for parallel composition in other algebras can be defined. Parrow (2000) himself took this idea further to the context of mobile processes. In fact, he showed that every process in the synchronous π-calculus without sum and without matching can be mapped (up to weak bisimilarity) as a the parallel composition of a number of *trios*, i.e., prefixes with length at most three, possibly replicated. It is also shown that *duos*, i.e., prefixes of length at most two, are not sufficient to produce the same result. A similar result is shown by Laneve and Victor (2003) for the Fusion calculus.

Based on the results in (Honda and Yoshida, 1994a,b), Yoshida (2002) shows the *minimality* of *five* concurrent combinators that characterize the expressive power of the asynchronous π-calculus without sum. Such combinators correspond to small processes implementing output of messages, duplication of messages, and generation of links. Each of the five combinators is shown to be indispensable to represent the whole behavior of the calculus. Similar ideas were explored by Raja and Shyamasundar (1995a,b).

2.3.3.4 Other approaches

In a slightly different approach to expressiveness issues, a number of works has appealed to the generality of structural operational semantics, their associated rule formats and properties, as a way of gaining insights on the expressive power of languages that fit certain rule formats. For the sake of conciseness, we do not expand on these, and refer the interested reader to, e.g., (de Simone, 1985; Vaandrager, 1992; Dsouza and Bloom, 1995).

2.3.4 Expressiveness for Higher-Order Languages

We conclude this section by reviewing a number of proposals that address the expressiveness of higher-order languages.

Significant studies of the expressiveness of the higher-order communication paradigm are reported in Sangiorgi's PhD dissertation (Sangiorgi, 1992). In Section 2.2.2 we have given the main ideas underlying the compilation $\mathcal{C}$ from higher-order into first-order processes, which is central to his representability result. In (Sangiorgi, 1992) the compilation $\mathcal{C}$ is used to study encodings of (variants of) the λ-calculus into the π-calculus. An encoding of the lazy λ-calculus into HOπ, denoted $\mathcal{H}$, is proposed. The encoding $\mathcal{H}$ enjoys a tight operational correspondence; in fact, it allows to determine that the lazy λ-calculus is a *sub-calculus* of HOπ. Furthermore, it is shown that the composition of $\mathcal{C}$ with $\mathcal{H}$ corresponds with the encoding of the lazy λ-calculus into the π-calculus proposed by Milner (1992). Hence, the usefulness of $\mathcal{C}$ is shown by providing an alternative way of deriving results and transferring reasoning techniques between the lazy λ-calculus and the π-calculus. A similar approach is followed for the call-by-value λ-calculus.

Amadio (1993) obtains a finitely-branching bisimilarity for CHOCS by means of a reduction into bisimulation for a variant of the π-calculus. In such a variant, processes are only allowed to exchange names of *activation channels* (i.e. the channels that trigger a copy of a process in the representation of higher-order communication with first-order one). The desired finitely-branching bisimilarity is obtained by relying on a labeled transition system in which synchronizations on activation channels are distinguished.

Amadio (1994) investigates Core Facile, a λ-calculus with synchronization primitives, parallel composition, and dynamic creation of names. It is intended to serve as an intermediate language between theoretical formalisms (such as CHOCS and the π-calculus) and actual programming languages such as Facile and CML. A control operator is introduced to manipulate evaluation contexts and to define a translation of synchronous communication into asynchronous one. This translation is shown to be adequate, i.e. equivalence of the translated terms implies equivalence of the original terms. By means of a Continuation-Passing Style translation into Core Facile, the control operator is shown to be redundant. A translation of the asynchronous Core Facile into the π-calculus is also presented; this translation is further studied in (Amadio et al., 1995).

The expressiveness of the π-calculus wrt higher-order π was first studied by Sangiorgi (1996b), who isolated hierarchies of fragments of first-order and higher-order calculi with increasingly expressive power. For the former, he identifies a fragment of the π-calculus in which mobility is *internal*, i.e., where outputs are only on private names —no free outputs are allowed. This hierarchy is denoted as $\pi \mathrm{I}^n$, where the n denotes the degree of mobility allowed; e.g., $\pi \mathrm{I}^1$ does not allow mobility and corresponds to the core of CCS. The hierarchy in the higher-order case follows a similar rationale, and is based on the *strictly higher-order* π-calculus, i.e., a higher-order calculus without name-passing features. Also in this hierarchy, the less expressive language (denoted $\mathrm{HO}\pi^1$) corresponds to the core of CCS. Sangiorgi shows that $\pi \mathrm{I}^n$ and $\mathrm{HO}\pi^n$ have the same expressiveness, by exhibiting fully-abstract encodings. Sangiorgi and Walker's encoding of a variant of π-calculus into Higher-Order π-calculus 2001 relies on the abstraction mechanism of the Higher-Order π-calculus (it needs ω-order abstractions).

Vivas et al (Vivas and Dam, 1998; Vivas and Yoshida, 2002; Vivas, 2001) study extensions of the higher-order π-calculus for which the usual encoding of higher-order into first-order (Sangiorgi, 1992) does not work. This is the case of higher-order calculi involving locations, in which certain operations cannot be reduced to reference passing, such as e.g., retrieving some piece of code in a certain location and executing it elsewhere. This issue is first studied by Vivas and Dam (1998) who show that Sangiorgi's encoding schema breaks if *blocking* —a form of restriction based on dynamic scoping— is added to the language. Their motivation for such a construct is the modeling of cryptographic protocols; they claim that usual restriction (based on static scoping) as found in the first- and higher-order π-calculus is not adequate for certain security scenarios. They consider first- and higher-order calculi with mismatching, and show that in the first-order case blocking has the same expressive power as matching and mismatching. A rather involved schema for compiling higher-order calculi with blocking into first-order calculi is proposed; it consists in the communication the syntax tree of a process. Vivas and Yoshida (2002) propose an extension of a higher-order process language with a screening operator called *filtering*. The objective is to represent scenarios of code mobility in which resource access control involves both static and dynamic checkings. The filtering operator is intended to dynamically restrict the visibility of channels of a process: a filtered process can only perform actions present in its associated set of polarized channel names (i.e. channel names with either output or input capabilities). Similarly as blocking in (Vivas and Dam, 1998), the filtering operator exploits dynamic binding to implement a form of encapsulation that blocks external communication in the filtered channels. In this case, the usual restriction operator is claimed to be inadequate as it might allow for scope extrusion of the filtered channels. The higher-order language with filtering is studied with respect to the higher-order language proposed by Yoshida and Hennessy (1999) (which is, essentially, a call-by-value λ-calculus augmented with π-calculus operators). This language is endowed with a type system that assigns *interface types* to processes, i.e. a type that limits the resources a process might have access. An encoding of the latter into the former is proposed as a way of understanding how dynamic checkings enforced by the filtering operator can mimic the static checking enforced by the interface types. The paper shows that the encoding behaves correctly only in the cases in which name extrusion is not involved.

Bundgaard et al. (2006) investigate the expressive power of Homer by encoding the synchronous π-calculus. They succeed in showing that that higher-order process-passing together with mobile resources in, possibly local, named locations are enough to represent π-calculus name-passing. In the Homer case, because of the mobile computing resources and the nested locations, name-passing is a derived notion instead of a primitive. Similarly as the encoding by Thomsen (1990), the encoding of the π-calculus into Homer is not fully-compositional: names are translated at the top-level, separately from the transition of processes.

Bundgaard et al. (2009) study two approaches for obtaining finite-control fragments of Homer in which barbed bisimilarity is decidable. The first approach is based on a type system that bounds the size of processes in terms of their syntactic components (e.g. number of parallel components, location nesting). The second approach exploits results for the π-calculus and uses an encoding of the π-calculus into Homer to transport them in the form of a suitable subcalculus.

Chapter 3

A Core Calculus for Higher-Order Concurrency

In this chapter we introduce HOcore, the core of calculi for higher-order concurrency such as CHOCS (Thomsen, 1989), Plain CHOCS (Thomsen, 1993), and Higher-Order π-calculus (Sangiorgi, 1992, 1996a,b).

The syntax and the semantics of the calculus are given in Section 3.1. Then, Section 3.2 discusses the expressiveness of the language. The main result is an encoding of Minsky machines into HOcore, which allows to infer that the language is Turing complete. Section 3.3 provides some concluding remarks.

3.1 The Calculus

Syntax. We use a, b, c to range over names (also called channels), and x, y, z to range over variables; the sets of names and variables are disjoint.

Definition 3.1. *The set of* HOcore *processes is given by the following syntax:*

$$\begin{array}{rcll}
P, Q & ::= & \overline{a}\langle P \rangle & \textit{output} \\
 & \big| & a(x).\,P & \textit{input prefix} \\
 & \big| & x & \textit{process variable} \\
 & \big| & P \quad Q & \textit{parallel composition} \\
 & \big| & \mathbf{0} & \textit{nil}
\end{array}$$

An input $a(x).\,P$ binds the free occurrences of x in P. We write $\mathrm{fv}(P)$ for the set of free variables in P, and $\mathrm{bv}(P)$ for the bound variables. We identify processes up to a renaming of bound variables. A process is *closed* if it does not have free variables. In a statement, a name is *fresh* if it is not among the names of the objects (processes, actions, etc.) of the statement. We abbreviate $a(x).\,P$, with $x \notin \mathrm{fv}(P)$, as $a.\,P$, $\overline{a}\langle \mathbf{0} \rangle$ as $\overline{a}$, and $P_1 \quad \ldots \quad P_k$ as $\prod_{i=1}^{k} P_i$. Similarly, we write $\prod_1^n P$ as an abbreviation for the parallel composition of n

copies of P. Further, $P\{\widetilde{Q}/\widetilde{x}\}$ denotes the componentwise and simultaneous substitution of variables $\widetilde{x}$ with processes $\widetilde{Q}$ in P (we assume members of $\widetilde{x}$ are distinct).

The size of a process is defined as follows.

Definition 3.2. *The* size *of a process P, written $\#(P)$, is inductively defined as:*

$$\#(\mathbf{0}) = 0 \qquad \#(P \;\; Q) = \#(P) + \#(Q) \qquad \#(x) = 1$$
$$\#(\overline{a}\langle P\rangle) = 1 + \#(P) \qquad \#(a(x).\,P) = 1 + \#(P)$$

Semantics. Now we describe the LTS, which is defined on open processes. There are three forms of transitions: τ transitions $P \xrightarrow{\tau} P'$; input transitions $P \xrightarrow{a(x)} P'$, meaning that P can receive at a a process that will replace x in the continuation P'; and output transitions $P \xrightarrow{\overline{a}\langle P'\rangle} P''$ meaning that P emits P' at a, and in doing so it evolves to P''. We use α to indicate a generic label of a transition. The notions of free and bound variables extend to labels as expected.

$$\textsc{Inp} \quad a(x).\,P \xrightarrow{a(x)} P \qquad\qquad \textsc{Out} \quad \overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}$$

$$\textsc{Act1} \quad \frac{P_1 \xrightarrow{\alpha} P_1' \qquad \mathrm{bv}(\alpha) \cap \mathrm{fv}(P_2) = \emptyset}{P_1 \;\; P_2 \xrightarrow{\alpha} P_1' \;\; P_2}$$

$$\textsc{Tau1} \quad \frac{P_1 \xrightarrow{\overline{a}\langle P\rangle} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \;\; P_2 \xrightarrow{\tau} P_1' \;\; P_2'\{P/x\}}$$

(We have omitted Act2 and Tau2, the symmetric counterparts of the last two rules.)

Definition 3.3. *The* structural congruence *relation is the smallest congruence generated by the following laws:*
$P \;\; \mathbf{0} \equiv P,\;\; P_1 \;\; P_2 \equiv P_2 \;\; P_1,\;\; P_1 \;\; (P_2 \;\; P_3) \equiv (P_1 \;\; P_2) \;\; P_3.$

Reductions $P \longrightarrow P'$ are defined as $P \equiv\xrightarrow{\tau}\equiv P'$. We now state a few results which will be important later.

Lemma 3.1. *If $P \xrightarrow{\alpha} P'$ and $P \equiv Q$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.*

Proof. By induction on the derivation of $P \equiv Q$, then by case analysis on $P \xrightarrow{\alpha} Q$. □

Definition 3.4. *A variable x* is guarded *in $P \in$* HOcore *(or simply* guarded, *when P is clear from the context) if x only occurs free in an output or in subexpressions of P of the form $\pi.\,P'$, where π is any prefix. A process $P \in$* HOcore *is* guarded *(or has* guarded variables*) if all its free variables are guarded.*

In particular, notice that if x is guarded in P then it does not appear in evaluation contexts (i.e. contexts which allow transitions in the hole position), and if x is not free in P then it is guarded in P. In the lemma below, we recall that an output action from an open process may contain free variables, thus $\alpha\{\widetilde{R}/\widetilde{x}\}$ is the action obtained from α by applying the substitution $\{\widetilde{R}/\widetilde{x}\}$.

Lemma 3.2. *Suppose that $P \in$ HOcore and variables $\widetilde{x}$ are guarded in P. Then, for all $\widetilde{R} \in$ HOcore we have:*

1. *If $P \xrightarrow{\alpha} P'$, with variables in $\widetilde{R}$ disjoint from those in P, α and $\widetilde{x}$, then $P\{\widetilde{R}/\widetilde{x}\} \xrightarrow{\alpha\{\widetilde{R}/\widetilde{x}\}} P'\{\widetilde{R}/\widetilde{x}\}$;*
2. *If $P\{\widetilde{R}/\widetilde{x}\} \xrightarrow{\alpha'} M'$, with variables in $\widetilde{R}$ disjoint from those in P, α' and $\widetilde{x}$, then there is P' such that $P \xrightarrow{\alpha} P'$ and $M' = P'\{\widetilde{R}/\widetilde{x}\}$, $\alpha' = \alpha\{\widetilde{R}/\widetilde{x}\}$.*

Proof. By induction on the transitions. □

Lemma 3.3. *For all $P \in$ HOcore and x there is $P' \in$ HOcore with x guarded in P', and $n \geq 0$ such that*

1. *$P \equiv P' \;\; \prod_1^n x$*
2. *$P\{R/x\} \equiv P'\{R/x\} \;\; \prod_1^n R$, for all $R \in$ HOcore.*

Proof. By induction on the structure of processes. □

3.2 Expressiveness of HOcore

We first present encodings of a simple form of guarded choice and guarded replication. Then we use such encodings to encode Minsky machines.

3.2.1 Guarded Choice

We extend the HOcore syntax with a simple form of guarded choice to choose between different behaviors. Assume, for instance, that a_i should trigger P_i, for $i \in \{1, 2\}$. We write this as $a_1.\,P_1 + a_2.\,P_2$, and we write the choice of the behavior P_i as $\widehat{a_i}$. We then have, for each i, the reduction $(a_1.\,P_1 + a_2.\,P_2) \;\; \widehat{a_i} \longrightarrow P_i$. We encode these new operators as follows.

$$
\begin{aligned}
[a_1.\,P_1 + a_2.\,P_2]_+ &= \overline{a_1}\langle [P_1]_+ \rangle \;\; \overline{a_2}\langle [P_2]_+ \rangle \\
[\widehat{a_1}]_+ &= a_2(x_2).\, a_1(x_1).\, x_1 \\
[\widehat{a_2}]_+ &= a_1(x_1).\, a_2(x_2).\, x_2
\end{aligned}
$$

The translation is an homomorphism on the other operators. This way, $[\widehat{a_i}]_+$ for $i \in \{1, 2\}$ is a process that consumes both P_i's and spawns the one chosen. This encoding is correct as long as all guards used in the choices are different and there is at most one message at a guard, $\widehat{a_1}$ or $\widehat{a_2}$ in the previous example, enabled at any given time. The encoding introduces an extra communication for every guarded choice.

With a slight abuse of notation, in what follows we shall use disjoint sums inside HOcore processes without explicitely referring to the encoding $[\cdot]_+$.

3.2.2 Input-guarded Replication

We follow the standard encoding of replication in higher-order process calculi, adapting it to input-guarded replication so as to make sure that diverging behaviors are not introduced. As there is no restriction in HOcore, the encoding is not compositional and replications cannot be nested.

Definition 3.5. *Assume a fresh name c. The encoding of* input-guarded replication *is as follows:*

$$[\![!a(z).\, P]\!]_{\mathrm{i!}} = a(z).\, (c(x).\, (x \;\; \overline{c}\langle x\rangle \;\; P)) \;\; \overline{c}\langle a(z).\, (c(x).\, (x \;\; \overline{c}\langle x\rangle \;\; P))\rangle$$

where P contains no replications (nested replications are forbidden), and $[\![\cdot]\!]_{\mathrm{i!}}$ is an homomorphism on the other process constructs in HOcore*.*

It is worth noticing that after the input on a, a copy of P is only released after a synchronization on c. More precisely, we have the following correctness statement. We use $Q \not\longrightarrow$ to denote that there is no Q' such that $Q \longrightarrow Q'$, both in HOcore and for Minsky Machines.

Lemma 3.1 (Correctness of $[\![\cdot]\!]_{\mathrm{i!}}$). *Let P be a* HOcore *process with non-nested input-guarded replications.*

- *If $[\![P]\!]_{\mathrm{i!}} \longrightarrow Q$ then $\exists P'$ such that $P \longrightarrow P'$ and either $[\![P']\!]_{\mathrm{i!}} = Q$ or $Q \longrightarrow [\![P']\!]_{\mathrm{i!}}$.*
- *If $P \longrightarrow P'$ then either $[\![P]\!]_{\mathrm{i!}} \longrightarrow [\![P']\!]_{\mathrm{i!}}$ or $[\![P]\!]_{\mathrm{i!}} \longrightarrow\longrightarrow [\![P']\!]_{\mathrm{i!}}$.*
- *$[\![P]\!]_{\mathrm{i!}} \not\longrightarrow$ iff $P \not\longrightarrow$.*

Proof. By induction on the transitions. □

With a slight abuse of notation, in what follows we shall use input-guarded replications inside HOcore processes without explicitely referring to the encoding $[\![\cdot]\!]_{\mathrm{i!}}$.

3.2.3 Minsky machines

We present an encoding of Minsky machines (see Section 2.3.3) into HOcore. The encoding shows that HOcore is Turing complete and, as the encoding preserves termination, it also shows that termination in HOcore is undecidable. The only form of non-determinism in the encoding is due to possible unfoldings of (the encoding of) recursive definitions after they have been used; otherwise, at any step, in the encoding any process has at most one reduction.

We first show how to count and test for zero in HOcore; then, we present the encoding of a Minsky machine into HOcore, denoted as $[\![\cdot]\!]_{\mathrm{M}}$ (see Table 3.1).

Instructions $(i : I_i)$

$$
\begin{aligned}
[\![(i : \mathrm{INC}(r_j))]\!]_{\mathrm{M}} &= !p_i.(\widehat{inc_j} \;\; ack.\overline{p_{i+1}}) \\
[\![(i : \mathrm{DECJ}(r_j, k))]\!]_{\mathrm{M}} &= !p_i.(\widehat{dec_j} \;\; ack.(z_j.\overline{p_k} + n_j.\overline{p_{i+1}})
\end{aligned}
$$

Registers r_j

$$
\begin{aligned}
[\![r_j = 0]\!]_{\mathrm{M}} &= (inc_j.\overline{r_j^S}\langle \llparenthesis\, 0 \,\rrparenthesis_j \rangle \;+\; dec_j.(\overline{r_j^0} \;\; \widehat{z_j}))\;\; \mathrm{REG}_j \\
[\![r_j = m]\!]_{\mathrm{M}} &= (inc_j.\overline{r_j^S}\langle \llparenthesis\, m \,\rrparenthesis_j \rangle \;+\; dec_j.\llparenthesis\, m-1 \,\rrparenthesis_j)\;\; \mathrm{REG}_j
\end{aligned}
$$

where:

$$
\begin{aligned}
\mathrm{REG}_j &= !r_j^0.(\overline{ack} \;\; inc_j.\overline{r_j^S}\langle \llparenthesis\, 0 \,\rrparenthesis_j \rangle \;+\; dec_j.(\overline{r_j^0} \;\; \widehat{z_j})) \\
&\quad\; !r_j^S(Y).(\overline{ack} \;\; inc_j.\overline{r_j^S}\langle \overline{r_j^S}\langle Y \rangle \;\; \widehat{n_j} \rangle \;+\; dec_j.Y)
\end{aligned}
$$

$$
\llparenthesis\, k \,\rrparenthesis_j = \begin{cases} \overline{r_j^0} \;\; \widehat{n_j} & \text{if } k = 0 \\ \overline{r_j^S}\langle \llparenthesis\, k-1 \,\rrparenthesis_j \rangle \;\; \widehat{n_j} & \text{if } k > 0. \end{cases}
$$

Figure 3.1: Encoding of Minsky machines into HOcore

Counting in HOcore. The cornerstone of our encoding is the definition of counters that may be tested for zero. Numbers are represented as nested higher-order processes: the encoding of a number $k+1$ in register j, denoted $\llparenthesis\, k+1 \,\rrparenthesis_j$, is the parallel composition of two processes: $\overline{r_j^S}\langle \llparenthesis\, k \,\rrparenthesis_j \rangle$ (the successor of $\llparenthesis\, k \,\rrparenthesis_j$) and a flag $\widehat{n_j}$. The encoding of zero comprises such a flag, as well as the message $\overline{r_j^0}$. As an example, $\llparenthesis\, 2 \,\rrparenthesis_j$ is $\overline{r_j^S}\langle \overline{r_j^S}\langle \overline{r_j^0} \;\; \widehat{n_j} \rangle \;\; \widehat{n_j} \rangle \;\; \widehat{n_j}$.

Registers. Registers are counters that may be incremented and decremented. They consist of two parts: their current state and two mutually recursive processes used to generate a new state after an increment or decrement of the register. The state depends on whether the current value of the register is zero or not, but in both cases it consists of a choice between an increment and a decrement. In case of an increment, a message on r_j^S is sent containing the current register value, for instance m. This message is then received by the recursive definition of r_j^S that creates a new state with value $m+1$, ready for further increment or decrement. In case of a decrement, the behavior depends on the current value, as specified in the reduction relation in Table 2.6. If the current value is zero, then it stays at zero, recreating the state corresponding to zero for further operations using the message on r_j^0, and it spawns a flag $\widehat{z_j}$ indicating that a decrement on a zero-valued register has occurred. If the current value m is strictly greater than zero, then the process $\llparenthesis\, m-1 \,\rrparenthesis_j$ is spawned. If m was equal to 1, this puts the state of the register to zero (using a message on r_j^0). Otherwise, it keeps the message in a non-zero state, with value $m-1$, using a message on r_j^S. In both cases a flag $\widehat{n_j}$ is spawned to indicate that the register was not equal to zero before the decrement. When an increment or decrement has been processed, that is when the new current state has been created, an acknowledgment is sent to proceed with the execution of the next instruction.

Instructions. The encoding of instructions goes hand in hand with the encoding of registers. Each instruction $(i : I_i)$ is a replicated process guarded by p_i, which represents the program counter when $p = i$. Once p_i is consumed, the instruction is active and an interaction with a register occurs. In case of an increment instruction, the corresponding choice is sent to the relevant register and, upon reception of the acknowledgment, the next instruction is spawned. In case of a decrement, the corresponding choice is sent to the register, then an acknowledgment is received followed by a choice depending on whether the register was zero, resulting in a jump to the specified instruction, or the spawning of the next instruction otherwise.

The encoding of a configuration of a Minsky machine thus requires a finite number of fresh names (linear on n, the number of instructions).

Definition 3.6. *Let N be a Minsky machine with registers $r_0 = m_0$, $r_1 = m_1$ and instructions $(1 : I_1), \ldots, (n : I_n)$. Suppose fresh, pairwise different names r_j^0, r_j^S, $p_1, \ldots, p_n$, inc_j, dec_j, ack (for $j \in \{0, 1\}$). Given the encodings in Table 3.1, a configuration (i, m_0, m_1) of N is encoded as*

$$\overline{p_i} \;\; [\![r_0 = m_0]\!]_{\mathsf{M}} \;\; [\![r_1 = m_1]\!]_{\mathsf{M}} \;\; \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathsf{M}} \, .$$

Correctness of the Encoding. In HOcore, we write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$, and $P \Uparrow$ if P has an infinite sequence of reductions. Similarly, in Minsky machines $\longrightarrow_{\mathsf{M}}^*$ is the reflexive and transitive closure of $\longrightarrow_{\mathsf{M}}$, and $N \Uparrow_{\mathsf{M}}$ means that N has an infinite sequence of reductions.

Lemma 3.2. *Let N be a Minsky machine. We have:*

1. *$N \longrightarrow_{\mathsf{M}}^* N'$ iff $[\![N]\!]_{\mathsf{M}} \longrightarrow^* [\![N']\!]_{\mathsf{M}}$;*
2. *if $[\![N]\!]_{\mathsf{M}} \longrightarrow^* P_1$ and $[\![N]\!]_{\mathsf{M}} \longrightarrow^* P_2$, then there exists N' such that $P_1 \longrightarrow^* [\![N']\!]_{\mathsf{M}}$ and $P_2 \longrightarrow^* [\![N']\!]_{\mathsf{M}}$;*
3. *$N \Uparrow_{\mathsf{M}}$ iff $[\![N]\!]_{\mathsf{M}} \Uparrow$.*

The proof of Lemma 3.2 relies on two properties. The first one, given by Lemma 3.3, ensures that for every computation of the Minsky machine the encoding can perform a finite, non-empty sequence of reductions that correspond to the one made by the machine. Using Lemma 3.1, the second property (Lemma 3.4) ensures that if the process encoding a Minsky machine has a reduction then (i) the machine also has a reduction, and (ii) the encoding has a finite sequence of reductions that correspond to the result of the reduction of the Minsky machine.

We now proceed with the proofs. In what follows we assume a Minsky machine N with instructions $(1 : I_1), \ldots, (n : I_n)$ and with registers $r_0 = m_0$ and $r_1 = m_1$. The encoding of a configuration (i, m_0, m_1) of N is denoted $[\![(i, m_0, m_1)_N]\!]_{\mathsf{M}}$. We use $\longrightarrow^j$ to stand for a sequence of j reductions.

Lemma 3.3. *Let (i, m_0, m_1) be a configuration of a Minsky machine N. If $(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i', m_0', m_1')$ then there exist a finite j and a process P such that $[\![(i, m_0, m_1)_N]\!]_{\mathrm{M}} \longrightarrow^{j} P$ and $P = [\![(i', m_0', m_1')_N]\!]_{\mathrm{M}}$.*

Proof. We proceed by case analysis on the instruction performed by the Minsky machine. Hence, we distinguish three cases corresponding to the behaviors associated to rules M-Jmp, M-Dec, and M-Inc.

Case M-Jmp We have a Minsky configuration (i, m_0, m_1) with $m_0 = 0$ and $(i : \mathrm{DECJ}(r_0, k))$. By Definition 3.6, its encoding in HOcore is as follows:

$$\begin{array}{rcl} [\![(i, m_0, m_1)_N]\!]_{\mathrm{M}} & = & \overline{p_i} \;\; [\![r_0 = 0]\!]_{\mathrm{M}} \;\; [\![r_1 = m_1]\!]_{\mathrm{M}} \\ & & [\![(i : \mathrm{DECJ}(r_0, k))]\!]_{\mathrm{M}} \displaystyle\prod_{l=1..n, l \neq i} [\![(l : I_l)]\!]_{\mathrm{M}} \end{array}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i. The content of the instruction is thus exposed, and we then have

$$[\![(i, m_0, m_1)_N]\!]_{\mathrm{M}} \longrightarrow [\![r_0 = 0]\!]_{\mathrm{M}} \;\; \widehat{dec_0} \;\; ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \;\; S = P_1$$

where $S = [\![r_1 = m_1]\!]_{\mathrm{M}} \;\; \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathrm{M}}$ stands for the rest of the system. The only transition possible at this point is the behavior selection on dec_0, which yields the following:

$$P_1 \longrightarrow \overline{r_0^0} \;\; \widehat{z_0} \;\; \mathrm{REG}_0 \;\; ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \;\; S = P_2$$

Now there is a synchronization between $\overline{r_0^0}$ and REG_0 for reconstructing the register

$$\begin{array}{rcl} P_2 & \longrightarrow & \widehat{z_0} \;\; \overline{ack} \;\; (inc_0.\,\overline{r_0^S}\langle (\!|\, 0 \,|\!)_0 \rangle \; + \; dec_0.\,(\overline{r_0^0} \;\; \widehat{z_0})) \;\; \mathrm{REG}_0 \\ & & ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \;\; S = P_3 \end{array}$$

Once the register has been re-created, register and instruction can now synchronize on ack:

$$\begin{array}{rcl} P_3 & \longrightarrow & \widehat{z_0} \;\; (inc_0.\,\overline{r_0^S}\langle (\!|\, 0 \,|\!)_0 \rangle \; + \; dec_0.\,(\overline{r_0^0} \;\; \widehat{z_0})) \;\; \mathrm{REG}_0 \\ & & z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}} \;\; S = P_4 \end{array}$$

At this point, the only possible transition is the behavior selection on z_0, which indicates that the content of r_0 was indeed zero:

$$P_4 \longrightarrow (inc_0.\,\overline{r_0^S}\langle (\!|\, 0 \,|\!)_0 \rangle \; + \; dec_0.\,(\overline{r_0^0} \;\; \widehat{z_0})) \;\; \mathrm{REG}_0 \;\; \overline{p_k} \;\; S = P_5$$

Using the definitions of $[\![\cdot]\!]_{\mathrm{M}}$ and S, and some reordering, we note that P_5 can be equivalently written as

$$P_5 = \overline{p_k} \;\; [\![r_0 = 0]\!]_{\mathrm{M}} \;\; [\![r_1 = m_1]\!]_{\mathrm{M}} \prod_{l=1}^{n} [\![(l : I_l)]\!]_{\mathrm{M}}$$

which, in turn, corresponds to the encoding of $[\![(k, 0, m_1)_N]\!]_{\mathrm{M}}$, as desired.

Case M-Dec We have a Minsky configuration (i, m_0, m_1) with $m_0 = c$ (for some $c > 0$) and $(i : \mathrm{DECJ}(r_0, k))$. By Definition 3.6, its encoding in HOcore is as follows:

$$\begin{aligned}[(i, m_0, m_1)_N]_{\mathrm{M}} \quad = \quad & \overline{p_i} \;\; [r_0 = c]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \\ & [(i : \mathrm{DECJ}(r_0, k))]_{\mathrm{M}} \prod_{l=1..n, l \neq i} [(l : I_l)]_{\mathrm{M}}\end{aligned}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i. The content of the instruction is thus exposed, and we then have

$$[(i, m_0, m_1)_N]_{\mathrm{M}} \longrightarrow [r_0 = c]_{\mathrm{M}} \;\; \widehat{dec_0} \;\; ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \;\; S = P_1$$

where $S = [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{l=1}^{n} [(l : I_l)]_{\mathrm{M}}$ stands for the rest of the system. The only transition possible at this point is the behavior selection on dec_0, which yields the following:

$$P_1 \longrightarrow (\!|\; c-1 \;|\!)_0 \;\; \mathrm{REG}_0 \;\; ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \;\; S = P_2$$

It is worth recalling that $(\!|\; c-1 \;|\!)_0 = \overline{r_0^S}\langle (\!|\; c-2 \;|\!)_0 \rangle \;\; \widehat{n_0}$. Considering this, now there is a synchronization between $\overline{r_0^S}$ and REG_0 for decrementing the value of the register

$$\begin{aligned} P_2 \quad \longrightarrow \quad & \widehat{n_0} \;\; \overline{ack} \;\; (inc_0.\,(\overline{r_0^S}\langle (\!|\; c-1 \;|\!)_0 \rangle \;\; \widehat{n_0}) \; + \; dec_0.\,(\!|\; c-2 \;|\!)_0) \;\; \mathrm{REG}_0 \\ & ack.\,(z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}}) \;\; S = P_3 \end{aligned}$$

Once the register has been re-created, register and instruction can now synchronize on ack:

$$\begin{aligned} P_3 \quad \longrightarrow \quad & \widehat{n_0} \;\; (inc_0.\,(\overline{r_0^S}\langle (\!|\; c-1 \;|\!)_0 \rangle \;\; \widehat{n_0}) \; + \; dec_0.\,(\!|\; c-2 \;|\!)_0) \;\; \mathrm{REG}_0 \\ & z_0.\,\overline{p_k} + n_0.\,\overline{p_{i+1}} \;\; S = P_4 \end{aligned}$$

At this point, the only possible transition is the behavior selection on n_0, which indicates that the content of r_0 was greater than zero:

$$P_4 \longrightarrow (inc_0.\,(\overline{r_0^S}\langle (\!|\; c-1 \;|\!)_0 \rangle \;\; \widehat{n_0}) \; + \; dec_0.\,(\!|\; c-2 \;|\!)_0) \;\; \mathrm{REG}_0 \;\; \overline{p_{i+1}} \;\; S = P_5$$

Using the definitions of $[\cdot]_{\mathrm{M}}$ and S, and some reordering, we note that P_5 can be equivalently written as

$$P_5 = \overline{p_{i+1}} \;\; [r_0 = c-1]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{l=1}^{n} [(l : I_l)]_{\mathrm{M}}$$

which, in turn, corresponds to the encoding of $[(i+1, c-1, m_1)_N]_{\mathrm{M}}$, as desired.

Case M-Inc We have a Minsky configuration (i, m_0, m_1) with $(i : \mathrm{INC}(r_0))$. Its encoding in HOcore is as follows:

$$\begin{aligned}[(i, m_0, m_1)_N]_{\mathrm{M}} \quad = \quad & \overline{p_i} \;\; [r_0 = m_0]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \\ & [(i : \mathrm{INC}(r_0))]_{\mathrm{M}} \prod_{l=1..n, l \neq i} [(l : I_l)]_{\mathrm{M}}\end{aligned}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i:

$$[(i, m_0, m_1)_N]_{\mathrm{M}} \longrightarrow [r_0 = m_0]_{\mathrm{M}} \;\; \widehat{inc_0} \;\; ack.\overline{p_{i+1}} \;\; S = P_1$$

where $S = [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{l=1}^{n} [(l : I_l)]_{\mathrm{M}}$ stands for the rest of the system. The only transition possible at this point is the behavior selection on inc_0. After such a selection we have

$$P_1 \longrightarrow \overline{r_0^{\mathrm{S}}}\langle (\!| \; m_0 \; |\!)_0 \rangle \;\; \mathrm{REG}_0 \;\; ack.\overline{p_{i+1}} \;\; S = P_2$$

Now there is a synchronization between $\overline{r_0^{\mathrm{S}}}$ and REG_0 for incrementing the value of the register

$$\begin{aligned} P_2 \quad \longrightarrow \quad & \overline{ack} \;\; (inc_0.\,(\overline{r_0^{\mathrm{S}}}\langle \overline{r_0^{\mathrm{S}}}\langle (\!| \; m_0 \; |\!)_0 \rangle \;\; \widehat{n_0} \rangle) + dec_0.\,((\!| \; m_0 \; |\!)_0)) \;\; \mathrm{REG}_0 \\ & ack.\overline{p_{i+1}} \;\; S = P_3 \end{aligned}$$

Once the register has been re-created, a synchronization on ack is possible

$$\begin{aligned} P_3 \quad \longrightarrow \quad & (inc_0.\,(\overline{r_0^{\mathrm{S}}}\langle \overline{r_0^{\mathrm{S}}}\langle (\!| \; m_0 \; |\!)_0 \rangle \;\; \widehat{n_0} \rangle) + dec_0.\,((\!| \; m_0 \; |\!)_0)) \;\; \mathrm{REG}_0 \\ & \overline{p_{i+1}} \;\; S = P_4 \end{aligned}$$

Using the definition of $(\!| \cdot |\!)_j$ we note that P_4 actually corresponds to

$$P_4 = (inc_0.\,(\overline{r_0^{\mathrm{S}}}\langle (\!| \; m_0 + 1 \; |\!)_0 \rangle + dec_0.\,((\!| \; m_0 \; |\!)_0)) \;\; \mathrm{REG}_0 \;\; \overline{p_{i+1}} \;\; S$$

which in turn can be written as

$$P_4 = \overline{p_{i+1}} \;\; [r_0 = m_0 + 1]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{l=1}^{n} [(l : I_l)]_{\mathrm{M}}$$

which corresponds to the encoding of $[(i + 1, m_0 + 1, m_1)_N]_{\mathrm{M}}$, as desired. □

Lemma 3.4. *Let (i, m_0, m_1) be a configuration of a Minsky machine N.*
If $[(i, m_0, m_1)_N]_{\mathrm{M}} \longrightarrow P_1$ then for every computation of P_1 there exists a P_j such that $P_j = [(i', m'_0, m'_1)_N]_{\mathrm{M}}$ and $(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i', m'_0, m'_1)$.

Proof. Consider the reduction $[(i, m_0, m_1)_N]_{\mathrm{M}} \longrightarrow P_1$. An analysis of the structure of process $[(i, m_0, m_1)_N]_{\mathrm{M}}$ reveals that, in all cases, the only possibility for the first step corresponds to the consumption of the program counter p_i. This implies that there exists an instruction labeled with i, that can be executed from the configuration (i, m_0, m_1). We proceed by a case analysis on the possible instruction, considering also the fact that the register on which the instruction acts can hold a value equal or greater than zero. In all cases, it can be shown that computation evolves deterministically, until reaching a process in which a new program counter (that is, some $\overline{p_{i'}}$) appears. The program counter $\overline{p_{i'}}$ is inside a process that corresponds to $[(i', m'_0, m'_1)_N]_{\mathrm{M}}$, where $(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i', m'_0, m'_1)$. The analysis follows the same lines as the one reported for the proof of Lemma 3.3, and we omit it. □

Lemma 3.5. *Let N be a Minsky machine. We have that $N \not\longrightarrow_M$ if and only if $[N]_{\mathrm{M}} \not\longrightarrow$.*

Proof. Straightforward from Lemmas 3.3 and 3.4. □

The results above guarantee that HOcore is Turing complete, and since the encoding preserves termination, it entails the following corollary.

Corollary 3.1. *Termination in* HOcore *is undecidable.*

3.3 Concluding Remarks

The encoding of Turing complete models (such as Minsky and Random Access Machines, RAMs) is a common proof technique for carrying out expressiveness studies. Our encoding of Minsky machines into HOcore resembles in structure those in (Busi et al., 2003, 2009) where RAMs are used to investigate the expressive power of restriction and replication in name-passing calculi, and those in (Busi and Zavattaro, 2004), where the impact of restriction and movement on the expressiveness of Ambient calculi is studied. The similarities can be explained by the fact that all the encodings share the same guiding principle: representing counting as the nesting of suitable components. Those components are restricted names in CCS (Busi et al., 2009), recursive definitions in π-calculus (Busi et al., 2003), ambients themselves in Ambient calculus (Busi and Zavattaro, 2004), and higher-order messages in our case. Note that by combining our encoding with the one of higher-order π into π-calculus in (Sangiorgi and Walker, 2001), we obtain an encoding very similar to the one in (Busi et al., 2003). However, we do not know of other works using Turing complete models for proving expressiveness results in the context of higher-order process calculi.

Chapter 4

Behavioral Theory of HOcore

This chapter develops the behavioral theory of HOcore. In Section 4.1 a notion of strong bisimilarity for HOcore is studied; such a notion it is unique in that it coincides with other sensible behavioral equivalences in the higher-order setting. The most remarkable property of strong bisimilarity in HOcore is that it is *decidable*. Section 4.2 analyzes the relationship between strong bisimilarity and (asynchronous) barbed congruence. Section 4.3 introduces a sound and complete axiomatization of strong bisimilarity. This axiomatization is then used to obtain an upper bound to the complexity of the bisimilarity problem. In Section 4.4 it is shown that strong bisimilarity becoms undecidable if at least four static restriction are added to the calculus. Section 4.5 briefly analyzes the impact of some extensions to the language on the decidability results. Section 4.6 concludes.

4.1 Bisimilarity in HOcore

In this section we prove that the main forms of strong bisimilarity for higher-order process calculi coincide in HOcore, and that such a relation is decidable. As a key ingredient for our results, we introduce *open Input/Output (IO) bisimulations* in which the variable of input prefixes is never instantiated and τ-transitions are not observed. To the best of our knowledge, HOcore is the first calculus where IO bisimulation is discriminating enough to provide a useful characterization of process behavior.

We define different kinds of bisimulations by appropriate combinations of the clauses below.

Definition 4.1 (HOcore bisimulation clauses, open processes). *A symmetric relation $\mathcal{R}$ on* HOcore *processes is*

1. *a* τ-bisimulation *if $P \mathcal{R} Q$ and $P \xrightarrow{\tau} P'$ imply that there is Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \mathcal{R} Q'$;*

2. *a* higher-order output bisimulation *if $P \mathcal{R} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ imply that there are Q', Q'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ with $P' \mathcal{R} Q'$ and $P'' \mathcal{R} Q''$;*

3. *an* output normal bisimulation *if $P \mathcal{R} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ imply that there are Q', Q'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ with $m.P'' \;\; P' \mathcal{R} m.Q'' \;\; Q'$, where m is fresh.*

4. *an* open *bisimulation if whenever $P \mathcal{R} Q$:*
   - *$P \xrightarrow{a(x)} P'$ implies that there is Q' such that $Q \xrightarrow{a(x)} Q'$ and $P' \mathcal{R} Q'$,*
   - *$P \equiv x \;\; P'$ implies that there is Q' such that $Q \equiv x \;\; Q'$ and $P' \mathcal{R} Q'$.*

Definition 4.2 (HOcore bisimulation clauses, closed processes). *A symmetric relation $\mathcal{R}$ on closed* HOcore *processes is*

1. *an* output context bisimulation *if $P \mathcal{R} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ imply that there are Q', Q'' such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ and for all S with $\mathrm{fv}(S) \subseteq x$, it holds that $S\{^{P''}\!/x\} \;\; P' \mathcal{R} S\{^{Q''}\!/x\} \;\; Q'$;*

2. *an* input normal bisimulation *if $P \mathcal{R} Q$ and $P \xrightarrow{a(x)} P'$ imply that there is Q' such that $Q \xrightarrow{a(x)} Q'$ and $P'\{^{\overline{m}\langle \mathbf{0}\rangle}\!/x\} \mathcal{R} Q'\{^{\overline{m}\langle \mathbf{0}\rangle}\!/x\}$, where m is fresh;*

3. closed *if $P \mathcal{R} Q$ and $P \xrightarrow{a(x)} P'$ imply that there is Q' such that $Q \xrightarrow{a(x)} Q'$ and for all closed R, it holds that $P'\{^{R}\!/x\} \mathcal{R} Q'\{^{R}\!/x\}$.*

A combination of the bisimulation clauses in Definitions 4.1 and 4.2 is *complete* if it includes exactly one clause for input and output transitions (in contrast, it need not include a clause for τ-transitions).[1] We will show that all complete combinations coincide. We only give a name to those combinations that represent known forms of bisimulation for higher-order processes or that are needed in our proofs. In each case, as usual, a *bisimilarity* is the union of all bisimulations, and is itself a bisimulation (the functions from relations to relations that represent the bisimulation clauses in Definitions 4.1 and 4.2 are all monotonic).

Definition 4.3. Higher-order bisimilarity, *written $\sim_{\mathrm{HO}}$, is the largest relation on closed* HOcore *processes that is a τ-bisimulation, a higher-order output bisimulation, and is closed.*

Context bisimilarity, *written $\sim_{\mathrm{CON}}$, is the largest relation on closed* HOcore *processes that is a τ-bisimulation, an output context bisimulation, and is closed.*

Normal bisimilarity, *written $\sim_{\mathrm{NOR}}$, is the largest relation on closed* HOcore *processes that is a τ-bisimulation, an output normal bisimulation, and an input normal bisimulation.*

IO bisimilarity, *written $\sim^{\circ}_{\mathrm{IO}}$, is the largest relation on* HOcore *processes that is a higher-order output bisimulation and is open.*

Open normal bisimilarity, *written $\sim^{\circ}_{\mathrm{NOR}}$, is the largest relation on* HOcore *processes that is a τ-bisimulation, an output normal bisimulation, and is open.*

[1] The clauses of Definition 4.2 are however tailored to closed processes, therefore combining them with clause 4 in Definition 4.1 has little interest.

Environmental bisimilarity (Sangiorgi et al., 2007), a recent proposal of bisimilarity for higher-order calculi, in HOcore roughly corresponds to (and indeed coincides with) the complete combination that is a τ-bisimulation, an output normal bisimulation, and is closed.

Remark 4.1. *The input clause of Definition 4.2(3) is in the* late *style. It is known (Sangiorgi, 1992) that in calculi of pure higher-order concurrency early and late clauses are equivalent.*

Remark 4.2. *In contrast with ordinary normal bisimulation (Sangiorgi, 1992; Jeffrey and Rathke, 2005), our clause for output normal bisimulation does not use a replication in front of the introduced fresh name. Such a replication would be needed in extensions of the calculus (e.g., with recursion or restriction).*

A bisimilarity on closed processes is extended to open processes as follows.

Definition 4.4 (Extension of bisimilarities)**.** *Let $\mathcal{R}$ be a bisimilarity on closed* HOcore *processes. The extension of $\mathcal{R}$ to open* HOcore *processes is defined by*

$$\mathcal{R}^{\circ} = \{(P, Q) : a(x_1). \cdots . a(x_n). P \, \mathcal{R} \, a(x_1). \cdots . a(x_n). Q\}$$

where $\mathrm{fv}(P) \cup \mathrm{fv}(Q) = \{x_1, \ldots, x_n\}$*, and a is fresh in P, Q.*

The simplest complete form of bisimilarity is $\sim^{\circ}_{\mathrm{IO}}$. Not only $\sim^{\circ}_{\mathrm{IO}}$ is the less demanding for proofs; it also has a straightforward proof of congruence. This is significant because congruence is a notoriously hard problem in bisimilarities for higher-order calculi. Before describing the proof of congruence for $\sim^{\circ}_{\mathrm{IO}}$, we first define an auxiliary up-to technique that will be useful later.

Definition 4.5. *A symmetric relation $\mathcal{R}$ on* HOcore *is an* open IO bisimulation up-to $\equiv$ *if $P \, \mathcal{R} \, Q$ implies:*

1. *if $P \xrightarrow{a(x)} P'$ then $Q \xrightarrow{a(x)} Q'$ and $P' \equiv\mathcal{R}\equiv Q'$;*
2. *if $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$ then $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$ with $P' \equiv\mathcal{R}\equiv Q'$ and $P'' \equiv\mathcal{R}\equiv Q''$;*
3. *if $P \equiv x \;\; P'$ then $Q \equiv x \;\; Q'$ and $P' \equiv\mathcal{R}\equiv Q'$.*

Lemma 4.1. *If $\mathcal{R}$ is an open IO bisimulation up-to $\equiv$ and $(P, Q) \in \mathcal{R}$ then $P \sim^{\circ}_{\mathrm{IO}} Q$.*

Proof. The proof proceeds by a standard diagram-chasing argument (as in, e.g., (Milner, 1989)): using Lemma 5.1 one shows that $\equiv\mathcal{R} \equiv$ is a $\sim^{\circ}_{\mathrm{IO}}$-bisimulation. □

We now give the congruence result for $\sim^{\circ}_{\mathrm{IO}}$.

Lemma 4.2 (Congruence of $\sim^{\circ}_{\mathrm{IO}}$)**.** *Let P_1, P_2 be open* HOcore *processes. $P_1 \sim^{\circ}_{\mathrm{IO}} P_2$ implies:*

1. *$a(x). P_1 \sim^{\circ}_{\mathrm{IO}} a(x). P_2$*

2. *$P_1 \;\; R \sim^{\mathrm{o}}_{\mathrm{IO}} P_2 \;\; R$, for every R*

3. *$\overline{a}\langle P_1\rangle \sim^{\mathrm{o}}_{\mathrm{IO}} \overline{a}\langle P_2\rangle$*

Proof. Items (1) and (3) are straightforward by showing the appropriate $\sim^{\mathrm{o}}_{\mathrm{IO}}$-bisimulations. We consider only (2). We show that, for every R, P_1, and P_2

$$\mathcal{S} = \{(P_1 \;\; R,\, P_2 \;\; R) \;:\; P_1 \sim^{\mathrm{o}}_{\mathrm{IO}} P_2\}$$

is a $\sim^{\mathrm{o}}_{\mathrm{IO}}$-bisimulation. We first suppose $P_1 \;\; R \xrightarrow{\alpha} P'$; we need to find a matching action from $P_2 \;\; R$. We proceed by case analysis on the rule used to infer α. There are two cases. In the first one $P_1 \xrightarrow{\alpha} P'_1$ and $P' = P'_1 \;\; R$ is inferred using rule Act1 (by α-conversion we can ensure that R respects the side condition of the rule). By definition of $\sim^{\mathrm{o}}_{\mathrm{IO}}$-bisimulation, $P_2 \xrightarrow{\alpha} P'_2$ with $P'_1 \sim^{\mathrm{o}}_{\mathrm{IO}} P'_2$. Using rule Act1 we infer that also $P_2 \;\; R \xrightarrow{\alpha} P'_2 \;\; R$. We conclude that $(P'_1 \;\; R, P'_2 \;\; R) \in \mathcal{S}$. The second case follows by an analogous argument and occurs when $R \xrightarrow{\alpha} R'$ so that $P' = P_1 \;\; R'$ by rule Act2.

The last thing to consider is when $P_1 \;\; R \equiv x \;\; P'$; we need to show that $P_2 \;\; R \equiv x \;\; Q'$ and that $(P', Q') \in \mathcal{S}$. We distinguish two cases, depending on the shape of P'. First, assume that $P' \equiv P'_1 \;\; R$, that is, x is a subprocess of P_1. Since $P_1 \sim^{\mathrm{o}}_{\mathrm{IO}} P_2$, then it must be that $P_2 \equiv x \;\; P'_2$ for some P'_2, with $P'_1 \sim^{\mathrm{o}}_{\mathrm{IO}} P'_2$. Taking $Q' \equiv P'_2 \;\; R$ we thus have $(P', Q') \in \mathcal{S}$. Second, assume x is a subprocess of R, and we have $P' \equiv P_1 \;\; R'$, we then take $Q' \equiv P_2 \;\; R'$. Since $\mathcal{S}$ is defined over every R, then $(P', Q') \in \mathcal{S}$. □

Lemma 4.3 ($\sim^{\mathrm{o}}_{\mathrm{IO}}$ is preserved by substitutions). *If $P \sim^{\mathrm{o}}_{\mathrm{IO}} Q$ then for all x and R, also $P\{{}^{R}\!/x\} \sim^{\mathrm{o}}_{\mathrm{IO}} Q\{{}^{R}\!/x\}$.*

Proof. We show that, for processes P, Q in which x is guarded,

$$\mathcal{R} = \{(P\{{}^{R}\!/x\} \;\; L,\, Q\{{}^{R}\!/x\} \;\; L) \;:\; P \sim^{\mathrm{o}}_{\mathrm{IO}} Q\}$$

is a $\sim^{\mathrm{o}}_{\mathrm{IO}}$-bisimulation up-to $\equiv$ (Definition 4.5). (This suffices, because of Lemma 4.1.) Consider a pair $(P\{{}^{R}\!/x\} \;\; L,\, Q\{{}^{R}\!/x\} \;\; L) \in \mathcal{R}$. We shall concentrate on the possible moves from $P\{{}^{R}\!/x\}$, say $P\{{}^{R}\!/x\} \xrightarrow{\alpha} P'$; transitions from L, if any, can be handled analogously. We proceed by case analysis on the rule used to infer α.

We only detail the case in which α is an input action $a(x)$ inferred using rule Inp; the case in which α is an output is similar (there may be a substitution on the label). Since x is guarded in P, using Lemma 3.2(2), there is P_1 such that $P \xrightarrow{a(x)} P_1$ and $P' = P_1\{{}^{R}\!/x\}$. By definition of $\sim^{\mathrm{o}}_{\mathrm{IO}}$-bisimulation, also $Q \xrightarrow{a(x)} Q_1$ with $P_1 \sim^{\mathrm{o}}_{\mathrm{IO}} Q_1$. Hence, by Lemma 3.2(1), $Q\{{}^{R}\!/x\} \xrightarrow{a(x)} Q_1\{{}^{R}\!/x\}$. It remains to show that $P_1\{{}^{R}\!/x\}$ and $Q_1\{{}^{R}\!/x\}$ can be rewritten into the form required in the bisimulation. Using Lemma 3.3(1), we have

$$P_1 \equiv P'_1 \;\; \prod^{n} x \quad \text{and} \quad Q_1 \equiv Q'_1 \;\; \prod^{m} x$$

for P_1', Q_1' in which x is guarded. As $P_1 \sim^{o}_{\mathrm{IO}} Q_1$, it must be $n = m$ and $P_1' \sim^{o}_{\mathrm{IO}} Q_1'$. Finally, using Lemmas 3.3(2) and 4.2 we have

$$P_1\{^R\!/x\} \equiv P_1'\{^R\!/x\} \prod^{n} R \quad \text{and} \quad Q_1\{^R\!/x\} \equiv Q_1'\{^R\!/x\} \prod^{n} R$$

which closes up the bisimulation up-to $\equiv$. □

The most striking property of $\sim^{o}_{\mathrm{IO}}$ is its decidability. In contrast with the other bisimilarities, in $\sim^{o}_{\mathrm{IO}}$ the size of processes always decreases during the bisimulation game. This is because $\sim^{o}_{\mathrm{IO}}$ is an open relation and does not have a clause for τ transitions, hence process copying never occurs.

Lemma 4.4. *Relation $\sim^{o}_{\mathrm{IO}}$ is decidable.*

Next we show that $\sim^{o}_{\mathrm{IO}}$ is also a τ bisimulation. This will allow us to prove that $\sim^{o}_{\mathrm{IO}}$ coincides with other bisimilarities, and to transfer to them its properties, in particular congruence and decidability.

Lemma 4.5. *Relation $\sim^{o}_{\mathrm{IO}}$ is a τ-bisimulation.*

Proof. Suppose $(P, Q) \in \sim^{o}_{\mathrm{IO}}$ and $P \xrightarrow{\tau} P'$; we have to find a matching transition $Q \xrightarrow{\tau} Q'$. Action τ was inferred using either rule TAU1 or TAU2, so we have two cases. We consider only the first one as the second is analogous. If rule Tau1 was used, then we can decompose P's transition into an output $P \xrightarrow{\overline{a}\langle R\rangle} P_1$ followed by an input $P_1 \xrightarrow{a(x)} P_2$, with $P' = P_2\{^R\!/x\}$ (that is, the structure of P is $P \equiv \overline{a}\langle R\rangle \parallel a(x).\, P_2$). By definition of $\sim^{o}_{\mathrm{IO}}$, Q is capable of matching these two transitions, and the final derivative is a process Q_2 with $Q_2 \sim^{o}_{\mathrm{IO}} P_2$. Further, as HOcore has no output prefixes (i.e., it is an asynchronous calculus) the two transitions from Q can be combined into a τ-transition. Finally, since $\sim^{o}_{\mathrm{IO}}$ is preserved by substitutions (Lemma 4.3), we can use rule Tau1 to derive a process $Q' = Q_2\{^R\!/x\}$ that matches the τ-transition from P, with $(P', Q') \in \sim^{o}_{\mathrm{IO}}$. □

Corollary 4.1. *$\sim_{\mathrm{HO}}$ and $\sim^{o}_{\mathrm{IO}}$ coincide.*

Proof. The hard implication is the one right to left ($\sim^{o}_{\mathrm{IO}}$ implies $\sim_{\mathrm{HO}}$). One shows that for closed P, Q, $\sim^{o}_{\mathrm{IO}}$ is a $\sim_{\mathrm{HO}}$-bisimulation. Suppose $(P_1, P_2) \in \sim^{o}_{\mathrm{IO}}$ and $P_1 \xrightarrow{\alpha} Q_1$; we need to find a matching transition $P_2 \xrightarrow{\alpha} Q_2$. We consider three cases, one for each form that α can take. Case $\alpha = \overline{a}\langle R\rangle$ is immediate as both $\sim_{\mathrm{HO}}$ and $\sim^{o}_{\mathrm{IO}}$ are higher-order output bisimulations. As for cases $\alpha = a(x)$ and $\alpha = \tau$, the desired transition can be easily obtained using Lemmas 4.3 ($\sim^{o}_{\mathrm{IO}}$ is preserved by substitutions) and 4.5 ($\sim^{o}_{\mathrm{IO}}$ is a τ-bisimulation), respectively. □

We thus infer that $\sim_{\mathrm{HO}}$ is a congruence relation. A direct proof of this result (by exhibiting an appropriate bisimulation), in particular congruence for parallel composition, would have been harder. Congruence of higher-order bisimilarity is usually proved by appealing to, and adapting, Howe's method for the λ-calculus (Howe, 1996).

We now move to the relationship between $\sim_{\mathrm{HO}}$, $\sim^{\mathrm{o}}_{\mathrm{NOR}}$, and $\sim_{\mathrm{CON}}$. We begin by establishing a few properties of normal bisimulation.

Lemma 4.6. *If $m.P_1 \quad P \sim^{\mathrm{o}}_{\mathrm{NOR}} m.Q_1 \quad Q$, for some fresh m, then we have $P_1 \sim^{\mathrm{o}}_{\mathrm{NOR}} Q_1$ and $P \sim^{\mathrm{o}}_{\mathrm{NOR}} Q$.*

Proof. We show that, for any fresh names $m_1, \ldots,$

$$\mathcal{S} = \bigcup_{j=1}^{\infty}\{(P,\, Q) \;:\; P \quad \prod_{k\in 1..j} m_k.\, P_k \sim^{\mathrm{o}}_{\mathrm{NOR}} Q \quad \prod_{k\in 1..j} m_k.\, Q_k\}$$

$$\mathcal{S}_1 = \bigcup_{j=1}^{\infty}\{(P_1,\, Q_1) \;:\; P \quad \prod_{k\in 1..j} m_k.\, P_k \sim^{\mathrm{o}}_{\mathrm{NOR}} Q \quad \prod_{k\in 1..j} m_k.\, Q_k\}$$

are $\sim^{\mathrm{o}}_{\mathrm{NOR}}$-bisimulations.

We start with $\mathcal{S}$. Suppose $(P, Q) \in \mathcal{S}$ and that $P \xrightarrow{\alpha} P'$; we need to show a matching action from Q. We have different cases depending on the shape of α. We consider only the case $\alpha = \overline{a}\langle P''\rangle$, the others being simpler. By Act1 we have

$$P \quad \prod_{k\in 1..j} m_k.\, P_k \xrightarrow{\overline{a}\langle P''\rangle} P' \quad \prod_{k\in 1..j} m_k.\, P_k.$$

Since $P \quad \prod_{k\in 1..j} m_k.\, P_k \sim^{\mathrm{o}}_{\mathrm{NOR}} Q \quad \prod_{k\in 1..j} m_k.\, Q_k$, there should exist a Q^* such that

$$Q \quad \prod_{k\in 1..j} m_k.\, Q_k \xrightarrow{\overline{a}\langle Q''\rangle} Q^*\,,$$

with $P' \quad \prod_{k\in 1..j} m_k.\, P_k \quad m''.\, P'' \sim^{\mathrm{o}}_{\mathrm{NOR}} Q^* \quad m''.\, Q''$. As $m_1, \ldots, m_j$ are fresh, they do not occur in P, thus $\overline{a}\langle P''\rangle$ does not mention them. For the same reason, there cannot be any communication between $\prod_{k\in 1..j} m_k.\, Q_k$ and Q; so, we infer that the only possible transition is

$$Q \quad \prod_{k\in 1..j} m_k.\, Q_k \xrightarrow{\overline{a}\langle Q''\rangle} Q^* = Q' \quad \prod_{k\in 1..j} m_k.\, Q_k,$$

applying rule Act1 to $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$. Since $P' \quad \prod_{k\in 1..j} m_k.\, P_k \quad m''.\, P'' \sim^{\mathrm{o}}_{\mathrm{NOR}} Q' \quad \prod_{k\in 1..j} m_k.\, Q_k \quad m''.\, Q''$, we have $(P', Q') \in \mathcal{S}$ as needed.

Before considering $\mathcal{S}_1$, we find it useful to detail a procedure for *consuming* $\sim^{\mathrm{o}}_{\mathrm{NOR}}$-bisimilar processes.

Given a process P, let $o(P)$ denote the number of output actions in P. Let $m(P) = \#(P) + o(P)$ be the measure that considers both the (lexical) size of P and the number of output actions in it. Consider now two $\sim^{\mathrm{o}}_{\mathrm{NOR}}$-bisimilar processes P and Q. The procedure consists in consuming one of them by performing its actions completely; the other process can match these actions (as it is $\sim^{o}_{\mathrm{NOR}}$-bisimilar) and will be consumed as well. We will show that $m(P)$ decreases at each step of the bisimulation game; at the end, we will obtain processes P_n and Q_n with $m(P_n) = m(Q_n) = 0$.

To illustrate the procedure, suppose, w.l.o.g., process P has the following shape:

$$P \equiv \prod_{h \in 1..t} x_h \ \prod_{i \in 1..k} a_i(x_i).\, P_i \ \prod_{j \in 1..l} \overline{b_j}\langle P_j \rangle$$

where we have t top-level variables, k input actions, and l output actions. We use a_i and b_j for channels in input and output actions, respectively. The first step is to remove top-level variables; this relies on the fact $\sim^{o}_{\mathrm{NOR}}$ is an open bisimilarity. One thus obtains processes P_1 and Q_1 with only input and output actions, and both $m(P_1) < m(P)$ and $m(Q_1) < m(Q)$ hold. As a second step, the procedure exercises every output action in P_1. By definition of $\sim^{o}_{\mathrm{NOR}}$, Q_1 should be able to match those actions. Call the resulting processes P_2 and Q_2. Recall that when an output $\overline{a}\langle P_j \rangle$ is consumed in the bisimulation game, process $m_j.\, P_j$ is added in parallel. Thus since $\#(\overline{a}\langle P \rangle_j) = \#(m_j.\, P_j)$ and the number of outputs decreases, measure m decreases as well. More precisely, one has that

$$P_2 \equiv \prod_{i \in 1..k} a_i(x_i).\, P_i \ \prod_{j \in 1..l} m_j.\, P_j$$

where m_j stands for a fresh name. Then, one has to consider the $k + l$ input actions in each process; their consumption proceeds as expected. One obtains processes P_3 and Q_3 that are bisimilar, with strictly decreasing measures for both processes. The procedure concludes by iterating the above steps on P_3 and Q_3. In fact, we have shown that at each step measure m strictly decreases; this guarantees that eventually one will reach processes P_n and Q_n, with $m(P_n) = m(Q_n) = 0$ as desired.

With the above procedure showing $\mathcal{S}_1$ is a bisimulation is straightforward. Take the $\sim^{o}_{\mathrm{NOR}}$-bisimilar processes $P \quad \prod_{k \in 1..j} m_k.\, P_k$ and $Q \quad \prod_{k \in 1..j} m_k.\, Q_k$. Using the procedure above over P and Q we obtain $\prod_{k \in 1..j} m_k.\, P_k \sim^{o}_{\mathrm{NOR}} \prod_{k \in 1..j} m_k.\, Q_k$. This is because fresh names $m_1, \ldots, m_j$ do not occur in P and Q, and hence they do not intervene in P and Q's consumption. Similarly, we can consume $\prod_{k' \in 2..j} m_{k'}.\, P_{k'}$ (i.e. all the components excepting $m_1.\, P_1$) and the corresponding $\prod_{k' \in 2..j} m_{k'}.\, Q_{k'}$. We thus end up with $m_1.\, P_1 \sim^{o}_{\mathrm{NOR}} m_1.\, Q_1$, and we observe that the only possible action on each side is the input on m_1, which can be trivially matched by the other. We then infer that $(P_1, Q_1) \in \mathcal{S}_1$, as desired. □

Lemma 4.7. *$\sim_{\mathrm{HO}}$ implies $\sim_{\mathrm{CON}}$.*

Proof. We suppose $(P, Q) \in \sim_{\mathrm{HO}}$ and $P \xrightarrow{\alpha} P'$; we need to show a matching action from Q. We proceed by case analysis on the form α can take. The only interesting case is when α is a higher-order output; the remaining clauses are the same in both relations. By definition of $\sim_{\mathrm{HO}}$, if $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$ then $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$, with both $P'' \sim_{\mathrm{HO}} Q''$ and $P' \sim_{\mathrm{HO}} Q'$. We need to show that, for every S such that $\mathrm{fv}(S) = \{x\}$, $S\{P''/x\} \parallel P' \sim_{\mathrm{HO}} S\{Q''/x\} \parallel Q'$; this follows from $P'' \sim_{\mathrm{HO}} Q''$ and $P' \sim_{\mathrm{HO}} Q'$ and the fact that $\sim_{\mathrm{HO}}$ is both a congruence and preserved by substitutions. □

Lemma 4.8. $\sim_{\mathrm{CON}}$ *implies* $\sim_{\mathrm{NOR}}$.

Proof. Straightforward by showing an appropriate bisimulation. The result is immediate by noticing that (i) both relations are τ-bisimulations, and that (ii) the input and output clauses of $\sim_{\mathrm{NOR}}$ are instances of those of $\sim_{\mathrm{CON}}$. In the output case, by selecting a process $S = m.\,x$ (with m fresh) one obtains the desired form for the clause. The input clause is similar, and follows from the definition of closed bisimulation, which holds for every closed process R; in particular, it also holds for $R = \overline{m}\langle \mathbf{0} \rangle$ (with m fresh) as required by the clause of $\sim_{\mathrm{NOR}}$. □

Lemma 4.9. $\sim_{\mathrm{NOR}}$ *implies* $\sim^{\circ}_{\mathrm{NOR}}$.

Proof. We consider open processes, and as such, in what follows we consider the extension of $\sim_{\mathrm{NOR}}$ to open processes as in Definition 4.3, which we denote $\sim^{\star}_{\mathrm{NOR}}$. Notice that since $\sim_{\mathrm{NOR}}$ is an input normal bisimulation (Definition 4.2(2)), $\sim^{\star}_{\mathrm{NOR}}$ can be equivalently defined as

$$P \sim^{\star}_{\mathrm{NOR}} Q \text{ iff } P\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\} \sim_{\mathrm{NOR}} Q\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\}$$

where $\{x_1, \dots, x_n\} = \mathrm{fv}(P) \cup \mathrm{fv}(Q)$ and $m_1, \dots, m_n$ are fresh names. We will show that $\sim^{\star}_{\mathrm{NOR}}$ is an open normal bisimulation.

We first suppose $P \sim^{\star}_{\mathrm{NOR}} Q$ and $P \xrightarrow{\alpha} P'$; we need to find a matching transition $Q \xrightarrow{\alpha} Q'$. We perform a case analysis on the shape α can take. In all cases, one uses the definition of $\sim^{\star}_{\mathrm{NOR}}$ to show that the desired transition actually takes place. Next, we only detail the case in which $\alpha = a(x)$, so we suppose $P \xrightarrow{a(x)} P'$. Now, if $P \xrightarrow{a(x)} P'$ then also $P\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\}$ must be capable of performing such an action, so that

$$P\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\} \xrightarrow{a(x)} P'\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\}$$

exists. In turn, by definition of $\sim_{\mathrm{NOR}}$, such an action guarantees that there exists a Q' that matches that input action, i.e.

$$Q\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\} \xrightarrow{a(x)} Q'\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\};$$

with

$$P'\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\}\{\overline{m}\langle \mathbf{0} \rangle/x\} \sim_{\mathrm{NOR}} Q'\{\overline{m_1}\langle \mathbf{0} \rangle/x_1, \dots, \overline{m_n}\langle \mathbf{0} \rangle/x_n\}\{\overline{m}\langle \mathbf{0} \rangle/x\}.$$

Again, by using the definition the $\sim^{\star}_{\mathrm{NOR}}$ on the above facts, it is easy to see that there is a Q' such that $Q \xrightarrow{a(x)} Q'$ and $P' \sim^{\star}_{\mathrm{NOR}} Q'$, and we are done.

The last thing to consider are those variables in evaluation context in the open processes. This is straightforward by noting that by definition of $\sim^{\star}_{\mathrm{NOR}}$, all such variables have been closed with a trigger. So, suppose $P \sim^{\star}_{\mathrm{NOR}} Q$ and

$$P\{\overline{m_1}\langle\mathbf{0}\rangle/x_1, \ldots, \overline{m_n}\langle\mathbf{0}\rangle/x\} \equiv \overline{m_1}\langle\mathbf{0}\rangle \;\; P'\{\overline{m_1}\langle\mathbf{0}\rangle/x_1, \ldots, \overline{m_n}\langle\mathbf{0}\rangle/x_n\}$$

where m_1 is fresh. We need to show that Q has a similar structure, i.e. that $Q \equiv x \;\; Q'$, with $P' \sim^{\star}_{\mathrm{NOR}} Q'$. P can perform an output action on m_1, thus evolving to $P'\{\overline{m_1}\langle\mathbf{0}\rangle/x_1, \ldots, \overline{m_n}\langle\mathbf{0}\rangle/x_n\}$. By definition of $\sim^{\star}_{\mathrm{NOR}}$, Q can match this action, and evolves to some process Q^*, with $m'.\mathbf{0} \;\; P'\{\overline{m_1}\langle\mathbf{0}\rangle/x_1, \ldots, \overline{m_n}\langle\mathbf{0}\rangle/x_n\} \sim^{\star}_{\mathrm{NOR}} m'.\mathbf{0} \;\; Q^*$, where m' is a fresh name (obtained from the definition of $\sim^{\star}_{\mathrm{NOR}}$ for output actions). The input on m' can be trivially consumed on both sides, and one has $P'\{\overline{m_1}\langle\mathbf{0}\rangle/x_1, \ldots, \overline{m_n}\langle\mathbf{0}\rangle/x_n\} \sim^{\star}_{\mathrm{NOR}} Q^*$. At this point, since m_1 is a fresh name, we know that Q involves a variable in evaluation context. Furthermore, since there is a correspondence between P' and Q^*, they should involve substitutions in the very same fresh names. More precisely, we have that there should be a Q' such that

$$Q \equiv \overline{m_1}\langle\mathbf{0}\rangle \;\; Q'\{\overline{m_1}\langle\mathbf{0}\rangle/x_1, \ldots, \overline{m_n}\langle\mathbf{0}\rangle/x_n\} = \overline{m_1}\langle\mathbf{0}\rangle \;\; Q^*$$

as desired. □

Lemma 4.10. *$\sim^{\circ}_{\mathrm{NOR}}$ implies $\sim^{\circ}_{\mathrm{IO}}$.*

Proof. The only difference between the bisimilarities is their output clause: they are both open bisimulations. We analyze directly the case for output action. Suppose $P \sim^{\circ}_{\mathrm{NOR}} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$; we need to show a matching action from Q. By definition of $\sim^{\circ}_{\mathrm{NOR}}$, if $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ then also $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$, with $m.P'' \;\; P' \sim^{\circ}_{\mathrm{NOR}} m.Q'' \;\; Q'$. Using this and Lemma 4.6 we conclude that $P'' \sim^{\circ}_{\mathrm{NOR}} Q''$ and $P' \sim^{\circ}_{\mathrm{NOR}} Q'$. □

Lemma 4.11. *In* HOcore, *relations $\sim_{\mathrm{HO}}$, $\sim^{\circ}_{\mathrm{NOR}}$ and $\sim_{\mathrm{CON}}$ coincide.*

Proof. This is an immediate consequence of previous results. In fact, we have proved (on open processes) the following implications:

1. $\sim^{\circ}_{\mathrm{IO}}$ implies $\sim_{\mathrm{HO}}$ (Corollary 4.1).

2. $\sim_{\mathrm{HO}}$ implies $\sim_{\mathrm{CON}}$ (Lemma 4.7);

3. $\sim_{\mathrm{CON}}$ implies $\sim_{\mathrm{NOR}}$ (Lemma 4.8);

4. $\sim_{\mathrm{NOR}}$ implies $\sim^{\circ}_{\mathrm{NOR}}$ (Lemma 4.9);

5. $\sim^{\circ}_{\mathrm{NOR}}$ implies $\sim^{\circ}_{\mathrm{IO}}$ (Lemma 4.10).

□

We then extend the result to all complete combinations of the HOcore bisimulation clauses (Definitions 4.1 and 4.2).

Theorem 4.1. *All complete combinations of the* HOcore *bisimulation clauses coincide, and are decidable.*

Proof. In Lemma 4.11 we have proved that the least demanding combination ($\sim^{\mathrm{o}}_{\mathrm{IO}}$) coincides with the most demanding ones ($\sim_{\mathrm{HO}}$ and $\sim_{\mathrm{CON}}$). Decidability then follows from Lemma 4.4. □

We find this "collapsing" of bisimilarities in HOcore significant; the only similar result we are aware of is by Cao (2006), who showed that strong context bisimulation and strong normal bisimulation coincide in higher-order π-calculus.

4.2 Barbed Congruence and Asynchronous Equivalences

We now show that the labeled bisimilarities of Section 4.1 coincide with *barbed congruence*, the form of contextual equivalence used in concurrency to justify bisimulation-like relations. Below we use *reduction-closed* barbed congruence (Honda and Yoshida, 1995; Sangiorgi and Walker, 2001), as this makes some technical details simpler; however the results also hold for ordinary barbed congruence (Milner and Sangiorgi, 1992). It is worth recalling that the main difference between reduction-closed and ordinary barbed congruence is quantification over contexts (see (2) in Definition 4.6 below). More importantly, we consider the *asynchronous* version of barbed congruence, where barbs are only produced by output messages; in synchronous barbed congruence inputs may also contribute. We use the asynchronous version for two reasons. First, asynchronous barbed congruence is a weaker relation, which makes the results stronger (they imply the corresponding results for the synchronous relation). Second, asynchronous barbed congruence is more natural in HOcore because it is an asynchronous calculus — it has no output prefix.

Note also that the labeled bisimilarities of Section 4.1 have been defined in the synchronous style. In an *asynchronous labeled bisimilarity* (see, e.g., (Amadio et al., 1998)) the input clause is weakened so as to allow, in certain conditions, an input action to be matched also by a τ-action. For instance, input normal bisimulation (Definition 4.2(2)) would become:

- if $P \xrightarrow{a(x)} P'$ then, for some fresh name m,

  1. either $Q \xrightarrow{a(x)} Q'$ and $P'\{\overline{m}\langle\mathbf{0}\rangle/x\} \mathcal{R} Q'\{\overline{m}\langle\mathbf{0}\rangle/x\}$;
  2. or $Q \xrightarrow{\tau} Q'$ and $P'\{\overline{m}\langle\mathbf{0}\rangle/x\} \mathcal{R} Q' \;\; \overline{a}\langle\overline{m}\langle\mathbf{0}\rangle\rangle$.

We now define asynchronous barbed congruence. We write $P \downarrow_{\overline{a}}$ (resp. $P \downarrow_{a}$) if P can perform an output (resp. input) transition at a.

Definition 4.6. Asynchronous barbed congruence, $\simeq$, *is the largest symmetric relation on closed processes that is*

1. *a τ-bisimulation (Definition 4.1(1));*

2. *context-closed (i.e., $P \simeq Q$ implies $C[P] \simeq C[Q]$, for all closed contexts $C[\cdot]$);*

3. *barb preserving (i.e., if $P \simeq Q$ and $P \downarrow_{\overline{a}}$, then also $Q \downarrow_{\overline{a}}$).*

In *synchronous* barbed congruence, input barbs $P \downarrow_a$ are also observable.

Lemma 4.12. *Asynchronous barbed congruence coincides with normal bisimilarity.*

Proof. We first show that $\sim_{\text{NOR}}$ implies $\simeq$, and then its converse, which is harder. The relation $\sim_{\text{NOR}}$ satisfies the conditions in Definition 4.6 as follows. First, both relations are τ-bisimulations so condition (1) above trivially holds. Second, the context-closure condition follows from the fact that $\sim_{\text{NOR}}$ is a congruence. Finally, the barb-preserving condition is seen to hold by definition of $\sim_{\text{NOR}}$: having $P \sim_{\text{NOR}} Q$ implies that an output action of P on a has to be matched by an output action of Q on a; hence, we have that if $P \downarrow_{\overline{a}}$, then also $Q \downarrow_{\overline{a}}$.

Now the converse. We show that relation $\simeq$ satisfies the three conditions for $\sim_{\text{NOR}}$ in Definition 4.3. Suppose $P \simeq Q$ and $P \xrightarrow{\alpha} P'$; we have to show a matching transition $Q \xrightarrow{\alpha} Q'$. We proceed by a case analysis on the form α can take.

Case $\alpha = \tau$ Since by definition $\simeq$ is a τ-bisimulation, then there is a Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \simeq Q'$ and we are done.

Case $\alpha = \overline{a}\langle P'' \rangle$ We have $P \xrightarrow{\overline{a}\langle P''\rangle} P'$: it can be shown that $\simeq$ is an output normal bisimulation by showing a suitable context. Let $C_o^a[\cdot]$ be the context

$$C_o^a[\cdot] = [\cdot] \;\; a(x).(m.x \;\; \overline{n} \;\; n.\mathbf{0})$$

where m, n are fresh names. We then have $C_o^a[P] \xrightarrow{\tau} P_1$ with $P_1 \downarrow_{\overline{n}}$. Indeed, we have $P_1 \equiv P' \;\; m.P'' \;\; \overline{n} \;\; n.\mathbf{0}$. By definition of $\simeq$, we have also $C_o^a[Q] \xrightarrow{\tau} Q_1$ and necessarily, $Q_1 \downarrow_{\overline{n}}$. Since n is a fresh name, we infer that Q also has an output on a, such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$ and hence $Q_1 \equiv Q' \;\; m.Q'' \;\; \overline{n} \;\; n.\mathbf{0}$. Note that (P_1, Q_1) is in $\simeq$. They can consume the actions on n; since it is a fresh name, only the corresponding τ action of Q_1 can match it. As a result, both processes evolve to processes $P' \;\; m.P''$ and $Q' \;\; m.Q''$ that are still in $\simeq$. We then conclude that $\simeq$ is an output normal bisimulation.

Case $\alpha = a(x)$ We have $P \xrightarrow{a(x)} P'$. Again, to show that $\simeq$ is an input normal bisimulation, we define a suitable context. Here, the asynchronous nature of HOcore (more precisely, the lack of output prefixes, which prevents the control of output actions by modifying their continuation) and the input clause for $\sim_{\mathrm{NOR}}$ it induces (reported above) result in a more involved definition of these contexts. Notice that simply defining a context with an output action on a so as to force synchronization with the input action does not work here: process P itself could contain other output actions on a that could synchronize with the input we are interested in, and as output actions have no continuation, it is not possible to put a fresh barb indicating it has been consumed. We overcome this difficulty by (i) renaming *every output* in P, so as to avoid the possibility of τ actions (including those coming from synchronizations on channels different from a), (ii) consuming the input action on a (by placing the renamed process in a suitable context) and then (iii) restoring the initial outputs.

We define a context for (i) above, i.e., to rename every output in a process so as to prevent τ actions. We start by denoting by $out(P)$ the multiset of names in output subject position in a process P. Further, let σ denote an injective relation between each occurrence of name in $out(P)$ and a fresh name. Let $C[\cdot]$ be the context

$$C[\cdot] = [\cdot] \quad \prod_{(b_i,c_i)\in\sigma} b_i(x).\, \overline{c_i}\langle x\rangle$$

which uniquely renames every output $\overline{b_i}\langle S_i\rangle$ as $\overline{c_i}\langle S_i\rangle$. (We shall use c_i ($i \in 1..n$) to denote the fresh names for the renamed outputs.) Consider now processes $C[P]$ and $C[Q]$: since the renaming is on fresh channels, it can be ensured that the τ action due to the renaming of one output on one process is matched by the other process with a τ action that corresponds to the renaming of the same output. At the end, after a series of n τ actions, $C[P]$ and $C[Q]$ become processes P_1 and Q_1 that have no τ actions arising from their subprocesses and that are in $\simeq$. At this point it is then possible to use a context for (ii), to capture the input action on a in P_1. Let $C_{\mathrm{i}}^{a}[\cdot]$ be the context

$$C_{\mathrm{i}}^{a}[\cdot] = [\cdot] \quad \overline{a}\langle \overline{m}\langle \mathbf{0}\rangle\rangle$$

where m is a fresh name. We then have

$$C_{\mathrm{i}}^{a}[P_1] \xrightarrow{\tau} \overline{c_1}\langle S_1\rangle \quad \cdots \quad \overline{c_n}\langle S_n\rangle \quad P_a\{\overline{m}\langle \mathbf{0}\rangle/x\} \quad P'' = P_2$$

which, by definition of $\simeq$, implies that also it must be the case that, for some process Q_2, $C_{\mathrm{i}}^{a}[Q_1] \xrightarrow{\tau} Q_2$. In fact, since there is a synchronization at a, it implies that Q_1 must have at least one input action on a. More precisely, we have

$$Q_2 \equiv \overline{c_1}\langle S_1\rangle \quad \cdots \quad \overline{c_n}\langle S_n\rangle \quad Q_a\{\overline{m}\langle \mathbf{0}\rangle/x\} \quad Q''.$$

We notice that P_2 and Q_2 are still in $\simeq$; it remains however to perform (iii), i.e. to revert the renaming made by $C[\cdot]$. To do so, we proceed analogously as before and define the context

$$C'[\cdot] = [\cdot] \;\; \prod_{(c_i,b_i)\in\sigma^{-1}} c_i(x).\,\overline{b_i}\langle x\rangle.$$

We have that each of $C'[P_2]$ and $C'[Q_2]$ produces n τ steps that exactly revert the renaming done by context $C[\cdot]$ above and lead to P_3 and Q_3, respectively. This renaming occur in lockstep (and no other τ action may be performed by Q_2), as each one removes a barb on a fresh name, thus the other process has to remove the same barb by doing the renaming. Hence, P_3 and Q_3 have the same output actions as the initial P and Q. To conclude, it is worth remarking that $C_{\mathrm{i}}^{a}[P] \xrightarrow{\tau} P_3$ in one step. Indeed, we have

$$C_{\mathrm{i}}^{a}[P] \equiv T \;\; a(x).\,P_a \;\; P'' \;\; \overline{a}\langle \overline{m}\langle \mathbf{0}\rangle\rangle \xrightarrow{\tau} P_3 = P'\{\overline{m}\langle \mathbf{0}\rangle/x\}$$

where T stands for all the output actions in P (on which the renaming took place). By doing and undoing the renaming on every output, we were able to infer that Q has an analogous structure

$$C_{\mathrm{i}}^{a}[Q] \equiv T' \;\; a(x).\,Q_a \;\; Q'' \;\; \overline{a}\langle \overline{m}\langle \mathbf{0}\rangle\rangle \xrightarrow{\tau} Q_3$$

where T' stands for all the output actions in Q. From the definition of $\simeq$ we know it is a τ-bisimulation, and then we have $P_3 \simeq Q_3$. Let $Q' = T' \;\; Q_a \;\; Q''$, we then have $Q \xrightarrow{a(x)} Q'$ and $Q_3 = Q'\{\overline{m}\langle \mathbf{0}\rangle/x\}$. To summarize, we have $P \xrightarrow{a(x)} P'$, $Q \xrightarrow{a(x)} Q'$, and $P'\{\overline{m}\langle \mathbf{0}\rangle/x\} \simeq Q'\{\overline{m}\langle \mathbf{0}\rangle/x\}$ with m fresh. Hence we conclude that $\simeq$ is an input normal bisimulation. □

Remark 4.3. *The proof relies on the fact that* HOcore *has no operators of recursion, choice, and restriction. In fact, recursion would break the proof as there could be an infinite number of outputs to rename. Also, choice would prevent the renaming to be reversible, and restriction would prevent the renaming using a context as some names may be hidden. The higher-order aspect of* HOcore *does not really play a role. The proof could indeed be adapted to CCS-like, or π-calculus-like, languages in which the same operators are missing.*

Corollary 4.2. *In* HOcore *asynchronous and synchronous barbed congruence coincide, and they also coincide with all complete combinations of the* HOcore *bisimulation clauses of Theorem 4.1.*

Further, Corollary 4.2 can be extended to include the asynchronous versions of the labeled bisimilarities in Section 4.1 (precisely, the *complete asynchronous combinations* of the HOcore bisimulation clauses; that is, complete combinations that make use of an asynchronous input clause as outlined before Definition 4.6). This holds because: (i) all proofs of Section 4.1 can be easily adapted to the corresponding asynchronous labeled bisimilarities; (ii) using standard reasoning for barbed congruences, one can show that asynchronous normal bisimilarity coincides with asynchronous barbed congruence; (iii) via Corollary 4.2 one can then relate the asynchronous labeled bisimilarities to the synchronous ones.

4.3 Axiomatization and Complexity

We have shown in the previous section that the main forms of bisimilarity for higher-order process calculi coincide in HOcore. We therefore simply call *bisimilarity* such a relation, and write it as $\sim$. Here we present a sound and complete axiomatization of bisimilarity. We do so by adapting to a higher-order setting results by Moller and Milner on unique decomposition of processes (Milner and Moller, 1993; Moller, 1989), and by Hirschkoff and Pous on an axiomatization for a fragment of (finite) CCS (Hirschkoff and Pous, 2007). We then exploit this axiomatization to derive complexity bounds for bisimilarity checking.

4.3.1 Axiomatization

Lemma 4.13. *$P \sim Q$ implies $\#(P) = \#(Q)$.*

Proof. Suppose, for a contradiction, that there exist P, Q such that $P \sim Q$ with $\#(P) < \#(Q)$ and choose a P with a minimal size. If Q has no transition enabled, then it must be $\mathbf{0}$, thus $\#(Q) = 0$, which is impossible as $\#(Q) > \#(P) \geq 0$.

We thus have $Q \xrightarrow{\alpha} Q'$, hence there is a P' such that $P \xrightarrow{\alpha} P'$ with $P' \sim Q'$. We consider two cases, depending on the shape of α (we do not consider τ actions, as such an action implies both an input and an output).

If α is an input action, we have $Q \xrightarrow{a(x)} Q'$, and since $P \sim Q$, then also $P \xrightarrow{a(x)} P'$. We then have that $\#(P') = \#(P) - 1$ and $\#(Q') = \#(Q) - 1$, so it follows that $\#(P') < \#(Q')$. Further, one has $\#(P') < \#(P)$, which contradicts the minimality hypothesis.

Now suppose α is an output action: we have $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$, and by definition of $\sim$, also that $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$ with both $P' \sim Q'$ and $P'' \sim Q''$. By the definition of size, we have that $\#(P') = \#(P) - (1 + \#(P''))$ and $\#(Q') = \#(Q) - (1 + \#(Q''))$. Notice that P'', Q'' are strict subterms of P and Q, respectively. If their size is not the same, we have a contradiction. Otherwise, we have $\#(P') < \#(Q')$ and also $\#(P') < \#(P)$, which is also a contradiction. □

Following (Milner and Moller, 1993; Moller, 1989) we prove a result of unique prime decomposition of processes.

Definition 4.7 (Prime decomposition). *A process P is* prime *if $P \not\sim \mathbf{0}$ and $P \sim P_1 \quad P_2$ imply $P_1 \sim \mathbf{0}$ or $P_2 \sim \mathbf{0}$. When $P \sim \prod_{i=1}^{n} P_i$ where each P_i is prime, we call $\prod_{i=1}^{n} P_i$ a* prime decomposition *of P.*

Proposition 4.1 (Cancellation). *For all P, Q, and R, if $P \quad R \sim Q \quad R$ then also $P \sim Q$.*

Proof. The proof, which proceeds by induction on $\#(P) + \#(Q) + \#(R)$, is a simple adaptation of the one in (Milner and Moller, 1993). □

Proposition 4.2 (Unique decomposition). *Any process P admits a prime decomposition $\prod_{i=1}^{n} P_i$ which is unique up to bisimilarity and permutation of indices (i.e., given two prime decompositions $\prod_{i=1}^{n} P_i$ and $\prod_{i=1}^{m} P'_i$, then $n = m$ and there is a permutation σ of $\{1, \ldots, n\}$ such that $P_i \sim P'_{\sigma(i)}$ for each $i \in \{1, \ldots, n\}$).*

Proof. The proof, also similar to the one in (Milner and Moller, 1993) with just variables to be considered in addition, uses Proposition 4.1. □

Both the key law for the axiomatization and the following results are inspired by similar ones by Hirschkoff and Pous (2007) for pure CCS. Using their terminology, we call *distribution law*, briefly (DIS), the axiom schema below (recall that $\prod_1^k Q$ denotes the parallel composition of k copies of Q).

$$a(x).\,(P \;\; \prod_1^{k-1} a(x).\,P) \;=\; \prod_1^{k} a(x).\,P \tag{DIS}$$

We then call *extended structural congruence*, written $\equiv_{\mathrm{E}}$, the extension of the structural congruence relation ($\equiv$, Definition 6.2) with the axiom schema (DIS). We write $P \rightsquigarrow Q$ when there are processes P' and Q' such that $P \equiv P'$, $Q' \equiv Q$ and Q' is obtained from P' by rewriting a subterm of P' using law (DIS) from left to right. Below we prove that $\equiv_{\mathrm{E}}$ provides an algebraic characterization of $\sim$ in HOcore. Establishing the soundness of $\equiv_{\mathrm{E}}$ is easy; below we discuss completeness.

Definition 4.8. *A process P is in* normal form *if it cannot be further simplified in the system $\equiv_{\mathrm{E}}$ by using $\rightsquigarrow$.*

Any process P has a normal form that is unique up to $\equiv$, and which will be denoted by $\mathrm{n}(P)$. Below A and B range over normal forms, and a process is said to be *non-trivial* if its size is not 0.

Lemma 4.14. *If $P \rightsquigarrow Q$, then $P \sim Q$. Also, for any P, $P \sim \mathrm{n}(P)$.*

Proof. The proof proceeds by showing that $(\rightsquigarrow \cup (\rightsquigarrow)^{-1} \cup \equiv)$ is a bisimulation (as $\sim^{\mathrm{o}}_{\mathrm{IO}}$, for instance). □

Lemma 4.15. *If $a(x).\,P \sim Q \;\; Q'$ with $Q, Q' \not\to \mathbf{0}$, then $a(x).\,P \sim \prod_1^k a(x).\,A$, where $k > 1$ and $a(x).\,A$ is in normal form.*

Proof. By Lemma 4.14, $a(x).\,P \sim \mathrm{n}(Q \;\; Q')$. Furthermore, by Proposition 4.2, we have that

$$\mathrm{n}(Q \;\; Q') \equiv \prod_{i \leq k} a_i(x_i).\,A_i \;\; \prod_{j \leq l} \overline{b_j}\langle B_j \rangle,$$

where the processes $a_i(x_i).\,A_i$ and $\overline{b_j}\langle B_j \rangle$ are in normal form and prime. Since the prefix $a(x)$ must be triggered to answer any challenge from the right-hand side, we have $a_i = a$, and $x_i = x$ (this can be obtained via α-conversion, but we can suppose that $a_i(x_i).\,A_i$ was already α-converted to the correct form), and we have $l = 0$ (there is no output in the prime decomposition). As there are at least two processes that are not $\mathbf{0}$, we have $k > 1$. To summarize:

$$a(x).\,P \sim \prod_{i \leq k} a(x).\,A_i.$$

After an input action on the right-hand side, we derive

$$P \sim A_i \;\; \prod_{l \neq i} a(x).\,A_l$$

for every $i \leq k$. In particular, when $i \neq j$, we have

$$P \sim A_i \;\; a(x).\,A_j \;\; \prod_{l \notin \{i,j\}} a(x).\,A_l \qquad P \sim A_j \;\; a(x).\,A_i \;\; \prod_{l \notin \{i,j\}} a(x).\,A_l$$

and, by Proposition 4.1, $A_i \;\; a(x).\,A_j \sim A_j \;\; a(x).\,A_i$. Since $a(x).\,A_i$ is prime and it has larger size than A_i (and any of its components), it should correspond in the prime decomposition to $a(x).\,A_j$, i.e. $a(x).\,A_i \sim a(x).\,A_j$. As this was shown for every $i \neq j$, we thus have $a(x).\,P \sim \prod_1^k a(x).\,A_1$ with $k > 1$ and $a(x).\,A_1$ in normal form. □

Lemma 4.16. *For A, B in normal form, if $A \sim B$ then $A \equiv B$.*

Proof. We show, simultaneously, the following two properties:

1. if A is a prefixed process in normal form, then A is prime;

2. for any B in normal form, $A \sim B$ implies $A \equiv B$.

We proceed by induction on n, for all A with $\#(A) = n$. The case $n = 0$ is immediate as the only process of this size is $\mathbf{0}$. Suppose that the property holds for all $i < n$, with $n \geq 1$. In the reasoning below, we exploit the characterization of $\sim$ as $\sim^{\mathrm{o}}_{\mathrm{IO}}$.

1. Process A is of the form $a(x).\,A'$. Suppose, as a contradiction, that A is not prime. Then we have $A \sim P_1 \;\; P_2$ with P_1 and P_2 non-trivial. By Lemma 4.15, then $A \sim \Pi_1^k\; a(x).\,B$ with $k > 1$ and $a(x).\,B$ in normal form. By consuming the prefix on the left-hand side, we have $A' \sim B \;\; \Pi_1^{k-1}\; a(x).\,B$. It follows by induction (using property (2)) that $A' \equiv B \;\; \Pi_1^{k-1}\; a(x).\,B$, and hence also $A \equiv a(x).\,(B \;\; \Pi_1^{k-1}\; a(x).\,B)$. This is impossible, as A is in normal form.

2. Suppose $A \sim B$. We proceed by case analysis on the structure of A.

   - Case $A = x$. We have that B should be the same variable, so $A \equiv B$ trivially.
   - Case $A = \overline{a}\langle P\rangle$. We have that $B = \overline{a}\langle P'\rangle$ with $P \sim P'$ by definition of $\sim^{\mathrm{o}}_{\mathrm{IO}}$. By the induction hypothesis, $P \equiv P'$, thus $\overline{a}\langle P\rangle \equiv \overline{a}\langle P'\rangle$.

- Case $A = a(x).\, A'$. We show by contradiction that $B = a(x).\, B'$. Assume $B = Q \;\; Q'$, then by Lemma 4.15, A is a parallel composition of at least two processes. But according to the first property, as A is prefixed, it is prime, a contradiction. We thus have $B = a(x).\, B'$ with $A' \sim^{\mathrm{o}}_{\mathrm{IO}} B'$. By induction this entails $A' \equiv B'$ and $A \equiv B$.
- Case $A = \prod_{i \leq k} P_i$ with $k > 1$ where no P_i has a parallel composition at top-level. We reason on the possible shape of the P_i.

  If there exists j such that $P_j = x$ then also $B \equiv x \;\; B'$. The thesis then follows by induction hypothesis on $\prod_{i \leq k, i \neq j} P_i$ and B'.

  If P_i is an output, B must contain an output on the same channel. The thesis then follows by applying the induction hypothesis twice, to the arguments and to the other parallel components.

  The last case is when $A \equiv \prod_{i \leq k} a_i(x_i).\, A_i$ with $k > 1$. We know by the induction hypothesis (property (1)) that each component $a_i(x_i).\, A_i$ is prime. Similarly, it must be $B \equiv \prod_{i \leq l} b_i(x_i).\, B_i$ with $b_i(x_i).\, B_i$ prime for all $i \leq l$. By Proposition 4.2 (unique decomposition), we infer $k = l$ and $a_i(x_i).\, A_i \sim b_i(x_i).\, B_i$ (up to a permutation of indices). Thus $a_i = b_i$ and $A_i \sim B_i$; then by induction $A_i \equiv B_i$ for all i, which finally implies $A \equiv B$.

□

The theorem below follows from Lemmas 4.14 and 4.16.

Theorem 4.1. *For any processes P and Q, we have $P \sim Q$ iff $\mathsf{n}(P) \equiv \mathsf{n}(Q)$.*

Corollary 4.3. *$\equiv_{\mathrm{E}}$ is a sound and complete axiomatization of bisimilarity in* HOcore.

4.3.2 Complexity of Bisimilarity Checking

To analyze the complexity of deciding whether two processes are bisimilar, one could apply the technique from (Dovier et al., 2004), and derive that bisimilarity is decidable in time which is linear in the size of the LTS for $\sim^{\mathrm{o}}_{\mathrm{IO}}$ (which avoids τ transitions). This LTS is however exponential in the size of the process. A more efficient solution exploits the axiomatization above: one can normalize processes and reduce bisimilarity to syntactic equivalence of normal forms.

For simplicity, we assume a process P is represented as an ordered tree (but we will transform it into a DAG during normalization). In the following, let us denote with $t[m_1, \ldots, m_k]$ the ordered tree with root labeled t and with (ordered) descendants $m_1, \ldots, m_k$. We write $t[]$ for a tree labeled t and without descendants (i.e., a leaf).

Definition 4.9 (Tree representation). *Let P be a* HOcore *process. Its associated ordered tree representation is labeled and defined inductively by*

- $\mathsf{Tree}(\mathbf{0}) = \mathbf{0}[]$
- $\mathsf{Tree}(x) = \mathsf{db}(x)[]$
- $\mathsf{Tree}(\overline{a}\langle Q\rangle) = \bar{a}[\mathsf{Tree}(Q)]$
- $\mathsf{Tree}(a(x).\, Q) = a[\mathsf{Tree}(Q)]$
- $\mathsf{Tree}(\prod_{i=1}^{n} P_i) = \prod_{i=1}^{n}[\mathsf{Tree}(P_1), \ldots, \mathsf{Tree}(P_n)]$

where db *is a function assigning De Bruijn indices De Bruijn (1972) to variables.*

We now describe the normalization steps. The first deals with parallel composition nodes: it removes all unnecessary **0** nodes, and relabels the nodes when the parallel composition has only one or no descendants.

Normalization step 1. *Let $\rightsquigarrow_{\mathrm{N1}}$ be a transformation rule over trees associated to* HOCORE *processes, defined by:*

1. $\prod_{i=1}^{n}[\mathsf{Tree}(P_1), \ldots, \mathsf{Tree}(P_n)] \rightsquigarrow_{\mathrm{N1}} \prod_{i=1}^{m}[\mathsf{Tree}(P_{\sigma(1)}), \ldots, \mathsf{Tree}(P_{\sigma(m)})]$, *where $m < n$ is the number of processes in $P_1, \ldots, P_n$ that are different from $\mathbf{0}$, and σ is a bijective function from $\{1, \ldots, m\}$ to $\{i \mid i \in \{1, \ldots, n\} \wedge P_i \neq \mathbf{0}\}$.*
2. $\prod_{i=1}^{0}[] \rightsquigarrow_{\mathrm{N1}} \mathbf{0}[]$
3. $\prod_{i=1}^{1}[\mathsf{Tree}(P_1)] \rightsquigarrow_{\mathrm{N1}} \mathsf{Tree}(P_1)$

After this first step, the tree is traversed bottom-up, applying the following normalization steps.

Normalization step 2. *Let $\rightsquigarrow_{\mathrm{N2}}$ be a transformation rule over trees associated to* HOCORE *processes, defined as follows. If the node is a parallel composition, sort all the children lexicographically. If n children are equal, leave just one and make n references to it.*

The last step applies `DIS` from left to right if possible.

Normalization step 3. *Let $\rightsquigarrow_{\mathrm{N3}}$ be a transformation rule over trees associated to* HOCORE *processes, defined by:*

$$a\left[\prod_{i=1}^{k+1}[\mathsf{Tree}(P), \mathsf{Tree}(a(x).\, P), \ldots, \mathsf{Tree}(a(x).\, P)]\right] \rightsquigarrow_{\mathrm{N3}} \prod_{j=1}^{k+1}[\mathsf{Tree}(a(x).\, P), \ldots, \mathsf{Tree}(a(x).\, P)]$$

where $\mathsf{Tree}(a(x).\, P)$ appears k times in the left-hand side, and $k+1$ times in the right-hand side.

Notice that applying DIS changes De Bruijn indices, but the first instance of P already has the correct indices, thus making n references to it produces the correct DAG. Note also that in the comparison between the different instances of P, care should be taken because of the De Bruijn indices. In fact, De Bruijn indices of all the instances of P but the first one are increased by one, since there is one more binding occurrence of x.

Lemma 4.17. *Let T_P, T_Q be two tree representations of processes P and Q (as in Definition 4.9), normalized according to normalization steps 1 and 2. Then $P \equiv Q$ iff $T_P = T_Q$.*

Proof. Immediate from Definitions 6.2 and 4.9, and from normalization steps 1 and 2. In particular, $\rightsquigarrow_{\text{N1}}$ corresponds to the elimination of all occurrences of $\mathbf{0}$ in parallel, and $\rightsquigarrow_{\text{N2}}$ corresponds to the choice of a representative process, up to associativity and commutativity. □

Lemma 4.18. *Let P, Q be processes and T_P, T_Q their tree representations normalized according to steps 1, 2 and 3. Then $P \sim Q$ iff $T_P = T_Q$.*

Proof. Immediate using Lemmas 4.17 and 4.14 ($P \rightsquigarrow Q$ implies $P \sim Q$). □

We now give a lemma on the cost of sorting the tree representation of a process. Given a process P, we define the size of its tree representation T_P to be the number of nodes of the tree, and denote it as **size**(P).

Lemma 4.19. *Consider n HOcore processes $P_1, \ldots, P_n$ and their tree representations $T_{P_1}, \ldots, T_{P_n}$. Their sorting has complexity $O(t \log n)$, where $t = \sum_{i \in 1..n} \mathbf{size}(P_i)$.*

Proof. Let us assume MergeSort as sorting algorithm. MergeSort sorts a list of elements by (i) splitting the list to be sorted in two; (ii) recursing on both sublists; and (iii) merging the sorted sublists. A merge function starts by comparing the first element of each list and then copies the smallest one to the new list. Comparing two elements P_i, P_j costs $\min(\mathbf{size}(P_i), \mathbf{size}(P_j))$. As each T_{P_i} is considered once (when it is copied to the new list) the cost of merging two lists is the sum of the size of their elements (the actual copying of an element has constant cost since it is just a pointer operation). Let us call a *slice* of MergeSort the juxtaposition of every recursive call at the same depth. In this way, e.g., the first slice considers the lists when recursion depth is equal to 1: the first two recursive calls, each one having half of the original list. In general, at every slice one finds a partition of the list in 2^d sublists, where d is the recursion depth. Each recursive call in every slice is going to merge two sublists, with a complexity of the sum of the sizes of these sublists. Summing everything, we get a cost of $t = \sum_{i \in 1..n} \mathbf{size}(P_i)$ at each recursion depth. Therefore, as there are $\log n$ different depths, the total complexity is $O(t \log n)$. □

Theorem 4.2. *Consider two* HOcore *processes P and Q. $P \sim Q$ can be decided in time $O(n^2 \log m)$ where $n = \max(\mathbf{size}(P), \mathbf{size}(Q))$ (i.e., the maximum number of nodes in the tree representations of P and Q) and m is the maximum branching factor in them (i.e., the maximum number of components in a parallel composition).*

Proof. Bisimilarity check proceeds as follows: first normalize the tree representations of the two processes, then check them for syntactic equality.

Normalization step 1 can be performed in time $O(n)$. In fact, a visit ($O(n)$) is enough to apply the first rule where needed, and a second visit is enough to apply the other two rules. Normalization step 2 should be applied if the node is a parallel composition. By Lemma 4.19 this can be done in $O(n \log m)$ for each parallel composition node. Normalization step 3 should be applied if the node is a prefix node. The check for applicability requires one comparison ($O(n)$) and the check that all the other components coincide (simply check that the subtrees have been merged by Normalization step 2: $O(n)$). Applying $\leadsto_{\mathrm{N3}}$ simply entails collapsing the trees ($O(n)$). Other nodes require no operations.

Thus the normalization for a single node can be done in $O(n \log m)$, and the whole normalization can be done in $O(n^2 \log m)$. □

4.4 Bisimilarity is Undecidable with Four Static Restrictions

If the restriction operator is added to HOcore, as in Plain CHOCS or Higher-Order π-calculus, then recursion can be encoded (Thomsen, 1990; Sangiorgi and Walker, 2001) and most of the results in Sections 4.1-4.3 would break. In particular, higher-order and context bisimilarities are different and both undecidable (Sangiorgi, 1992, 1996a).

We discuss here the addition of a limited form of restriction, which we call *static restriction*. These restrictions may not appear inside output messages: in any output $\overline{a}\langle P \rangle$, P is restriction-free. This limitation is important: it prevents for instance the above-mentioned encoding of recursion from being written. Static restrictions could also be defined as top-level restrictions since, by means of standard structural congruence laws, any static restriction can be pulled out at the top-level. Thus the processes would take the form $\boldsymbol{\nu} a_1 \ldots \boldsymbol{\nu} a_n\, P$, where $\boldsymbol{\nu} a_i$ indicates the restriction on the name a_i, and where restriction cannot appear inside P itself. The operational semantics—LTS and bisimilarities—are extended as expected. For instance, one would have bounded outputs as actions, as well as rules

$$\textsc{StRes}\quad \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathrm{fn}(\alpha)}{\nu z P \xrightarrow{\alpha} \nu z P'}$$

$$\textsc{StOpen}\quad \frac{P \xrightarrow{\overline{a}\langle R \rangle} P' \qquad z \in \mathrm{fn}(R)}{\nu z P \xrightarrow{\nu z\, \overline{a}\langle R \rangle} P'}$$

defining static restriction and extrusion of restricted names, respectively. Note that there is no need to define how a bounded output interacts with input as every τ transition takes place under the restrictions. Also, structural congruence (Definition 6.2) would be extended with the axioms for restriction $\nu z\, \nu w P \equiv \nu w\, \nu z P$ and $\nu z \mathbf{0} \equiv \mathbf{0}$. (In contrast, notice that we do not require the axiom: $\nu z(P \;\; Q) \equiv P \;\; (\nu z Q)$, where z does not occur in P.) We sometimes write $\nu a_1, \ldots, a_n$ to stand for $\nu a_1, \ldots, \nu a_n$.

We show that *four* static restrictions are enough to make undecidable any bisimilarity that has little more than a clause for τ-actions. For this, we reduce the Post correspondence problem (PCP) (Post, 1946; Sipser, 2005) to the bisimilarity of some processes. We call *complete τ-bisimilarity* any complete combination of the HOcore bisimulation clauses (as defined in Section 4.1) that includes the clause for τ actions (Definition 4.1(1)); the bisimilarity can even be asynchronous (Section 4.2).

Definition 4.10 (PCP). *An* instance *of PCP consists of an alphabet A containing at least two symbols, and a finite list $T_1, \ldots, T_n$ of tiles, where each tile is a pair of words over A. We use $T_i = (u_i, l_i)$ to denote a tile T_i with upper word u_i and lower word l_i. A solution to this instance is a non-empty sequence of indices $i_1, \ldots, i_k$, $1 \leq i_j \leq n$ $(j \in 1 \cdots k)$, such that $u_{i_1} \cdots u_{i_k} = l_{i_1} \cdots l_{i_k}$. The decision problem is then to determine whether such a solution exists or not.*

Having (static) restrictions, we refine the encoding of non-nested replications (Definition 3.5) and define it in the unguarded case:

$$[\![!P]\!]_! = \boldsymbol{\nu} c\,(Q_c \;\; \overline{c}\langle Q_c\rangle)$$

where $Q_c = c(x).(x \;\; \overline{c}\langle x\rangle \;\; P)$ and P is a HOcore process (i.e., it is restriction-free). In what follows we thus also consider *extended* HOcore processes, that include replication and for which structural congruence (Definition 6.2) is extended with the axiom $!P \equiv P \;\; !P$.

Lemma 4.1 (Correctness of $[\![\cdot]\!]_!$). *For each extended* HOcore *process P:*

- *if $[\![P]\!]_! \xrightarrow{\alpha} Q$ then $\exists P'$ such that either $P \xrightarrow{\alpha} P'$ and $[\![P']\!]_! = Q$ or $\alpha = \tau$, $P \equiv P'$ and $[\![P']\!]_! = Q$.*
- *$P \xrightarrow{\alpha} P'$ implies either $[\![P]\!]_! \xrightarrow{\alpha} [\![P']\!]_!$ or $[\![P]\!]_! \xrightarrow{\tau}\xrightarrow{\alpha} [\![P']\!]_!$.*

Proof. By transition induction. □

Now, $[\![!\mathbf{0}]\!]_!$ is a purely divergent process, as it can only make τ-transitions, indefinitely; it is written using only one static restriction. Given an instance of PCP we build a set of processes $P_1, \ldots, P_n$, one for each tile $T_1, \ldots, T_n$, and show that, for each i, P_i is bisimilar to $[\![!\mathbf{0}]\!]_!$ iff the instance of PCP has no solution ending with T_i. Thus PCP is solvable iff there exists j such that P_j is not bisimilar to $[\![!\mathbf{0}]\!]_!$.

Letters	$[a_1, P]_u = [a_2, P]_l = \overline{a}\langle P\rangle$
	$[a_2, P]_u = [a_1, P]_l = a(x).\,(x \quad P)$
Strings	$[a_i \cdot s, P]_w = [a_i, [s, P]_w]_w$
	$[\varepsilon, P]_w = P \qquad (\varepsilon \text{ is the empty word})$
Creators	$C_k = up(x).\,low(y).\,(\overline{up}\langle [u_k, x]_u\rangle \quad \overline{low}\langle [l_k, y]_l\rangle)$
Starters	$S_k = \overline{up}\langle [u_k, \overline{b}]_u\rangle \quad \overline{low}\langle [l_k, b.\,\overline{success}]_l\rangle$
Executor	$E = up(x).\,low(y).\,(x \quad y)$
System	$P_j = \boldsymbol{\nu} up\ \boldsymbol{\nu} low\ \boldsymbol{\nu} a\ \boldsymbol{\nu} b\ (\ S_j \quad !\prod_k C_k \quad E)$

Figure 4.1: Encoding of PCP into HOcore

The processes $P_1, \ldots, P_n$ execute in two distinct phases: first they build a possible solution of PCP, then they non-deterministically stop building the solution and execute it. If the chosen composition is a solution then a signal on a free channel *success* is sent, thus performing a visible action, which breaks bisimilarity with $[!\mathbf{0}]_!$.

The precise encoding of PCP into HOcore is shown in Figure 4.1, and described below. We consider an alphabet of two letters, a_1 and a_2. The upper and lower words of a tile are treated as separate *strings*, which are encoded letter by letter. The encoding of a letter is then a process whose continuation encodes the rest of the string, and varies depending on whether the letter occurs in the upper or in the lower word. We use a single channel to encode both letters: for the upper word, a_1 is encoded as $\overline{a}\langle P\rangle$ and a_2 as $a(x).\,(x \quad P)$, where P is the continuation and x does not occur in P; for the lower word the encodings are switched. In Figure 4.1, $[a_i, P]_w$ denotes the encoding of the letter a_i with continuation P, with $w = u$ if the encoding is on the upper word, $w = l$ otherwise. Hence, given a string $s = a_i \cdot s'$, its encoding $[s, P]_w$ is $[a_i, [s', P]_w]_w$, i.e., the first letter with the encoding of the rest as continuation. Notice that the encoding of an a_i in the upper word can synchronize only with the encoding of a_i for the lower word.

The whole system P_j is composed by a (replicated) *creator* C_k for each tile T_k, a *starter* S_j that launches the building of a tile composition ending with (u_j, l_j), and an *executor* E. The starter makes the computation begin; creators non-deterministically add their tile to the beginning of the composition. Also non-deterministically, the executor blocks the building of the composition and starts its execution. This proceeds if no difference is found: if both strings end at the same character, then synchronization on channel b can be performed, which in turn, makes action $\overline{success}$ visible. Notice that without synchronizing on b, action $\overline{success}$ could be visible even in the case in which one of the strings is a prefix of the other one.

The encoding of replication requires another restriction, thus P_j has five restrictions. However, names low and a are used in different phases; thus choosing $low = a$ does not create interferences, and four restrictions are enough.

Theorem 4.3. *Given an instance of PCP and one of its tiles T_j, there is a solution of the instance of PCP ending with T_j iff P_j is not bisimilar to* $[!\mathbf{0}]_!$ *according to any complete τ-bisimilarity.*

Proof. We start by proving the left to right implication. Note that $[!\mathbf{0}]_!$ has a unique possible computation, that is infinite and includes only τ actions. Let $T_{i_1}, \ldots, T_{i_m}$ be a solution of the instance of the PCP problem such that $T_{i_m} = T_j$. Then P_j can perform the computation described below, which contains the action $\overline{success}$, thus it is not bisimilar to $[!\mathbf{0}]_!$. The computation is as follows:

1. $P_j \xrightarrow{\tau}{}^{*} \nu up, a, b.\ S_j \ \ \prod_{h=1..m-1} C_{i_h} \ \ \prod C \ \ !\prod_k C_k \ \ E = P_1'$, by replication unfolding (the $\prod C$ is the parallel composition of the creators that have been replicated and will not be used);

2. $P_1' \xrightarrow{\tau}{}^{*} \nu up, a, b.\ \overline{up}\langle[u, \overline{b}]_u\rangle \ \ \overline{a}\langle[l, b.\overline{success}]_l\rangle \ \ \prod C \ \ !\prod_k C_k \ \ E = P_2'$, where (u, l) is the solution of the instance of the PCP problem, by making the starter S_j interact with the creators $C_{i_{m-1}} .. C_{i_1}$;

3. $P_2' \xrightarrow{\tau}\xrightarrow{\tau} \nu up, a, b. \prod C \ \ !\prod_k C_k \ \ [u, \overline{b}]_u \ \ [l, b.\overline{success}]_l = P_3'$, by making the starter interact with the executor (note that as every creator starts by an input on up, none of them my be triggered by messages on $\overline{a}$);

4. $P_3' \xrightarrow{\tau}{}^{*} \nu up, a, b. \prod C \ \ !\prod_k C_k \ \ \overline{b} \ \ b.\overline{success} = P_4'$, by executing the encodings of the two strings, exploiting the fact that they are equal;

5. $P_4' \xrightarrow{\tau} \nu up, a, b. \prod C \ \ !\prod_k C_k \ \ \overline{success} = P_5'$, by synchronizing on b;

6. $P_5' \xrightarrow{\overline{success}} \nu up, a, b. \prod C \ \ !\prod_k C_k$.

For the other implication, first notice that all the computations of P_j are infinite since one can always unfold recursion, and action $\overline{success}$ is the only possible visible action. Thus the only possibility for having P_j not bisimilar to $[!\mathbf{0}]_!$ is that P_j has a computation executing $\overline{success}$. The only computations that may produce $\overline{success}$ are structured as follows: they build two strings by concatenating the tiles, and then they execute them. One can prove by induction on the minimum length of the strings that if the two strings are different then either their execution gets stuck, or synchronization at b is not possible (this last case occurs if one of the strings is a prefix of the other). Thus, the two strings must be equal and they are the solution of the instance of the PCP problem. □

Corollary 4.1. *Barbed congruence and any complete τ-bisimilarity are undecidable in* HOcore *with four static restrictions.*

Theorem 4.3 actually shows that even *asynchronous barbed bisimilarity* (defined as the largest τ-bisimilarity that is output-barb preserving, and used in the definition of ordinary—as opposed to reduction-closed—barbed congruence) is undecidable. The corollary above then follows from the fact that all the relations there mentioned are at least as demanding as asynchronous barbed bisimilarity.

4.5 Other Extensions

We now examine the impact on decidability of bisimilarity of some extensions of HOcore. We omit the details, including precise statements of the results.

Abstractions. An *abstraction* is an expression of the form $(x)P$; it is a parametrized process. An abstraction has a functional type. Applying an abstraction $(x)P$ of type $T \rightarrow \diamond$ (where $\diamond$ is the type of all processes) to an argument W of type T yields the process $P\{{}^{W}\!/x\}$. The argument W can itself be an abstraction; therefore the order of an abstraction, that is, the level of arrow nesting in its type, can be arbitrarily high. The order can also be ω, if there are recursive types. By setting bounds on the order of the types of abstractions, one can define a hierarchy of subcalculi of the Higher-Order π-calculus (Sangiorgi and Walker, 2001); and when this bound is ω, one obtains a calculus capable of representing the π-calculus (for this all operators of the Higher-Order π-calculus are needed, including full restriction).

Allowing the communication of abstractions, as in the Higher-Order π-calculus, one then also needs to add in the grammar for processes an *application* construct of the form $P_1\langle P_2\rangle$, as a destructor for abstractions. Extensions in the LTS would be as follows. Suppose, as in (Sangiorgi, 1996b), that *beta-conversion* $\succ$ is the least precongruence on HOcore processes generated by the rule

$$(x)P_1\langle P_2\rangle \succ P_1\{{}^{P_2}\!/x\}.$$

The LTS could be then extended with a rule

$$\textsc{Beta}\qquad \frac{P \succ P_1' \qquad P_1' \xrightarrow{\alpha} Q}{P \xrightarrow{\alpha} Q}$$

Notice that with these additions, the characterization of bisimilarity as IO bisimilarity still holds. For a HOcore extended with abstractions and applications, $\sim^{\mathrm{o}}_{\mathrm{IO}}$ is still a congruence and is preserved by substitutions (by straighforward extensions of the proofs of Lemmas 4.2 and 4.3). Note that, however, abstraction application may increase the size of processes. If abstractions are of finite type (i.e., their order is smaller than ω) then only a finite number of such applications is possible, and decidability of bisimilarity is preserved. Decidability fails if the order is ω, intuitively because in this case it is possible to simulate the λ-calculus.

Output prefix If we add an output prefix construct $\overline{a}\langle P \rangle.\, Q$ to HOcore, then the proof of the characterization as IO bisimilarity breaks and, with it, the proof of decidability. Decidability proofs can however be adjusted by appealing to results on unique decomposition of processes and axiomatization (along the lines of Section 4.3).

Choice. Decidability remains with the addition of a choice operator to HOcore. The proofs require little modifications. The addition of both choice and output prefix is harder. It might be possible to extend the decidability proof for output prefix mentioned above so to accommodate also choice, but the details become much more complex.

Recursion. We do not know whether decidability is maintained by the addition of recursion (or similar operators such as replication).

4.6 Concluding Remarks

Process calculi are usually Turing complete and have an undecidable bisimilarity (and barbed congruence). Subcalculi have been studied where bisimilarity becomes decidable but then one loses Turing completeness. Examples are BPA and BPP (see, e.g., (Kucera and Jancar, 2006)) and CCS without restriction and relabeling (Christensen et al., 1994). In this chapter we have shown that HOcore is a Turing complete formalism for which bisimilarity is decidable. We do not know other concurrency formalisms where the same happens. Other peculiarities of HOcore are:

1. it is higher-order, and contextual bisimilarities (barbed congruence) coincide with higher-order bisimilarity (as well as with others, such as context and normal bisimilarities); and
2. it is asynchronous (in that there is no continuation underneath an output), yet asynchronous and synchronous bisimilarities coincide.

We do not know other non-trivial formalisms in which properties (1) or (2) hold (of course (1) makes sense only on higher-order models).

We have also given an axiomatization for bisimilarity. From this we have derived polynomial upper bounds to the decidability of bisimilarity. The axiomatization also intuitively explains why results such as decidability, and the collapse of many forms of bisimilarity, are possible even though HOcore is Turing complete: the bisimilarity relation is very discriminating.

While in Chapter 3 we have used encodings of Minsky machines, here we have used encodings of the Post correspondence problem (PCP) for our undecidability results. The encodings are tailored to analyze different problems: undecidability of termination, and undecidability of bisimilarity with static restrictions. The PCP encoding is always divergent, and therefore cannot be used to reason about termination. On the other hand, the encoding of Minsky machines would require at least one restriction for each instruction of the machine, and therefore would have given us a (much) worse result for static restrictions. We find both encodings interesting: they show different ways to exploit higher-order communications for modeling.

We have shown that bisimilarity becomes undecidable with the addition of four static restrictions. We do not know what happens with one, two, or three static restrictions. We also do not know whether the results presented would hold when one abstracts from τ-actions and moves to *weak* equivalences. The problem seems much harder; it reminds us of the situation for BPA and BPP, where strong bisimilarity is decidable but the decidability of weak bisimilarity is a long-standing open problem (see, e.g., (Kucera and Jancar, 2006)).

Chapter 5

On the Expressiveness of Forwarding and Suspension

In higher-order communication there are only two capabilities for received processes: execution and forwarding. In this chapter we aim at identifying the intrinsic souce of expressive power in HOcore by studying a limited form of forwarding. Such a form is obtained from the following syntactic restriction: output actions can only communicate the parallel composition of known closed processes and processes received through previously executed input actions. We study the expressiveness of $\mathrm{Ho}^{-\mathrm{f}}$, the fragment of HOcore featuring this style of communication, using decidability of termination and convergence of processes as a yardstick. Our main result shows that in $\mathrm{Ho}^{-\mathrm{f}}$ termination is decidable while convergence is undecidable. Then, as a way of recovering the expressiveness loss due to limited forwarding, we extend the calculus with a form of process suspension called *passivation*. The resulting calculus is called $\mathrm{HoP}^{-\mathrm{f}}$. Somewhat surprisingly, in $\mathrm{HoP}^{-\mathrm{f}}$ both termination and convergence are undecidable. This reveals a great deal of expressive power inherent to forms of suspension such as passivation.

The chapter is structured as follows. The syntax and semantics of $\mathrm{Ho}^{-\mathrm{f}}$ are introduced in Section 5.2. The encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$, and the undecidability of convergence are discussed in Section 5.3. The decidability of termination for $\mathrm{Ho}^{-\mathrm{f}}$ is addressed in Section 5.4. The expressiveness results for $\mathrm{HoP}^{-\mathrm{f}}$ are presented in Section 5.5. Some final remarks, as well as a review of related work, are included in Section 5.6.

While the decidability results for $\mathrm{Ho}^{-\mathrm{f}}$ have been previously presented as (Di Giusto et al., 2009a), the extension of $\mathrm{Ho}^{-\mathrm{f}}$ with passivation and its associated decidability results are original to this dissertation.

5.1 Introduction

Despite its minimality, in Chapter 3 HOcore was shown to be Turing complete by exhibiting an encoding of Minsky machines.[1] Therefore, properties such as

- *termination*, i.e., non existence of divergent computations
- *convergence*, i.e., existence of a terminating computation

are both undecidable in HOcore[2]. In contrast, somewhat surprisingly, strong bisimilarity is decidable, and several sensible bisimilarities coincide with it.

In this chapter, we shall aim at identifying the intrinsic source of expressive power in HOcore. A substantial part of the expressive power of a concurrent language comes from the ability of accounting for infinite behavior. In higher-order process calculi there is no explicit operator for such a behavior, as both recursion and replication can be encoded. We then find that infinite behavior resides in the interplay of higher-order communication, in particular, in the ability of *forwarding* a received process within an *arbitrary context*. For instance, consider the process $R = a(x).\,\overline{b}\langle P_x \rangle$, where P_x stands for a process P with free occurrences of a variable x. Intuitively, R receives a process on name a and forwards it on name b. It is easy to see that since there are no limitations on the structure of objects in output actions, the actual structure of P_x can be fairly complex. One could even "wrap" the process to be received in x using an arbitrary number of k "output layers", i.e., by letting $P_x = \overline{b_1}\langle \overline{b_2}\langle \ldots \overline{b_k}\langle x \rangle\rangle \ldots \rangle$. This *nesting capability* embodies a great deal of the expressiveness of HOcore: as a matter of fact, the encoding of Minsky machines in HOcore depends critically on nesting-based counters. Therefore, investigating suitable limitations to the kind of processes that can be communicated in an output action appears as a legitimate approach to assess the expressive power of higher-order concurrency.

With the above consideration in mind, in this chapter we propose $\mathrm{Ho}^{-\mathrm{f}}$, a sublanguage of HOcore in which output actions are limited so as to rule out the nesting capability (Section 5.2). In $\mathrm{Ho}^{-\mathrm{f}}$, output actions can communicate the parallel composition of two kinds of objects:

1. closed processes (i.e., processes that do not contain free variables), and
2. processes received through previously executed input actions.

Hence, the context in which the output action resides can only contribute to communication by "appending" pieces of code that admit no inspection, available in the form of a black-box.

[1] Along the paper we use the appellations "Turing complete" and "weak Turing complete" as in the criteria defined by Bravetti and Zavattaro (2009) and discussed in Section 2.3.3.

[2] Termination and convergence are sometimes also referred to as *universal* and *existential* termination, respectively.

More precisely, the grammar of $\mathrm{Ho}^{-\mathrm{f}}$ processes is the same as that of HOCORE, except for the production for output actions, which is replaced by the following one:

$$\overline{a}\langle x_1 \; \cdots \; x_k \;\; P\rangle$$

where $k \geq 0$ and P is a closed process. This modification directly restricts forwarding capabilities for output processes, which in turn, leads to a more limited structure of processes along reductions.

The limited style of higher-order communication enforced in $\mathrm{Ho}^{-\mathrm{f}}$ is relevant from a pragmatic perspective. In fact, communication in $\mathrm{Ho}^{-\mathrm{f}}$ is inspired by those cases in which a process P is communicated in a translated format $[\![P]\!]$, and the translation is not compositional. That is, the cases in which, for any process context C, the translation of $C[P]$ cannot be seen as a function of the translation of P, i.e., there exists no context D such that $[\![C[P]]\!] = D[\![P]\!]$.

More concretely, communication as in $\mathrm{Ho}^{-\mathrm{f}}$ can be related to several existing programming scenarios. The simplest example is perhaps mobility of already compiled code, on which it is not possible to apply inverse translations (such as reverse engineering). Other examples include *proof-carrying code* (Necula and Lee, 1998) and communication of *obfuscated code* (Collberg et al., 1998). The former features communication of executable code that comes with a certificate: a recipient can only check the certificate and decide whether to execute the code or not. The latter consists of the communication of source code that is made difficult to understand for, e.g., security/copyright reasons, while preserving its functionality.

In this chapter we study the expressiveness of $\mathrm{Ho}^{-\mathrm{f}}$ using decidability of termination and convergence of processes as a yardstick. Our main results are:

Undecidability of Convergence in $\mathrm{Ho}^{-\mathrm{f}}$. Similarly as in HOCORE, in $\mathrm{Ho}^{-\mathrm{f}}$ it is possible to encode Minsky machines. The calculus thus retains a significant expressive power despite of the limited forwarding capability. Unlike HOCORE, however, $\mathrm{Ho}^{-\mathrm{f}}$ is only weakly Turing complete. In fact, the encoding of Minsky machines in $\mathrm{Ho}^{-\mathrm{f}}$ is *not faithful* for it may introduce computations which do not correspond to the expected behavior of the modeled machine. Such computations are forced to be infinite and thus regarded as non-halting computations which are therefore ignored. This allows us to prove that a Minsky machine terminates if and only if its encoding in $\mathrm{Ho}^{-\mathrm{f}}$ converges. Consequently, convergence in $\mathrm{Ho}^{-\mathrm{f}}$ is *undecidable*.

Decidability of Termination in $\mathrm{Ho}^{-\mathrm{f}}$. In sharp contrast with HOCORE, termination in $\mathrm{Ho}^{-\mathrm{f}}$ is *decidable*. This result is obtained by appealing to the theory of *well-structured transition systems* (Finkel, 1990; Abdulla et al., 2000; Finkel and Schnoebelen, 2001), following the approach used by Busi et al. (2009). To the best of our knowledge, this is the first time the theory of well-structured transition systems is applied in a higher-order concurrency setting. This is significant because the adaptation to the $\mathrm{Ho}^{-\mathrm{f}}$ case is far from trivial. Indeed, as we shall discuss, this approach relies on defining an upper bound on the *depth* of the (set of) derivatives of a process. By depth of a process we mean its maximal nesting of input/output actions. Notice that, even with the limitation on forwarding enforced by $\text{Ho}^{-\text{f}}$, because of the "term copying" feature of higher-order calculi, variable instantiation might lead to a potentially larger process. Hence, finding suitable ways of bounding the set of derivatives of a process is rather challenging and needs care.

Undecidability of Termination *and* Convergence in $\text{Ho}^{-\text{f}}$ with Passivation. The decidability of termination in $\text{Ho}^{-\text{f}}$ provides compelling evidence on the fact that the limited forwarding entails a loss of expressive power for HOcore. It is therefore legitimate to investigate whether such an expressive power can be recovered while preserving the essence of the limited forwarding in $\text{Ho}^{-\text{f}}$. For this purpose, we extend $\text{Ho}^{-\text{f}}$ with a *passivation* construct that allows to *suspend* the execution of a running process. Forms of process suspension (such as passivation) are of both practical and theoretical interest as they are at the heart of mechanisms for *dynamic system reconfiguration*. The extension of $\text{Ho}^{-\text{f}}$ with passivation, called $\text{HoP}^{-\text{f}}$, is shown to be Turing complete by exhibiting a *faithful* encoding of Minsky machines. Therefore, in $\text{HoP}^{-\text{f}}$ *both* convergence and termination are *undecidable*. To the best of our knowledge, ours is the first result on the expressiveness and decidability of constructs for process suspension in the context of higher-order process calculi.

5.2 The Calculus

We now introduce the syntax and semantics of $\text{Ho}^{-\text{f}}$. We use a, b, c to range over names, and x, y, z to range over variables; the sets of names and variables are disjoint.

$$\begin{array}{rcllr}
P,\, Q & ::= & \overline{a}\langle x_1 \ \cdots \ x_k \ \ P\rangle & (\text{with } k \geq 0,\ \text{fv}(P) = \emptyset) & \text{output} \\
& \mid & a(x).\,P & & \text{input prefix} \\
& \mid & P \ \ Q & & \text{parallel composition} \\
& \mid & x & & \text{process variable} \\
& \mid & \mathbf{0} & & \text{nil}
\end{array}$$

An input $a(x).\,P$ binds the free occurrences of x in P. This is the only binder in the language. We write $\text{fv}(P)$ and $\text{bv}(P)$ for the set of free and bound variables in P, respectively. A process is *closed* if it does not have free variables. When $x \notin \text{fv}(P)$, we abbreviate $a(x).\,P$ as $a.\,P$. We also abbreviate $\overline{a}\langle\mathbf{0}\rangle$ as $\overline{a}$, $P_1 \ \ldots \ P_k$ as $\prod_{i=1}^{k} P_i$, and omit trailing occurrences of $\mathbf{0}$. Hence, an output action can be written as $\overline{a}\langle\prod_{k\in K} x_k \ \ P\rangle$. We write $\prod_1^n P$ as an abbreviation for the

$$\textsc{Inp}\quad a(x).\,P \xrightarrow{a(x)} P \qquad\qquad \textsc{Out}\quad \overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}$$

$$\textsc{Act1}\ \frac{P_1 \xrightarrow{\alpha} P_1' \qquad \mathrm{bv}(\alpha)\cap\mathrm{fv}(P_2)=\emptyset}{P_1 \ \ P_2 \xrightarrow{\alpha} P_1' \ \ P_2} \qquad \textsc{Tau1}\ \frac{P_1 \xrightarrow{\overline{a}\langle P\rangle} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \ \ P_2 \xrightarrow{\tau} P_1' \ \ P_2'\{{}^{P}\!/x\}}$$

Figure 5.1: An LTS for Ho$^{-\mathrm{f}}$. Rules Act2 and Tau2, the symmetric counterparts of Act1 and Tau1, have been omitted.

parallel composition of n copies of P. Further, $P\{{}^{Q}\!/x\}$ denotes the substitution of the free occurrences of x with process Q in P.

The LTS of Ho$^{-\mathrm{f}}$ is defined in Figure 5.1. It decrees there are three forms of transitions: τ transitions $P \xrightarrow{\tau} P'$; input transitions $P \xrightarrow{a(x)} P'$, meaning that P can receive at a a process that will replace x in the continuation P'; and output transitions $P \xrightarrow{\overline{a}\langle P'\rangle} P''$ meaning that P emits P' at a, and in doing so it evolves to P''. We use α to indicate a generic label of a transition. The notions of free and bound variables extend to labels as expected.

The internal runs of a process are given by sequences of *reductions*. Given a process P, its reductions $P \longrightarrow P'$ are defined as $P \xrightarrow{\tau} P'$. We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$; notation $\longrightarrow^j$ is to stand for a sequence of j reductions. We use $P \not\longrightarrow$ to denote that there is no P' such that $P \longrightarrow P'$. Following Busi et al. (2009) we now define process convergence and process termination. Observe that termination implies convergence while the opposite does not hold.

Definition 5.1. *Let P be a* Ho$^{-\mathrm{f}}$ *process.*

1. *We say that P* converges *iff there exists P' such that $P \longrightarrow^* P'$ and $P' \not\longrightarrow$.*

2. *We say that P* terminates *iff there exist no $\{P_i\}_{i\in\mathbb{N}}$ such that $P_0 = P$ and $P_j \longrightarrow P_{j+1}$ for any j.*

5.3 Convergence is Undecidable in Ho$^{-\mathrm{f}}$

In this section we show that Ho$^{-\mathrm{f}}$ is powerful enough to model Minsky machines (see Section 2.3.3). We present an encoding that is not *faithful*: unlike the encoding of Minsky machines in HOcore, it may introduce computations which do not correspond to the expected behavior of the modeled machine. Such computations are forced to be infinite and thus regarded as non-halting computations which are therefore ignored. More precisely, given a Minsky machine N, its encoding $[\![N]\!]$ has a terminating computation if and only if N terminates. This allows to prove that convergence is undecidable.

The following notion of structural congruence will be useful later on.

$$\textsc{Register } r_j \qquad [\![r_j = m]\!]_{\mathrm{M}} = \prod_1^m \overline{u_j}$$

$$\begin{array}{lcl}
\multicolumn{3}{l}{\textsc{Instructions } (i : I_i)} \\
[\![(i : \mathrm{INC}(r_j))]\!]_{\mathrm{M}} & = & !p_i.\,(\overline{u_j} \;\; set_j(x).\,(\overline{set_j}\langle x \;\; \textsc{Inc}\rangle \;\; \overline{p_{i+1}})) \\
[\![(i : \mathrm{DECJ}(r_j, s))]\!]_{\mathrm{M}} & = & \quad !p_i.\,(\overline{loop} \;\; u_j.\,loop.\,set_j(x).\,(\overline{set_j}\langle x \;\; \textsc{Dec}\rangle \;\; \overline{p_{i+1}})) \\
 & & !p_i.\,set_j(x).\,(x \;\; \overline{set_j}\langle x\rangle \;\; \overline{p_s})
\end{array}$$

where

$$\textsc{Inc} \;=\; \overline{loop} \qquad\qquad \textsc{Dec} = loop$$

Figure 5.2: Encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$

Definition 5.2. *The* structural congruence *relation is the smallest congruence generated by the following laws:*

$$P \;\; \mathbf{0} \equiv P, \;\; P_1 \;\; P_2 \equiv P_2 \;\; P_1, \;\; P_1 \;\; (P_2 \;\; P_3) \equiv (P_1 \;\; P_2) \;\; P_3.$$

Lemma 5.1. *If $P \xrightarrow{\alpha} P'$ and $P \equiv Q$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.*

Proof. By induction on the derivation of $P \equiv Q$, then by case analysis on $P \xrightarrow{\alpha} Q$. □

5.3.1 Encoding Minsky Machines into $\mathrm{Ho}^{-\mathrm{f}}$

The encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$ is denoted by $[\![\cdot]\!]_{\mathrm{M}}$ and presented in Figure 5.2. The encoding is assumed to execute in parallel with a process $loop.\,$Div, which represents divergent behavior that is spawned in certain cases with an output on name $loop$. This will be made more precise later, when defining the encoding of a configuration of a Minsky machine. Before that, we begin by discussing the encodings of registers and instructions.

A register r_j that stores the number m is encoded as the parallel composition of m copies of the unit process $\overline{u_j}$. To implement the test for zero it is necessary to record how many increments and decrements have been performed on the register r_j. This is done by using a special process $\textsc{Log}_j$, which is communicated back and forth on name set_j. More precisely, every time an increment instruction occurs, a new copy of the process $\overline{u_j}$ is created, and the process $\textsc{Log}_j$ is updated by adding the process Inc in parallel. Similarly for decrements: a copy of $\overline{u_j}$ is consumed and the process Dec is added to $\textsc{Log}_j$. As a result, after k increments and l decrements on register r_j, we have that $\textsc{Log}_j = \prod_k \textsc{Inc} \;\; \prod_l \textsc{Dec}$, which we abbreviate as $\textsc{Log}_j[k, l]$.

Each instruction $(i : I_i)$ is a replicated process guarded by p_i, which represents the program counter when $p = i$. Once p_i is consumed, the instruction is active and, in the case of increments and decrements, an interaction with a register occurs. We already described the behavior of increments. Let us now focus on decrements, the instructions that can introduce divergent —unfaithful— computations. In this case, the process can internally choose either to actually perform a decrement and proceed with the next instruction, or to jump. This internal choice takes place on p_i; it can be seen as a *guess* the process makes on the actual number stored by the register r_j. Therefore, two situations can occur:

1. *The process chooses to decrement r_j.* In this case a process $\overline{loop}$ as well as an input on u_j become immediately available. The purpose of the latter is to produce a synchronization with a complementary output on $\overline{u_j}$ (that represents a unit of r_j).

   If this operation succeeds (i.e., the guess is right as the content of r_j is greater than 0) then a synchronization between the output $\overline{loop}$ —available at the beginning— and the input on $loop$ that guards the update of $\textsc{Log}_j$ takes place. After this synchronization, the log of the register is updated (this is represented by two synchronizations on name set_j) and instruction p_{i+1} is enabled.

   Otherwise, if the synchronization on u_j fails then it is because the content of r_j is zero and the process made a wrong guess. The process $\overline{loop}$ available at the beginning then synchronizes with the external process $loop.\,\textsc{Div}$, thus spawning a divergent computation.

2. *The process chooses to jump to instruction p_s.* In this case, the encoding checks if the actual value stored by r_j is zero. To do so, the process receives the process $\textsc{Log}_j$ on name set_j and launches it. The log contains a number of $\textsc{Inc}$ and $\textsc{Dec}$ processes; depending on the actual number of increments and decrements, two situations can occur.

   In the first situation, the number of increments is equal to the number of decrements (say k); hence, the value of the r_j is indeed zero and the process made a right guess. In this case, k synchronizations on name $loop$ take place and instruction p_s is enabled.

   In the second situation, the number of increments is greater than the number of decrements; hence, the value of r_j is greater than zero and the process made a wrong guess. As a result, at least one of the $\overline{loop}$ signals remains active; by means of a synchronization the process $loop.\,\textsc{Div}$ this is enough to to spawn a divergent computation.

Before executing the instructions, we require both registers in the Minsky machine to be set to zero. This is to guarantee correctness: starting with values different from zero in the registers (without proper initialization of the logs) can lead to inconsistencies. For instance, the test for zero would succeed (i.e., without spawning a divergent computation) even for a register whose value is different from zero.

The following notation will be useful.

Notation 5.1. *Let N be a Minsky machine. The configuration (i, m_0, m_1) of N is* annotated *as $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$, where, for $j \in \{0,1\}$, k_j and l_j stand for the number of increments and decrements performed on r_j.*

Because we assume the value of both registers to be initialized with zero before executing the instructions, the following is immediate.

Fact 5.1. *Let* $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ *be an annotated Minsky configuration. We then have, for* $n \in \{0,1\}$*: (i)* $k_n = l_n$ *if and only if* $r_n = 0$*; and (ii)* $k_n > l_n$ *if and only if* $r_n > 0$.

We are now ready to define the encoding of a configuration of the Minsky machine. As mentioned before, the encodings of instructions and registers are put in parallel with a process that spawns divergent behavior in case of a wrong guess.

Definition 5.3 (Encoding of Configurations). *Let* N *be a Minsky machine with registers* r_0*,* r_1 *and instructions* $(1 : I_1), \dots, (n : I_n)$*. For* $j \in \{0,1\}$*, suppose fresh, pairwise different names* r_j*,* $p_1, \dots, p_n$*,* set_j*,* $loop$*,* $check_j$*. Also, let* $\textsc{Div}$ *be a divergent process (e.g.* $\overline{w}\ \ !w.\overline{w}$*). Given the encodings in Figure 5.2, we have:*

1. *The initial configuration* $(1, 0^{0,0}, 0^{0,0})$ *of* N *is encoded as:*

$$[\![(1, 0^{0,0}, 0^{0,0})]\!]_{\mathrm{M}} ::= \overline{p_1}\ \ \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathrm{M}}\ \ loop.\textsc{Div}\ \ \overline{set_0}\langle \mathbf{0} \rangle\ \ \overline{set_1}\langle \mathbf{0} \rangle\ .$$

2. *A configuration* $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ *of* N*is encoded as:*

$$\begin{aligned}[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} \quad = \quad & \overline{p_i}\ \ [\![r_0 = m_0]\!]_{\mathrm{M}}\ \ [\![r_1 = m_1]\!]_{\mathrm{M}}\ \ \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathrm{M}} \\ & loop.\textsc{Div}\ \ \overline{set_0}\langle \textsc{Log}_0[k_0, l_0] \rangle\ \ \overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle\ .\end{aligned}$$

5.3.2 Correctness of the Encoding

We formalize the correctness of our encoding by means of two lemmas ensuring completeness (Lemma 5.2) and soundness (Lemma 5.3). Both these lemmas give us Theorem 5.1. We begin by formalizing the following intuition: removing the program counter from the encoding of configurations leads to a stuck process.

Proposition 5.1. *Let* N *be a Minsky machine with registers* r_0*,* r_1 *and instructions* $(1 : I_1), \dots, (n : I_n)$*. Given the encodings in Figure 5.2, let* P *be defined as:*

$$\begin{aligned}P \quad = \quad & [\![r_0 = m_0]\!]_{\mathrm{M}}\ \ [\![r_1 = m_1]\!]_{\mathrm{M}}\ \ \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathrm{M}}\ \ loop.\textsc{Div} \\ & \overline{set_0}\langle \textsc{Log}_0[k_0, l_0] \rangle\ \ \overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle\,.\end{aligned}$$

Then $P \not\longrightarrow$*.*

Proof. Straightforward by the following facts:

1. Processes $[\![r_0 = m_0]\!]_{\mathrm{M}}$, $[\![r_1 = m_1]\!]_{\mathrm{M}}$, $\overline{set_0}\langle \textsc{Log}_0[k_0, l_0] \rangle$, and $\overline{set_1}\langle \textsc{Log}_1[k_1, l_1] \rangle$ are output actions that cannot evolve on their own.

2. For every $i \in 1..n$, each $[\![(i : I_i)]\!]_{\mathrm{M}}$ is an input-guarded process, waiting for an activation signal on p_i.

3. $loop.\,\textsc{Div}$ is an input-guarded process, and every output on $loop$ appears guarded inside a decrement instruction.

□

Remark 5.1. *Before entering into the proofs two remarks are in order. First, with a little abuse of notation, we use notation $Q \not\longrightarrow$ also for configurations of Minsky machines. Second, the encoding of input-guarded replication we have introduced here takes two reductions to release a new copy of the guarded process (see Definition 3.5 and Lemma 3.1). However, for the sake of simplicity, in proofs we shall denote only one of such reductions. In any case, it must be taken into account that two reductions are required.*

We now state that the encoding is correct.

Lemma 5.2 (Completeness). *Let $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ be an (annotated) configuration of a Minsky machine N. Then, it holds:*

1. *If $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1}) \not\longrightarrow$ then $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} \not\longrightarrow$*

2. *If $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1}) \longrightarrow_{\mathrm{M}} (i', m_0'^{\,k_0',l_0'}, m_1'^{\,k_1',l_1'})$ then, for some P, $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} \longrightarrow^* P \equiv [\![(i', m_0'^{\,k_0',l_0'}, m_1'^{\,k_1',l_1'})]\!]_{\mathrm{M}}$*

Proof. For (1) we have that if $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1}) \not\longrightarrow$ then, by definition of Minsky machine, the program counter p is set to a non-existent instruction; i.e., for some $i \notin [1..n]$, $p = i$. Therefore, in process $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}}$ no instruction is guarded by p_i. The thesis then follows as an easy consequence of Proposition 5.1.

For (2) we proceed by a case analysis on the instruction performed by N. Hence, we distinguish three cases corresponding to the behaviors associated to rules M-Jmp, M-Dec, and M-Inc. Without loss of generality we assume instructions on register r_0.

Case M-Inc We have a Minsky machine configuration $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ with $(i : \mathrm{INC}(r_0))$. By definition, its encoding into $\mathrm{Ho}^{-\mathrm{f}}$ is as follows:

$$\begin{array}{rcl}
[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} & = & \overline{p_i} \;\; [\![r_0 = m_0]\!]_{\mathrm{M}} \;\; [\![r_1 = m_1]\!]_{\mathrm{M}} \prod_{h=1..n, i \neq h} [\![(h : I_h)]\!]_{\mathrm{M}} \\
 & & !p_i.\,(\overline{u_0} \;\; set_0(x).\,(\overline{set_0}\langle x \;\; \textsc{Inc}\rangle \;\; \overline{p_{i+1}})) \\
 & & loop.\,\textsc{Div} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle
\end{array}$$

We begin by noting that the program counter p_i is consumed by the encoding of the instruction i. As a result, process $\overline{u_0}$ is left unguarded; this represents the actual increment. We then have:

$$\begin{aligned}[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}} \quad \longrightarrow\equiv \quad & [r_0 = m_0 + 1]_{\mathrm{M}} \;\; set_0(x).(\overline{set_0}\langle x \;\; \textsc{Inc}\rangle \;\; \overline{p_{i+1}}) \\ & \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; S = T\end{aligned}$$

where S stands for the rest of the system, i.e.,

$$S = [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [(h : I_h)]_{\mathrm{M}} \;\; loop.\,\textsc{Div} \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle.$$

Now there is a synchronization on set_0 for updating the log of register r_0. This leaves $\overline{p_{i+1}}$ unguarded, so the next instruction is enabled.

$$\begin{aligned}T \quad \longrightarrow \quad & \overline{p_{i+1}} \;\; [r_0 = m_0 + 1]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [(h : I_h)]_{\mathrm{M}} \\ & loop.\,\textsc{Div} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0 + 1, l_0]\rangle \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle = T'.\end{aligned}$$

We notice that $T' \equiv [(i + 1, m_0 + 1^{k_0+1,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}}$, as desired.

Case M-Dec We have a Minsky machine configuration $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ with $r_0 > 0$ and $(i : \mathrm{DECJ}(r_0, s))$. By definition, its encoding into $\mathrm{Ho}^{-\mathrm{f}}$ is as follows:

$$\begin{aligned}[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}} \quad = \quad & \overline{p_i} \;\; [r_0 = m_0]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1..n, i \neq h} [(h : I_h)]_{\mathrm{M}} \\ & !p_i.(\overline{loop} \;\; u_0.\,loop.\,set_0(x).(\overline{set_0}\langle x \;\; \textsc{Dec}\rangle \;\; \overline{p_{i+1}})) \\ & !p_i.\,set_0(x).(x \;\; \overline{set_0}\langle x\rangle \;\; \overline{p_s}) \\ & loop.\,\textsc{Div} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle\end{aligned}$$

In $[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}}$ there is an internal choice on the program counter p_i. This represents a guess on the value of r_0: $\overline{p_i}$ can either synchronize with the first input-guarded process (so as to perform the actual decrement of the register) or with the second one (so as to perform a jump). Let us suppose that $[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}}$ makes the right guess in this case, i.e., $\overline{p_i}$ synchronizes with the first input-guarded process. We then have:

$$\begin{aligned}[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}} \quad \longrightarrow \quad & [r_0 = m_0]_{\mathrm{M}} \\ & \overline{loop} \;\; u_0.\,loop.\,set_0(x).(\overline{set_0}\langle x \;\; \textsc{Dec}\rangle \;\; \overline{p_{i+1}}) \\ & \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; S = T_1\end{aligned}$$

where S stands for the rest of the system, i.e.,

$$\begin{aligned}S \quad = \quad & [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [(h : I_h)]_{\mathrm{M}} \;\; loop.\,\textsc{Div} \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle \\ & !p_i.\,set_0(x).(x \;\; \overline{set_0}\langle x\rangle \;\; \overline{p_s}).\end{aligned}$$

Since we have assumed that $r_0 > 0$, we are sure that a synchronization on u_0 can take place, and thus the value of r_0 decreases. Immediately after, there is also a synchronization on $loop$. More precisely, we have

$$T_1 \quad \longrightarrow^2 \quad [r_0 = m_0 - 1]_{\mathrm{M}} \;\; set_0(x).\,(\overline{set_0}\langle x \;\; \textsc{Dec}_0\rangle \;\; \overline{p_{i+1}}) \;\; S = T_2\,.$$

Now the update of the log associated to r_0 can take place, and a synchronization on set_0 is performed. As a result, the process $\overline{p_{i+1}}$ becomes unguarded and the next instruction is enabled:

$$\begin{aligned} T_2 \quad \longrightarrow\equiv \quad & \overline{p_{i+1}} \;\; [r_0 = m_0 - 1]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [(h : I_h)]_{\mathrm{M}} \\ & loop.\,\textsc{Div} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0, l_0 + 1]\rangle \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle = T_3\,. \end{aligned}$$

Clearly, $T_3 \equiv [(i + 1, m_0 - 1^{k_0, l_0+1}, m_1^{k_1, l_1})]_{\mathrm{M}}$, as desired.

Case M-Jmp This case is similar to the previous one. We have a Minsky machine configuration $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ with $(i : \mathrm{DECJ}(r_0, s))$. In this case, $m_0 = 0$. Hence, using Fact 5.1 we have that $k_0 = l_0$.

Again, we start from $[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}}$. There is an internal choice on the name p_i. Let us suppose that $[(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}}$ makes the right guess, which in this case corresponds to the synchronization of $\overline{p_i}$ and the second input-guarded process. We then have

$$\begin{aligned} [(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]_{\mathrm{M}} \quad \longrightarrow \quad & [r_0 = m_0]_{\mathrm{M}} \;\; set_0(x).\,(x \;\; \overline{set_0}\langle x\rangle \;\; \overline{p_s}) \\ & \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; S' = T_1\,. \end{aligned}$$

where S' stands for the rest of the system, i.e.,

$$\begin{aligned} S \quad = \quad & [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [(h : I_h)]_{\mathrm{M}} \;\; loop.\,\textsc{Div} \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle \\ & !p_i.\,(\overline{loop} \;\; u_0.\,loop.\,set_0(x).\,(\overline{set_0}\langle x \;\; \textsc{Dec}\rangle \;\; \overline{p_{i+1}}))\,. \end{aligned}$$

Now there is a synchronization on set_0. As a result, the content of the log is left at the top-level and hence executed. It is not lost, however, as it is still preserved inside an output on set_0:

$$\begin{aligned} T_1 \quad \longrightarrow\equiv \quad & \overline{p_s} \;\; [r_0 = m_0]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [(h : I_h)]_{\mathrm{M}} \\ & loop.\,\textsc{Div} \;\; \prod^{k_0} \textsc{Inc} \;\; \prod^{l_0} \textsc{Dec} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \\ & \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle = T_2\,. \end{aligned}$$

Recall that $k_0 = l_0$. Starting in T_2, we have that k_0 synchronizations on $loop$ take place; each of these corresponds to the interaction between a process INC and a corresponding process DEC. All of these processes are consumed. We then have that there exists a T_3 such that (i) $T_2 \longrightarrow^{k_0} T_3$ and (ii) $T_3 \equiv [\![(s, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}}$, as wanted.

□

Proposition 5.2. *Let $P_0 = [\![(i, r_0^{k_0,l_0}, r_1^{k_1,l_1})]\!]_{\mathrm{M}}$ be the encoding of a Minsky machine configuration as in Definition 5.3, with $(i : \mathrm{DEC}(r_j, s))$ and $k_j > l_j$ (for $j \in \{0,1\}$).*

Suppose $P_0 \longrightarrow^* P$ such that

$$\begin{aligned} P \quad \equiv \quad & [\![r_0 = m_0]\!]_{\mathrm{M}} \prod^{k_j} \textsc{Inc} \prod^{l_j} \textsc{Dec} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; \overline{p_s} \\ & loop.\,\textsc{Div} \;\; S \end{aligned}$$

and where S is defined as

$$\begin{aligned} S \quad = \quad & [\![r_1 = m_1]\!]_{\mathrm{M}} \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathrm{M}} \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle \\ & !p_i.\,(\overline{loop} \;\; u_0.\,loop.\,set_0(x).\,(\overline{set_0}\langle x \;\; \textsc{Dec}\rangle \;\; \overline{p_{i+1}}))\,. \end{aligned}$$

Then P does not converge.

Proof (Sketch). Without loss of generality, we focus on the case in which $j = 0$ —the proof is analogous for $j = 1$— and assume that $k_0 = l_0 + 1$. The thesis follows by noticing that the only possibilities for behavior are given by sub-processes $\prod^{k_0}$ INC, $\prod^{l_0}$ DEC, and $loop.$ DIV of P. In fact, using Definition 5.3 it is possible to infer all the other processes cannot reduce on their own. The same definition decrees that INC $= \overline{loop}$ and DEC $= loop$. It is easy to see that divergent behavior can be spawned by any of the k_0 occurrences of INC. Notice that there is always at least one occurrence of INC ready to spawn divergency: even in the case some of such occurrences would reduce with corresponding input actions on $loop$, since $k_0 = l_0 + 1$ in every computation there is at least an output $\overline{loop}$ ready to reduce with $loop.$ DIV. Since no other process can reduce with the free $\overline{loop}$, this means there is always a computation in which it reduces with the process $loop.$ DIV. Hence, divergent behavior is spawned in all cases, and we are done. □

Lemma 5.3 (Soundness). *Let $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})$ be a configuration of a Minsky machine N. Given $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}}$, for some $n > 0$ and process $P \in \mathrm{Ho}^{-\mathrm{f}}$, we have that:*

1. *If $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} \longrightarrow^n P$ then either:*
   - *$P \equiv [\![(i', m_0'^{k_0',l_0'}, m_1'^{k_1',l_1'})]\!]_{\mathrm{M}}$ and $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1}) \longrightarrow_{\mathrm{M}} (i', m_0'^{k_0',l_0'}, m_1'^{k_1',l_1'})$, or*
   - *P is a divergent process.*

2. *For all* $0 \leq m < n$, *if* $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} \longrightarrow^m P$ *then, for some* P', $P \longrightarrow P'$.

3. *If* $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} \not\longrightarrow$ *then* $(i, m_0^{k_0,l_0}, m_1^{k_1,l_1}) \not\longrightarrow$.

Proof. For (1), since $n > 0$, in all cases there is at least one reduction from $[\![(i, m_0, m_1)]\!]_{\mathrm{M}}$. An analysis of the structure of process $[\![(i, m_0, m_1)]\!]_{\mathrm{M}}$ reveals that, in all cases, the first step corresponds to the consumption of the program counter p_i. This implies that there exists an instruction labeled with i, that can be executed from the configuration (i, m_0, m_1). We proceed by a case analysis on the possible instruction, considering also the fact that the register on which the instruction acts can hold a value equal or greater than zero.

Case $i : \mathrm{INC}(r_0)$**:** Then the process evolves deterministically (up-to structural congruence) to $P \equiv [\![(i+1, m_0+1, m_1)]\!]_{\mathrm{M}}$ in $n = 2$ reductions. This is illustrated in the analogous case in the proof of Lemma 5.2(2).

Case $i : \mathrm{DEC}(r_0, s)$ **with** $r_0 > 0$**:** We then have three main reduction sequences; one of them is finite, the other two are infinite. The finite reduction sequence is illustrated in the analogous case in the proof of Lemma 5.2(2), where it is shown how $[\![(i, r_0^{k_0,l_0}, r_1^{k_1,l_1})]\!]_{\mathrm{M}}$ may perform a sequence of $n = 4$ reductions that leads to $[\![(i+1, m_0 - 1, m_1)]\!]_{\mathrm{M}}$.

The remaining (infinite) reduction sequences arise from the internal choice in p_i that takes place in $[\![(i, r_0^{k_0,l_0}, r_1^{k_1,l_1})]\!]_{\mathrm{M}}$. The first such sequences arises when $\overline{p_i}$ synchronizes with the first input-guarded replication on p_i (the one implementing decrement); this is as in the analogous case in the proof of Lemma 5.2(2). This synchronization leads to process T_1 in which the diverging computation arises from the synchronization between the process $\overline{loop}$ and the process $loop.\,\textsc{Div}$ that spawns divergent behavior and is always in parallel.

The second infinite sequence arises when $\overline{p_i}$ synchronizes with the second input-guarded replication on p_i (the one implementing jump). Notice that since $r_0 > 0$, using Fact 5.1, we know that $k_0 > l_0$. It is sufficient to assume that $k_0 = l_0 + 1$. We have

$$\begin{array}{rcl} [\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathrm{M}} & \longrightarrow & [\![r_0 = m_0]\!]_{\mathrm{M}} \;\; set_0(x).\,(x \;\; \overline{set_0}\langle x\rangle \;\; \overline{p_s}) \\ & & loop.\,\textsc{Div} \;\; \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \;\; S = T_1 \end{array}$$

where S stands for the rest of the system, i.e.,

$$\begin{array}{rcl} S & = & [\![r_1 = m_1]\!]_{\mathrm{M}} \;\; \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathrm{M}} \;\; \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle \\ & & !p_i.\,(\overline{loop} \;\; u_0.\,loop.\,set_0(x).\,(\overline{set_0}\langle x \;\; \textsc{Dec}\rangle \;\; \overline{p_{i+1}}))\,. \end{array}$$

In T_1 there is a synchronization on set_0. Using the definition of $\textsc{Log}$, we have:

$$\begin{array}{rcl} T_1 & \longrightarrow & [r_0 = m_0]_{\mathsf{M}} \ \prod^{l_0+1} \textsc{Inc} \ \prod^{l_0} \textsc{Dec} \ \ \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \ \ \overline{p_s} \\ & & loop.\, \textsc{Div} \ \ S = T_2\,. \end{array}$$

The above puts us in the scenario of Proposition 5.2 which ensures that whenever a configuration in which the number of increments is greater or equal than the number of decrements is reached (as in T_2 above), the corresponding $\mathsf{Ho}^{-\mathsf{f}}$ process does not converge. This concludes the analysis for the case of decrement.

Case $i : \mathsf{DEC}(r_0, s)$ **with** $r_0 = 0$**:** Also in this case we have three main reduction sequences, one of them is finite, while the other two are infinite. The finite reduction sequence is illustrated in the analogous case in the proof of Lemma 5.2(2), where it is shown how $[\![(i, r_0^{k_0,l_0}, r_1^{k_1,l_1})]\!]_{\mathsf{M}}$ may perform a sequence of $n = 2 + l_0$ reductions that leads to $[\![(s, m_0, m_1)]\!]_{\mathsf{M}}$.

The two infinite reduction sequences arise similarly as in the previous case. The first one arises after the two reduction steps that lead to process T_3 in the analogous case in the proof of Lemma 5.2(2). Indeed, only a single occurrence of process $\textsc{Inc}$ is sufficient to synchronize with process $loop.\,\textsc{Div}$ and to produce divergent behavior.

The second infinite reduction sequence arises when the process makes a wrong guess on the content of the register. Again, we carry our analysis starting from process $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$, given in the analogous case of the proof of Lemma 5.2(2). After the synchronization on p_i we have

$$\begin{array}{rcl} [\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}} & \longrightarrow & [r_0 = m_0]_{\mathsf{M}} \\ & & \overline{loop} \ \ u_0.\, loop.\, set_0(x).\, (\overline{set_0}\langle x \ \ \textsc{Dec}\rangle \ \ \overline{p_{i+1}}) \\ & & \overline{set_0}\langle \textsc{Log}_0[k_0, l_0]\rangle \ \ loop.\,\textsc{Div} \ \ S' = T_1 \end{array}$$

where S' is the rest of the system, i.e.

$$\begin{array}{rcl} S' & = & !m_i.\, (set_0(x).\, (x \ \ \overline{set_0}\langle x\rangle \ \ \overline{p_s})) \ \ [r_1 = m_1]_{\mathsf{M}} \\ & & \prod_{h=1}^{n} [\![(h : I_h)]\!]_{\mathsf{M}} \ \ \overline{set_1}\langle \textsc{Log}_1[k_1, l_1]\rangle. \end{array}$$

It is easy to observe that since $r_0 = 0$ there is no output on u_j that can synchronize with the input in T_1. In fact, the only possible synchronization is on $loop$, which leaves the divergent process unguarded. So we have that in two reduction steps $[\![(i, m_0^{k_0,l_0}, m_1^{k_1,l_1})]\!]_{\mathsf{M}}$ evolves into a diverging process, and the thesis holds.

Notice that statement (2) follows easily from the above analysis.

As for (3), using Proposition 5.1 we know that if $[\![(i, m_0, m_1)]\!]_{\mathrm{M}} \not\longrightarrow$ then it is because p_i is not enabling any instruction. Hence, $[\![(i, m_0, m_1)]\!]_{\mathrm{M}}$ corresponds to the encoding of a halting instruction and we have that $(i, m_0, m_1) \not\longrightarrow$, as desired. □

Summarizing Lemmas 5.2 and 5.3 we have the following:

Theorem 5.1. *Let N be a Minsky machine with registers $r_0 = m_0$, $r_1 = m_1$, instructions $(1 : I_1), \ldots, (n : I_n)$, and configuration (i, m_0, m_1). Then (i, m_0, m_1) terminates if and only if process $[\![(i, m_0, m_1)]\!]_{\mathrm{M}}$ converges.*

As a consequence of the results above we have that convergence is undecidable.

Corollary 5.1. *Convergence is undecidable in* $\mathrm{Ho}^{-\mathrm{f}}$.

5.4 Termination is Decidable in $\mathrm{Ho}^{-\mathrm{f}}$

In this section we prove that termination is decidable for $\mathrm{Ho}^{-\mathrm{f}}$ processes. As hinted at in the introduction, this is in sharp contrast with the analogous result for HOcore. The proof appeals to the theory of well-structured transition systems, whose main definitions and results we summarize next.

5.4.1 Well-Structured Transition Systems

The following results and definitions are from (Finkel and Schnoebelen, 2001), unless differently specified. Recall that a *quasi-order* (or, equivalently, preorder) is a reflexive and transitive relation.

Definition 5.4 (Well-quasi-order). *A* well-quasi-order *(wqo) is a quasi-order $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2 \ldots \in X$, there exist indexes $i < j$ such that $x_i \leq x_j$.*

Note that if $\leq$ is a wqo then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 < i_1 < i_2 < \ldots$). Thus well-quasi-orders exclude the possibility of having infinite strictly decreasing sequences.

We also need a definition for (finitely branching) transition systems. This can be given as follows. Here and in the following $\rightarrow^*$ denotes the reflexive and transitive closure of the relation $\rightarrow$.

Definition 5.5 (Transition system). *A* transition system *is a structure $TS = (S, \rightarrow)$, where S is a set of states and $\rightarrow \subseteq S \times S$ is a set of transitions. We define $Succ(s)$ as the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of S. We say that TS is finitely branching if, for each $s \in S$, $Succ(s)$ is finite.*

The function $Succ$ will also be used on sets by assuming the point-wise extension of the above definitions. The key tool to decide several properties of computations is the notion of *well-structured transition system*. This is a transition system equipped with a well-quasi-order on states which is (upward) compatible with the transition relation. Here we will use a strong version of compatibility; hence the following definition.

Definition 5.6 (Well-structured transition system). *A* well-structured transition system with strong compatibility *is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-order $\leq$ on S, such that the two following conditions hold:*

1. *$\leq$ is a well-quasi-order;*

2. *$\leq$ is strongly (upward) compatible with $\rightarrow$, that is, for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$ holds.*

The following theorem is a special case of Theorem 4.6 in (Finkel and Schnoebelen, 2001) and will be used to obtain our decidability result.

Theorem 5.2. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leq$, and computable $Succ$. Then the existence of an infinite computation starting from a state $s \in S$ is decidable.*

We will also need a result due to Higman (1952) which allows to extend a well-quasi-order from a set S to the set of the finite sequences on S. More precisely, given a set S let us denote by S^* the set of finite sequences built by using elements in S. We can define a quasi-order on S^* as follows.

Definition 5.7. *Let S be a set and $\leq$ a quasi-order over S. The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1 t_2 \ldots t_m$ and $u = u_1 u_2 \ldots u_n$. We have that $t \leq_* u$ if and only if there exists an injection f from $\{1, 2, \ldots m\}$ to $\{1, 2, \ldots n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, \ldots, m$.*

The relation $\leq_*$ is clearly a quasi-order over S^*. It is also a wqo, since we have the following result.

Lemma 5.4 (Higman (1952)). *Let S be a set and $\leq$ a wqo over S. Then $\leq_*$ is a wqo over S^*.*

Finally we will use also the following proposition, whose proof is immediate.

Proposition 5.3. *Let S be a finite set. Then the equality is a wqo over S.*

$$\textsc{Inp}\quad a(x).\,P \overset{a(x)}{\longmapsto} P \qquad\qquad \textsc{Out}\quad \overline{a}\langle P\rangle \overset{\overline{a}\langle P\rangle}{\longmapsto} \mathbf{0}$$

$$\textsc{Act1}\ \frac{P_1 \overset{\alpha}{\longmapsto} P_1'}{P_1 \;\; P_2 \overset{\alpha}{\longmapsto} P_1' \;\; P_2} \qquad \textsc{Tau1}\ \frac{P_1 \overset{\overline{a}\langle P\rangle}{\longmapsto} P_1' \qquad P_2 \overset{a(x)}{\longmapsto} P_2'}{P_1 \;\; P_2 \overset{\tau}{\longmapsto} P_1' \;\; P_2'\{P/x\}}$$

Figure 5.3: A finitely branching LTS for Ho$^{-\mathrm{f}}$. Rules Act2 and Tau2, the symmetric counterparts of Act1 and Tau1, have been omitted.

5.4.2 A Finitely Branching LTS for Ho$^{-\mathrm{f}}$

In order to exploit the theory of well-structured transition systems, a finitely branching LTS for Ho$^{-\mathrm{f}}$ is necessary. This is not a significant requirement in our case; the sensible issue here is the treatment of alpha-conversion. To that end, we introduce an alternative LTS *without* alpha-conversion. As we shall see, since we restrict ourselves to closed processes and proofs focus on internal synchronizations, the finitely branching LTS is equivalent to that introduced in Section 5.2. The alternative LTS is given in Figure 5.3; its most noticeable feature is the absence of a side condition on rule Act1.

Lemma 5.5. *Let P be a closed* Ho$^{-\mathrm{f}}$ *process. For every $P' \in$* Ho$^{-\mathrm{f}}$ *if $P \longrightarrow P'$ then P' is a closed process.*

Proof. By induction on the height of the inference tree for $P \longrightarrow P'$ considering the possible cases of the last step of the inference. There are four cases, corresponding to those related to rules Tau1, Tau2, Act1, and Act2. Let us focus only in the case in which Tau1 is the last rule applied; the other cases are similar or simpler. Then $P \equiv P_1 \;\; P_2$ with $P_1 \xrightarrow{\overline{a}\langle R\rangle} P_1'$ and $P_2 \xrightarrow{a(x)} P_2'$. Hence $P_1 \equiv \overline{a}\langle R\rangle \;\; P_1'$ and $P_2 \equiv a(x).\,P_2'$. As such, $P' \equiv P_1' \;\; P_2'\{R/x\}$. We know that P is a closed process; hence, both P_1' and R are closed processes, and P_2' is an open process such that $\mathrm{fn}(P_2') = \{x\}$. Then the process P' is closed since it is equivalent to the process P_2' where all the free occurrences of the name x has been replaced with the closed process R. □

Remark 5.1. *Notice that since in* Ho$^{-\mathrm{f}}$ *there is no restriction the only binder in the language is the input prefix. Therefore, within a closed process P, the only non closed processes in P are those occurring behind an input prefix, where they cannot evolve. By considering closed processes and restricting ourselves to reductions then α-conversion is not necessary.*

As before, the internal runs of a process are given by sequences of *reductions*. Given a process P, its reductions in the alternative LTS $P \longmapsto P'$ are defined as $P \overset{\tau}{\longmapsto} P'$. We denote with $\longmapsto^*$ the reflexive and transitive closure of $\longmapsto$. We use $P \not\longmapsto$ to denote that there is no P' such that $P \longmapsto P'$.

Given a process P, we shall use P_α to denote the result of applying the standard alpha-conversion without name captures over P.

Lemma 5.6. *Let P be a closed* $\mathrm{Ho}^{-\mathrm{f}}$ *process. Then, $P \longrightarrow P'$ iff $P \longmapsto P''$ and $P'' \equiv P'_\alpha$, for some P' in* $\mathrm{Ho}^{-\mathrm{f}}$*.*

Proof. The "if" direction follows easily from Remark 5.1. The "only if" direction is straightforward by observing that since P is a closed process, P'' is one of the possible evolutions of P in $\longrightarrow$. □

Corollary 5.2. *Let P be a closed* $\mathrm{Ho}^{-\mathrm{f}}$ *process. If $P \longmapsto P'$ then P' is a closed process in* $\mathrm{Ho}^{-\mathrm{f}}$*.*

Proof. Straightforward from Lemma 5.5 and Lemma 5.6. □

Corollary 5.3. *Let P be a closed* $\mathrm{Ho}^{-\mathrm{f}}$ *process. $P \not\longrightarrow$ iff $P \not\longmapsto$.*

Proof. Straightforward from Lemma 5.6. □

Remark 5.2. *The encoding of a Minsky machine presented in Section 5.3 is a closed process. Hence, all the results in that section hold for the LTS in Figure 5.3 as well.*

The *alphabet* of an $\mathrm{Ho}^{-\mathrm{f}}$ process is defined as follows:

Definition 5.8 (Alphabet of a process)**.** *Let P be a* $\mathrm{Ho}^{-\mathrm{f}}$ *process. The* alphabet of P*, denoted $\mathcal{A}(P)$, is inductively defined as:*

$$\mathcal{A}(\mathbf{0}) = \emptyset \qquad \mathcal{A}(P \quad Q) = \mathcal{A}(P) \cup \mathcal{A}(Q) \qquad \mathcal{A}(x) = \{x\}$$

$$\mathcal{A}(a(x).\,P) = \{a, x\} \cup \mathcal{A}(P) \qquad \mathcal{A}(\overline{a}\langle P\rangle) = \{a\} \cup \mathcal{A}(P)$$

The following proposition can be shown for the alternative LTS because it does not consider alpha-conversion. As a matter of fact, had we considered open processes, we would have required α-conversion. In such a case, the inclusion $\mathcal{A}(P'_2\{{}^{R}\!/\!{}_{x}\}) \subseteq \mathcal{A}(P'_2) \cup \mathcal{A}(R)$ would no longer hold. This is because by using α-conversion during substitution some new variables could be added to the alphabet.

Proposition 5.4. *Let P and P' be closed* $\mathrm{Ho}^{-\mathrm{f}}$ *processes. If $P \longmapsto P'$ then $\mathcal{A}(P') \subseteq \mathcal{A}(P)$.*

Proof. We proceed by a case analysis on the rule used to infer $\longmapsto$. We thus have four cases:

Case Tau1 Then $P = P_1 \quad P_2$ with $P_1 \overset{\overline{a}\langle R\rangle}{\longmapsto} P'_1$ and $P_2 \overset{a(x)}{\longmapsto} P'_2$. Hence $P_1 \equiv \overline{a}\langle R\rangle \quad P'_1$, $P_2 \equiv a(x).\,P'_2$, and $P' = P'_1 \quad P'_2\{{}^{R}\!/\!{}_{x}\}$. By Definition 5.8 we have that $\mathcal{A}(P_1) = \{a\} \cup \mathcal{A}(P'_1) \cup \mathcal{A}(R)$ and hence $\mathcal{A}(P'_1) \subseteq \mathcal{A}(P_1)$. Also by Definition 5.8 we have $\mathcal{A}(P_2) = \{a, x\} \cup \mathcal{A}(P'_2)$. Now, the process R is closed: therefore, during substitution, no variable can be captured. Hence, α-conversion is not needed, and we have $\mathcal{A}(P'_2\{{}^{R}\!/\!{}_{x}\}) \subseteq \mathcal{A}(P'_2) \cup \mathcal{A}(R)$. The result then follows.

Case Tau2 Similarly as for Tau1.

Case Act1 Then $P \equiv P_1 \;\; P_2$, $P' \equiv P'_1 \;\; P_2$, and $P_1 \xmapsto{\alpha} P'_1$. We then have $\mathcal{A}(P'_1) \subseteq \mathcal{A}(P_1)$ by using one of the above cases. By noting that $\mathcal{A}(P'_1) \cup \mathcal{A}(P_2) \subseteq \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$, the thesis holds.

Case Act2 Similarly as for Act1.

□

Fact 5.2. *The LTS for* $\mathrm{Ho}^{-\mathrm{f}}$ *given in Figure 5.3 is finitely branching.*

5.4.3 Termination is Decidable in $\mathrm{Ho}^{-\mathrm{f}}$

Here we prove that termination is decidable in $\mathrm{Ho}^{-\mathrm{f}}$. The crux of the proof consists in finding an upper bound for a process and its derivatives. This is possible in $\mathrm{Ho}^{-\mathrm{f}}$ because of the limited structure allowed in output actions.

We proceed as follows. First we define a notion of *normal form* for $\mathrm{Ho}^{-\mathrm{f}}$ processes. We then characterize an upper bound for the derivatives of a given process, and define an ordering over them. This ordering is then shown to be a wqo that is strongly compatible with respect to the LTS of $\mathrm{Ho}^{-\mathrm{f}}$ given in Section 5.4.2. The decidability result is then obtained by resorting to the theory of well-structured transition systems introduced in Section 5.4.1.

Definition 5.9 (Normal Form). *Let* $P \in \mathrm{Ho}^{-\mathrm{f}}$. *P is in* normal form *iff*

$$P = \prod_{k=1}^{l} x_k \;\; \prod_{i=1}^{m} a_i(y_i).\, P_i \;\; \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle$$

where each P_i *and* P'_j *are in normal form.*

Lemma 5.7. *Every process* $P \in \mathrm{Ho}^{-\mathrm{f}}$ *is structurally congruent to a normal form.*

Proof. By induction on the structure of P. The base cases are when $P = \mathbf{0}$ and when $P = x$, and are immediate. Cases $P = \overline{a}\langle Q \rangle$ and $P = a(x).\, Q$ follow by applying the inductive hypothesis on Q. For the case $P = P_1 \;\; P_2$, we apply the inductive hypothesis twice and we obtain that

$$P_1 \equiv \prod_{k=1}^{l} x_k \;\; \prod_{i=1}^{m} a_i(y_i).\, P_i \;\; \prod_{j=1}^{n} \overline{b_j}\langle P_j \rangle \quad \text{and} \quad P_2 \equiv \prod_{k=1}^{l'} x_k \;\; \prod_{i=1}^{m'} a'_i(y'_i).\, P'_i \;\; \prod_{j=1}^{n'} \overline{b'_j}\langle P'_j \rangle\,.$$

It is then easy to see that $P_1 \;\; P_2$ is structurally congruent to a normal form, as desired. □

We now define an ordering over normal forms. Intuitively, a process is larger than another if it has more parallel components.

Definition 5.10 (Relation $\preceq$). *Let $P, Q \in \mathrm{Ho}^{-\mathrm{f}}$. We write $P \preceq Q$ iff there exist $x_1 \ldots x_l$, $P_1 \ldots P_m$, $P'_1 \ldots P'_n$, $Q_1 \ldots Q_m$, $Q'_1 \ldots Q'_n$, and R such that*

$$\begin{array}{rcl} P & \equiv & \prod_{k=1}^{l} x_k \;\; \prod_{i=1}^{m} a_i(y_i).\, P_i \;\; \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle \\ Q & \equiv & \prod_{k=1}^{l} x_k \;\; \prod_{i=1}^{m} a_i(y_i).\, Q_i \;\; \prod_{j=1}^{n} \overline{b_j}\langle Q'_j \rangle \;\; R \end{array}$$

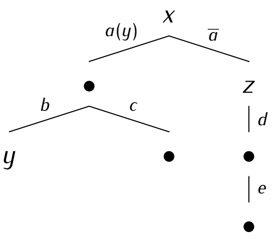
with $P_i \preceq Q_i$ and $P'_j \preceq Q'_j$, for $i \in [1..m]$ and $j \in [1..n]$.

The normal form of a process can be intuitively represented in a tree-like manner. More precisely, given the process in normal form

$$P = \prod_{k=1}^{l} x_k \;\; \prod_{i=1}^{m} a_i(y_i).\, P_i \;\; \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle$$

we shall decree its associated tree to have a root node labeled $x_1, \ldots, x_k$. This root node has $m+n$ children, corresponding to the trees associated to processes $P_1, \ldots, P_m$ and $P'_1, \ldots, P'_m$; the outgoing edges connecting the root node and the children are labeled $a_1(y_1), \ldots, a_m(y_m)$ and $\overline{b_1}, \ldots, \overline{b_n}$.

Example 5.1. *Process $P = x \;\; a(y).\, (b.\, y \;\; c) \;\; \overline{a}\langle z \;\; d.\, e \rangle$ has the following tree representation:*



This intuitive representation of processes in normal form as trees will be useful to reason about the structure of $\mathrm{Ho}^{-\mathrm{f}}$ terms. We begin by defining the *depth* of a process. Notice that such a depth corresponds to the maximum depth of its tree representation.

Definition 5.11 (Depth). *Let $P = \prod_{k=1}^{l} x_k \;\; \prod_{i=1}^{m} a_i(y_i).\, P_i \;\; \prod_{j=1}^{n} \overline{b_j}\langle P'_j \rangle$ be a $\mathrm{Ho}^{-\mathrm{f}}$ process in normal form. The* depth *of P is given by*

$$\mathrm{depth}(P) = \max\{1 + \mathrm{depth}(P_i), 1 + \mathrm{depth}(P'_j) \mid i \in [1..m] \wedge j \in [1..n]\}.$$

Given a natural number n and a process P, the set $\mathcal{P}_{P,n}$ contains all those processes in normal form that can be built using the alphabet of P and whose depth is at most n.

Definition 5.12. *Let n be a natural number and $P \in \mathrm{Ho}^{-\mathrm{f}}$. We define the set $\mathcal{P}_{P,n}$ as follows:*

$$\begin{array}{rll} \mathcal{P}_{P,n} = & \{Q \mid & Q \equiv \prod_{k \in K} x_k \;\; \prod_{i \in I} a_i(y_i).\, Q_i \;\; \prod_{j \in J} \overline{b_j}\langle Q'_j \rangle \\ & & \wedge\ \mathcal{A}(Q) \subseteq \mathcal{A}(P) \\ & & \wedge\ Q_i, Q'_j \in \mathcal{P}_{P,n-1} \;\; \forall i \in I, j \in J\} \end{array}$$

where $\mathcal{P}_{P,0}$ contains processes that are built out only of variables in $\mathcal{A}(P)$.

As it will be shown later, the set of all derivatives of P is a subset of $\mathcal{P}_{P,2\cdot\mathrm{depth}(P)}$.

When compared to processes in languages such as CCS, higher-order processes have a more complex structure. This is because, by virtue of reductions, an arbitrary process can take the place of possibly several occurrences of a single variable. As a consequence, the depth of (the syntax tree of) a process cannot be determined (or even approximated) before its execution: it can vary arbitrarily along reductions. Crucially, in Ho$^{-\mathrm{f}}$ it is possible to bound such a depth. Our approach is the following: rather than solely depending on the depth of a process, we define measures on the relative position of variables within a process. Informally speaking, such a position will be determined by the number of prefixes guarding a variable. Since variables are allowed only at the top level of the output objects, their relative distance will remain invariant during reductions. This allows to obtain a bound on the structure of Ho$^{-\mathrm{f}}$ processes. Finally, it is worth stressing that even if the same notions of normal form, depth, and distance can be defined for HOcore, a finite upper bound for such a language does not exist.

We first define the maximum distance between a variable and its binder.

Definition 5.13. *Let* $P = \prod_{k\in K} x_k \;\; \prod_{i\in I} a_i(y_i).\,P_i \;\; \prod_{j\in J} \overline{b_j}\langle P'_j\rangle$ *be a* Ho$^{-\mathrm{f}}$ *process in normal form. We define the* maximum distance of P *as:*

$$\mathrm{maxDistance}(P) = \max\{\mathrm{maxDist}_{y_i}(P_i),\\ \mathrm{maxDistance}(P_i), \mathrm{maxDistance}(P'_j) \mid i \in I, j \in J\}$$

where

$$\mathrm{maxDist}_x(P) = \begin{cases} 1 & \textit{if } P = x,\\ 1 + \mathrm{maxDist}_x(P_z) & \textit{if } P = a(z).\,P_z \wedge\ x \neq z,\\ 1 + \mathrm{maxDist}_x(P') & \textit{if } P = \overline{a}\langle P'\rangle,\\ \max\{\mathrm{maxDist}_x(R), \mathrm{maxDist}_x(Q)\} & \textit{if } P = R \;\; Q,\\ 0 & \textit{otherwise.} \end{cases}$$

Lemma 5.8 (Properties of maxDistance). *Let P be a closed* Ho$^{-\mathrm{f}}$ *process. It holds that:*

1. $\mathrm{maxDistance}(P) \leq \mathrm{depth}(P)$

2. *For every Q such that $P \longmapsto Q$,* $\mathrm{maxDistance}(Q) \leq \mathrm{maxDistance}(P)$.

Proof. Part (1) is immediate from Definitions 5.11 and 5.13. Part (2) follows by a case analysis on the rule used to infer $\longmapsto$. We focus in the case Tau1: the other cases are similar or simpler. We then have that $P = P_1 \;\; P_2$ with $P_1 \overset{\overline{a}\langle S\rangle}{\longmapsto} T$ and $P_2 \overset{a(x)}{\longmapsto} R$. Hence $P_1 \equiv \overline{a}\langle S\rangle \;\; T$ and

$P_2 \equiv a(x).\, R$. We then have that $P \equiv \overline{a}\langle S\rangle \quad T \quad a(x).\, R$ and $Q \equiv R\{^{S}\!/x\} \quad T$. Applying Definition 5.13 in both processes, we obtain

$$\begin{aligned} \mathsf{maxDistance}(P) &= \max\{\mathsf{maxDistance}(S), \mathsf{maxDist}_x(R), \\ &\qquad \mathsf{maxDistance}(R), \mathsf{maxDistance}(T)\} \\ \mathsf{maxDistance}(Q) &= \max\{\mathsf{maxDistance}(R\{^{S}\!/x\}), \mathsf{maxDistance}(T)\}\,. \end{aligned}$$

We can disregard the contribution of $\mathsf{maxDistance}(T)$, since it does not participate in the synchronization. We then focus on determining $\mathsf{maxDistance}(R\{^{S}\!/x\})$. Notice that the only way in which the value of $\mathsf{maxDistance}(R\{^{S}\!/x\})$ could be greater than that of $\mathsf{maxDistance}(R)$ is if S involves some free variables that get captured by an (input) binder in R by virtue of the substitution. Since S is a closed process, it has no free variables, and this capture is not possible. Consequently, we have

$$\mathsf{maxDistance}(R\{^{S}\!/x\}) \leq \max\{\mathsf{maxDistance}(R), \mathsf{maxDistance}(S)\}$$

and the thesis holds. □

We now define the maximum depth of processes that can be communicated. Notice that the continuations of inputs are considered as along reductions they could become communication objects themselves:

Definition 5.14. *Let $P = \prod_{k\in K} x_k \quad \prod_{i\in I} a_i(y_i).\, P_i \quad \prod_{j\in J} \overline{b_j}\langle P'_j\rangle$ be a* $\mathsf{Ho}^{-\mathsf{f}}$ *process in normal form. We define the* maximum depth *of a process that can be communicated (*$\mathsf{maxDepCom}(P)$*) in P as:*

$$\mathsf{maxDepCom}(P) = \max\{\mathsf{maxDepCom}(P_i), \mathsf{depth}(P'_j) \mid i \in I, j \in J\}\ .$$

Lemma 5.9 (Properties of maxDepCom). *Let P be a closed* $\mathsf{Ho}^{-\mathsf{f}}$ *process. It holds that:*

1. $\mathsf{maxDepCom}(P) \leq \mathsf{depth}(P)$
2. *For every Q such that $P \longmapsto Q$,* $\mathsf{maxDepCom}(Q) \leq \mathsf{maxDepCom}(P)$.

Proof. Part (1) is immediate from Definitions 5.11 and 5.14. Part (2) follows by a case analysis on the rule used to infer $\longmapsto$. Again, we focus in the case Tau1: the other cases are similar or simpler. We then have that $P = P_1 \quad P_2$ with $P_1 \xrightarrow{\overline{a}\langle S\rangle} T$ and $P_2 \xrightarrow{a(x)} R$. Hence $P_1 \equiv \overline{a}\langle S\rangle \quad T$ and $P_2 \equiv a(x).\, R$. We then have that $P \equiv \overline{a}\langle S\rangle \quad T \quad a(x).\, R$ and $Q \equiv R\{^{S}\!/x\} \quad T$. Applying Definition 5.14 in both processes, we obtain

$$\begin{aligned} \mathsf{maxDepCom}(P) &= \max\{\mathsf{maxDepCom}(T), \mathsf{maxDepCom}(S), \\ &\qquad \mathsf{maxDepCom}(R), \mathsf{depth}(S)\} \\ \mathsf{maxDepCom}(Q) &= \max\{\mathsf{maxDepCom}(T), \mathsf{maxDepCom}(R\{^{S}\!/x\})\}\,. \end{aligned}$$

We now focus on analyzing the influence a substitution has on communicated objects. Since variables can occur in output objects, the sensible case to check is if x appears inside some communication object in R. It is worth noticing that x is a variable that becomes free only as a result of the input on a, which consumes its binder. We thus have two cases:

1. *There are no communication objects in R with occurrences of x.* Then, S will only occur at the top level in $R\{{}^{S}\!/\!{}_{x}\}$. Since $\mathsf{depth}(S)$ was already taken into account when determining $\mathsf{maxDepCom}(P)$, we then have that $\mathsf{maxDepCom}(R\{{}^{S}\!/\!{}_{x}\}) \leq \mathsf{maxDepCom}(P)$, and the thesis holds.

2. *Some communication objects in R have occurrences of x.* Then, R contains as subprocess an output message $\overline{b}\langle P_x \rangle$ where, for some $k > 0$ and a closed process S', process $P_x \equiv \prod^k x \;\; S'$. Process $\overline{b}\langle P_x\{{}^{S}\!/\!{}_{x}\}\rangle$ then occurs in $R\{{}^{S}\!/\!{}_{x}\}$. Clearly, an eventual increase of $\mathsf{maxDepCom}(Q)$ depends on the depth of $P_x\{{}^{S}\!/\!{}_{x}\}$. We have that $\mathsf{depth}(P_x\{{}^{S}\!/\!{}_{x}\}) = \max(\mathsf{depth}(S), \mathsf{depth}(S'))$. Since both $\mathsf{depth}(S)$ and $\mathsf{depth}(S')$ were considered when determining $\mathsf{maxDepCom}(P)$, we conclude that $\mathsf{maxDepCom}(R\{{}^{S}\!/\!{}_{x}\})$ can be at most equal to $\mathsf{maxDepCom}(P)$, and so the thesis holds.

□

Generalizing Lemmas 5.8 and 5.9 we obtain:

Corollary 5.4. *Let P be a closed Ho^{-f} process. For every Q such that $P \longmapsto^* Q$, it holds that:*

1. $\mathsf{maxDistance}(Q) \leq \mathsf{depth}(P)$

2. $\mathsf{maxDepCom}(Q) \leq \mathsf{depth}(P)$.

We are interested in characterizing the derivatives of a given process P. We shall show that they are over-approximated by means of the set $\mathcal{P}_{P,2\cdot\mathsf{depth}(P)}$. We will investigate the properties of the relation $\preceq$ on such an approximation; such properties will also hold for the set of derivatives.

Definition 5.15. *Let $P \in \mathsf{Ho}^{-f}$. Then we define* $\mathsf{Deriv}(P) = \{Q \mid P \longmapsto^* Q\}$

The following results hold because of the limitations we have imposed on the output actions for Ho^{-f} processes.

Lemma 5.10. *Let P, Q be Ho^{-f} processes such that $\mathcal{A}(Q) \subseteq \mathcal{A}(P)$. $Q \in \mathcal{P}_{P,n}$ if and only if* $\mathsf{depth}(Q) \leq n$.

Proof. The "if" direction is straightforward by definition of $\mathcal{P}_{P,n}$ (Definition 5.12).

For the "only if" direction we proceed by induction on n. If $n = 0$ then $Q = \mathbf{0}$ or $Q = x_1 \ \ \cdots \ \ x_k$. In both cases, since $\mathcal{A}(Q) \subseteq \mathcal{A}(P)$, Q is easily seen to be in $\mathcal{P}_{P,0}$. If $n > 0$ then

$$Q = \prod_{k \in K} x_k \ \ \prod_{i \in I} a_i(y_i).\, Q_i \ \ \prod_{j \in J} \overline{b_j}\langle Q'_j \rangle$$

where, for every $i \in I$ and $j \in J$, both $\mathrm{depth}(Q_i) \leq \mathrm{depth}(Q) \leq n-1$ and $\mathrm{depth}(Q'_j) \leq \mathrm{depth}(P) \leq n-1$. By inductive hypothesis, each Q_i and Q'_j is in $\mathcal{P}_{P,n-1}$. Then, by Definition 5.12, $Q \in \mathcal{P}_{P,n}$ and we are done. □

Proposition 5.5. *Let P be a* $\mathrm{Ho}^{-\mathrm{f}}$ *process. Suppose, for some R and n, that $P \in \mathcal{P}_{R,n}$. For every Q such that $P \longmapsto Q$, it holds that $Q \in \mathcal{P}_{R,2\cdot n}$.*

Proof. We proceed by case analysis on the rule used to infer $\longmapsto$. We focus on the case such a rule is Tau1; the remaining cases are similar or simpler. Recall that by Lemmas 5.8(1) and 5.9(1) the maximum distance between an occurrence of a variable and its binder is bounded by $\mathrm{depth}(P)$. By Definition 5.12 any process that can be communicated in P is in $\mathcal{P}_{R,n-1}$ and its maximum depth is also bounded by $\mathrm{depth}(P)$ —which, in turn, by Lemma 5.10, is bounded by n. The deepest position for a variable is when it is a leaf in the tree associated to the normal form of P. That is, when its depth is exactly $\mathrm{depth}(P)$. If in that position we place a process in $\mathcal{P}_{R,n-1}$ — whose depth is also $\mathrm{depth}(P)$ — then it is easy to see that (the associated tree of) Q has a depth of $2 \cdot \mathrm{depth}(P)$, which is bounded by $2 \cdot n$. Hence, by Lemma 5.10, Q is in $\mathcal{P}_{R,2\cdot n}$. □

The lemma below generalizes Proposition 5.5 to a sequence of reductions.

Lemma 5.11. *Let P be a* $\mathrm{Ho}^{-\mathrm{f}}$ *process. Suppose, for some R and n, that $P \in \mathcal{P}_{R,n}$. For every Q such that $P \longmapsto^{*} Q$, it holds that $Q \in \mathcal{P}_{R,2\cdot n}$.*

Proof. The proof proceeds by induction on k, the length of $\longmapsto^{*}$, exploiting Proposition 5.5. The base case is when $k = 1$, and it follows by Proposition 5.5. For the inductive step we assume $k > 1$, so we have that, for some P', $P \longmapsto^{*} P' \longmapsto Q$ where the sequence from P to P' has lenght $k-1$. By induction hypothesis we know that $P' \in \mathcal{P}_{R,2\cdot n}$. We then proceed by a case analysis on the rule used to infer $P' \longmapsto Q$. As usual, we content ourselves with illustrating the case Tau1; the other ones are similar or simpler. We then have that $P' = P_1 \ \ P_2$ with $P_1 \xmapsto{\overline{a}\langle T \rangle} S$ and $P_2 \xmapsto{a(x)} V$. Hence $P_1 \equiv \overline{a}\langle T \rangle \ \ S$ and $P_2 \equiv a(x).\, V$. We then have that $P' \equiv \overline{a}\langle T \rangle \ \ S \ \ a(x).\, V$ and $Q \equiv V\{^{T}\!/x\} \ \ S$.

By Corollary 5.4 the maximum distance between x and its binder $a(x)$ is $\mathrm{depth}(P)$, which in turn is bounded by n (Lemma 5.10). Moreover, the maximum depth of T is bounded by $\mathrm{maxDepCom}(P)$; by Corollary 5.4, $\mathrm{depth}(P) \leq n$. Therefore, the overall depth of process Q is $2 \cdot \mathrm{depth}(P)$. Hence, and by using Lemma 5.10, $Q \in \mathcal{R}_{P,2\cdot n}$, as wanted. □

Corollary 5.5. *Let $P \in$ Ho^{-f}. Then* Deriv$(P) \subseteq \mathcal{P}_{P,2\cdot\mathrm{depth}(P)}$.

We are now ready to prove that relation $\preceq$ is a wqo. We begin by showing that it is a quasi-order.

Proposition 5.6. *The relation $\preceq$ is a quasi-order.*

Proof. We need to show that $\preceq$ is both reflexive and transitive. From Definition 5.9, reflexivity is immediate.

Transitivity implies proving that, given processes P, Q, and R such that $P \preceq Q$ and $Q \preceq R$, $P \preceq R$ holds. We proceed by induction on $k = \mathrm{depth}(P)$. If $k = 0$ then we have that $P = x_1 \; \cdots \; x_k$. Since $P \preceq Q$, we have that $Q = x_1 \; \cdots \; x_k \; S$. and that $R = x_1 \; \cdots \; x_k \; S'$, for some S, S' such that $S \preceq S'$. By Definition 5.10, the thesis follows. Now suppose $k > 0$. By Definition 5.9 and by hypothesis we have the following:

$$\begin{aligned}
P &= \prod_{k=1}^{l} x_k \; \prod_{i=1}^{m} a_i(y_i).\, P_i \; \prod_{j=1}^{n} \overline{b_j}\langle P'_j\rangle \\
Q &= \prod_{k=1}^{l} x_k \; \prod_{i=1}^{m} a_i(y_i).\, Q_i \; \prod_{j=1}^{n} \overline{b_j}\langle Q'_j\rangle \;\; S \\
R &= \prod_{k=1}^{l} x_k \; \prod_{i=1}^{m} a_i(y_i).\, R_i \; \prod_{j=1}^{n} \overline{b_j}\langle R'_j\rangle \;\; S \;\; T.
\end{aligned}$$

with $P_i \preceq Q_i$, $P'_j \preceq Q'_j$, $Q_i \preceq R_i$, and $Q'_j \preceq R'_j$ ($i \in I, j \in J$). Since P_i, P'_j, Q_i, Q'_j, R_i, and R'_j have depth $k-1$, by inductive hypothesis $P_i \preceq R_i$ and $P'_j \preceq R'_j$. By Definition 5.10, the thesis follows and we are done. □

We are now in place to state that $\preceq$ is a wqo.

Theorem 5.3 (Well-quasi-order)**.** *Let $P \in$ Ho^{-f} be a closed process and $n \geq 0$. The relation $\preceq$ is a well-quasi-order over $\mathcal{P}_{P,n}$.*

Proof. The proof is by induction on n.

- Let $n = 0$. Then $\mathcal{P}_{P,0}$ contains processes made up only of variables taken from $\mathcal{A}(P)$. The equality on finite sets is a well-quasi-ordering; by Lemma 5.4 (Higman's Lemma) also $=_*$ is a well quasi-ordering: it corresponds to the ordering $\preceq$ on processes containing only variables.

- Let $n > 0$. Take an infinite sequence of processes $s = P_1, P_2, \ldots, P_l, \ldots$ with $P_l \in \mathcal{P}_{P,n}$. We shall show that the thesis holds by means of successive filterings of the normal forms of the processes in s. By Lemma 5.7 there exist K_l, I_l and J_l such that

$$P_l \equiv \prod_{k \in K_l} x_k \; \prod_{i \in I_l} a_i(y_i).\, P^l_i \; \prod_{j \in J_l} \overline{b_j}\langle P'^l_j\rangle$$

with P^l_i and $P'^l_j \in \mathcal{P}_{P,n-1}$. Hence each P_l can be seen as composed of 3 finite sequences: (i) $x_1 \ldots x_k$, (ii) $a_1(y_1).\, P^l_1 \ldots a_i(y_i).\, P^l_i$, and (iii) $\overline{b_1}\langle P'^l_1\rangle \ldots \overline{b_j}\langle P'^l_j\rangle$. We note that the first sequence is composed of variables from the finite set $\mathcal{A}(P)$ whereas the other two sequences are composed by elements in $\mathcal{A}(P)$ and $\mathcal{P}_{P,n-1}$. Since we have an infinite sequence of $\mathcal{A}(P)^*$, as $\mathcal{A}(P)$ is finite, by Proposition 5.3 and Lemma 5.4 we have that $=_*$ is a wqo over $\mathcal{A}(P)^*$.

By inductive hypothesis, we have that $\preceq$ is a wqo on $\mathcal{P}_{P,n-1}$, hence by Lemma 5.4 relation $\preceq_*$ is a wqo on $\mathcal{P}^*_{P,n-1}$. We start filtering out s by making the finite sequences $x_1 \ldots x_k$ increasing with respect to $=_*$; let us call this subsequence t. Then we filter out t, by making the finite sequence $a_1(y_1).\, P^l_1 \ldots a_i(y_i).\, P^l_i$ increasing with respect to both $=_*$ and $\preceq_*$. This is done in two steps: first, by considering the relation $=_*$ on the subject of the actions (recalling that $a_i, y_i \in \mathcal{A}(P)$), and second, by applying another filtering to the continuation using the inductive hypothesis. It is worth remarking that in the first step we do not consider symbols of the alphabet but pairs of symbols. Since the set of pairs on a finite set is still finite, we know by Higman's Lemma that $=_*$ is a wqo on the set of sequences of pairs (a_i, y_i).

For the sequence of outputs $\overline{b_1}\langle P'^l_1\rangle \ldots \overline{b_j}\langle P'^l_j\rangle$ this is also done in two steps: the subject of the outputs are ordered with respect to $=_*$ and the objects of the output action are ordered with respect to $\preceq_*$ using the inductive hypothesis.

At the end of the process we obtain an infinite subsequence of s that is ordered with respect to $\preceq$.

□

The last thing to show is that the well-quasi-ordering $\preceq$ is strongly compatible with respect to the LTS in Figure 5.3. We need some auxiliary results first.

Lemma 5.12. *Let P, P', Q and Q' be* $\mathrm{Ho}^{-\mathrm{f}}$ *processes in normal form such that $P \preceq P'$ and $Q \preceq Q'$. Then it holds that $P \quad Q \preceq P' \quad Q'$.*

Proof. Immediate from the definitions of normal form and $\preceq$ (Definitions 5.9 and 5.10). □

Lemma 5.13. *Let $P, P', Q,$ and Q' be* $\mathrm{Ho}^{-\mathrm{f}}$ *processes in normal form such that $P \preceq P'$ and $Q \preceq Q'$. Then it holds that $P\{Q/x\} \preceq P'\{Q'/x\}$.*

Proof. By induction on the structure of P.

1. Cases $P = \mathbf{0}$ and $P = y$, for some $y \neq x$: Immediate.

2. Case $P = x$. Then $P' = x \quad N$, for some process N. We have that $P\{Q/x\} = Q$ and that $P'\{Q'/x\} = Q' \quad N\{Q'/x\}$. Since $Q \preceq Q'$ the thesis follows.

3. Case $P = a(y).\,R$. Then $P' = a(y).\,R' \quad N$, for some process N. Since by hypothesis $P \preceq P'$, then $R \preceq R'$. We then have that $P\{^{Q}/x\} = a(y).\,R\{^{Q}/x\}$ and that $P'\{^{Q'}/x\} = a(y).\,R'\{^{Q}/'\}x \quad N\{^{Q'}/x\}$. By inductive hypothesis we obtain that $R\{^{Q}/x\} \preceq R'\{^{Q'}/x\}$, and the thesis follows.

4. Case $P = \overline{a}\langle R\rangle$: Similar to (3).

5. Case $P = R \quad S$. Then $P' = R' \quad S' \quad N$, for some process N, with $R \preceq R'$ and $S \preceq S'$. We then have that $P\{^{Q}/x\} = R\{^{Q}/x\} \quad S\{^{Q}/x\}$ and $P'\{^{Q'}/x\} = R'\{^{Q'}/x\} \quad S'\{^{Q'}/x\} \quad N\{^{Q'}/x\}$. The thesis then follows by inductive hypothesis and Lemma 5.12.

□

Theorem 5.4 (Strong Compatibility). *Let $P, Q, P' \in \mathrm{Ho}^{-\mathrm{f}}$. If $P \preceq Q$ and $P \longmapsto P'$ then there exists Q' such that $Q \longmapsto Q'$ and $P' \preceq Q'$.*

Proof. By case analysis on the rule used to infer reduction $P \longmapsto P'$. We content ourselves with illustrating the case derived from the use of rule Tau1; the other ones are similar or simpler. We then have that $P = P' \quad P''$ with $P' \xmapsto{\overline{a}\langle P_1\rangle} N$ and $P'' \xmapsto{a(y)} P_2$. Hence, $P \equiv \overline{a}\langle P_1\rangle \quad a(y).\,P_2 \quad N$. Since by hypothesis $P \preceq Q$, we obtain a similar structure for Q. Indeed, $Q \equiv \overline{a}\langle Q_1\rangle \quad a(y).\,Q_2 \quad N'$ with $P_1 \preceq Q_1$, $P_2 \preceq Q_2$, and $N \preceq N'$.

Now, if $P \longmapsto P' \equiv P_2\{^{P_1}/y\} \quad N$ then also $Q \longmapsto Q' \equiv Q_2\{^{Q_1}/y\} \quad N'$. By Lemma 5.13 we have $P_2\{^{P_1}/y\} \preceq Q_2\{^{Q_1}/y\}$; using this and the hypothesis the thesis follows. □

Theorem 5.5. *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$ be a closed process. The transition system $(\mathrm{Deriv}(P), \longmapsto, \preceq)$ is a finitely branching well-structured transition system with strong compatibility, decidable $\preceq$, and computable $Succ$.*

Proof. The transition system of $\mathrm{Ho}^{-\mathrm{f}}$ is finitely branching (Fact 5.2). The fact that $\preceq$ is a well-quasi-order on $\mathrm{Deriv}(P)$ follows from Corollary 5.5 and Theorem 5.3. Strong compatibility follows from Theorem 5.4. □

We can now state the main technical result of the section.

Corollary 5.6. *Let $P \in \mathrm{Ho}^{-\mathrm{f}}$ be a closed process. Then, termination of P is decidable.*

Proof. This follows from Theorem 5.2, Theorem 5.5, and Corollary 5.3. □

5.5 On the Interplay of Fowarding and Passivation

The decidability of termination in Ho$^{-\mathrm{f}}$ presented in Section 5.4 provides compelling evidence on the fact that the limited forwarding entails a loss of expressive power for HOcore. It is therefore worth investigating alternatives for recovering such an expressive power while preserving the essence of limited forwarding.

In this section we examine one such alternatives. We analyze the consequences of extending Ho$^{-\mathrm{f}}$ with a *passivation* construct, an operator that allows to *suspend* the execution of a process at run time. As such, it comes in handy to represent scenarios of (dynamic) system reconfiguration, which are often indispensable in the specification of open, extensible systems such as component-based ones. Passivation has been considered by higher-order calculi such as the Kell calculus (Schmitt and Stefani, 2004) and Homer (Hildebrandt et al., 2004), and finds several applications (see, e.g., (Bundgaard et al., 2008)). Here we shall consider a passivation construct of the form $\widetilde{a}\{P\}$, which represents a *passivation unit* named a that contains a process P. The passivation unit is a *transparent locality*, in that there are no restrictions on the interactions between P and processes surrounding a. The execution of P can be *passivated* at an arbitrary time; this is represented by the evolution of $\widetilde{a}\{P\}$ into the nil process by means of an output action $\overline{a}\langle P\rangle$. Hence, the passivation of $\widetilde{a}\{P\}$ process might lead to a synchronization with any interacting input action on a.

We consider HoP$^{-\mathrm{f}}$, the extension of Ho$^{-\mathrm{f}}$ with a passivation construct as described above. The syntax extends as expected; for the sake of consistency, we notice that the process P in $\widetilde{a}\{P\}$ respects the limitation on the shape of output objects introduced for Ho$^{-\mathrm{f}}$. The LTS for HoP$^{-\mathrm{f}}$ is the same as that for Ho$^{-\mathrm{f}}$ in Section 5.2, extended with the two following rules which formalize the intuitions given before with respect to transparent localities and passivation, respectively:

$$\frac{P \xrightarrow{\alpha} P'}{\widetilde{a}\{P\} \xrightarrow{\alpha} \widetilde{a}\{P'\}}\ \textsc{Loc} \qquad \widetilde{a}\{P\} \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}\ \textsc{Pas}\,.$$

5.5.1 A Faithful Encoding of Minsky Machines into HoP$^{-\mathrm{f}}$

Here we investigate the expressiveness of HoP$^{-\mathrm{f}}$ by exhibiting an encoding of Minsky machines into HoP$^{-\mathrm{f}}$. Interestingly, unlike the encoding presented in Section 5.3, the encoding into HoP$^{-\mathrm{f}}$ is *faithful*. As such, in HoP$^{-\mathrm{f}}$ *both* termination and convergence are *undecidable* problems. Hence, it is fair to say that the passivation construct —even with the limitation on the shape of (output) processes— allows to recover the expressive power lost in restricting HOcore as Ho$^{-\mathrm{f}}$.

The encoding is given in Figure 5.4; we now give some intuitions on it. A register k with value m is represented by a passivation unit r_k that contains the encoding of number

$$\textsc{Register}\ r_k \qquad [r_k = n]_{\mathsf{M}} = \widetilde{r_k}\{\llparenthesis\ n\ \rrparenthesis_k\}$$

where

$$\llparenthesis\ n\ \rrparenthesis_k = \begin{cases} z_k.\,\overline{a_z} & \text{if } n = 0 \\ u_k.\,(\overline{a_1}\ \ a_2.\,\llparenthesis\ n-1\ \rrparenthesis_k) & \text{if } n > 0. \end{cases}$$

$$\textsc{Instructions}\ (i : I_i)$$

$$\begin{aligned}
[(i : \mathsf{INC}(r_k))]_{\mathsf{M}} \quad &= \quad !p_i.\,(r_k(x).\,(\overline{c_k}\langle x\rangle\ \ \widetilde{r_k}\{c_k(y).\,(\overline{a_p}\ \ u_k.\,(\overline{a_1}\ \ a_2.\,y))\}\ \ a_p.\,\overline{p_{i+1}})) \\
[(i : \mathsf{DECJ}(r_k, s))]_{\mathsf{M}} \quad &= \quad\ \ !p_i.\,(m(x).\,x \\
&\qquad\qquad \widetilde{d}\{\overline{u_k}\ \ a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2}\ \ \overline{p_{i+1}})\rangle\} \\
&\qquad\qquad \widetilde{s}\{\overline{z_k}\ \ a_z.\,\overline{m}\langle d(x).\,s(x).\,r_k(t).\,(\widetilde{r_k}\{z_k.\,\overline{a_z}\}\ \ \overline{p_s})\rangle\})
\end{aligned}$$

Figure 5.4: Encoding of Minsky machines into HoP$^{-\mathsf{f}}$.

m, denoted $\llparenthesis\ m\ \rrparenthesis_k$. In turn, $\llparenthesis\ m\ \rrparenthesis_k$ consists of a chain of m nested input prefixes on name u_k; it also contains other prefixes on a_1 and a_2 which are used for synchronization purposes during the execution of instructions. The encoding of zero is given by an input action on z_k that prefixes a trigger $\overline{a_z}$.

As expected, the encoding of an increment operation on the value of register k consists in the enlargement of the chain of nested input prefixes it contains. For that purpose, the content of passivation unit r_k is obtained with an input on r_k. We therefore need to recreate the passivation unit r_k with the encoding of the incremented value. Notice that we require an additional synchronization on c_k in order to "inject" such a previous content in a new passivation unit called r_k. This way, the chain of nested inputs in r_k can be enlarged while respecting the limitation on the shape of processes inside passivation units. As a result, the chain is enlarged by putting it behind some prefixes, and the next instruction can be invoked. This is done by a synchronization on name a_p.

The encoding of a decrement of the value of register k consists of an internal, exclusive choice implemented as two passivation units that execute in parallel: the first one, named d, implements the behavior for decrementing the value of a register, while the second one, named s, implements the behavior for performing the jump to some given instruction. Unlike the encoding of Minsky machines in Ho$^{-\mathsf{f}}$ presented in Section 5.3, this internal choice behaves faithfully with respect to the encoding instruction, i.e., the behavior inside d will only execute if the value in r_k is greater than zero, whereas the behavior inside s will only execute if that value is equal to zero. It is indeed a deterministic choice in that it is *not* the case that both an input prefix on u_k (which triggers the "decrement branch" defined by d) and one on z_k (which triggers the "jump branch" defined by s) are available at the same time; this is because of the way in which we encode numbers, i.e., as a chain of input prefixes. In addition to the passivation units, the encoding of decrement features a “manager” (implemented as a synchronization on m) that enables the behavior of the chosen passivation unit by placing it at the top-level, and consumes both s and d afterwards, thus leaving no residual processes after performing the instruction. In case the value of the register is equal to some $n > 0$, then a decrement is implemented by consuming the input prefixes on u_k and a_2 and the output prefix on a_1 through suitable synchronizations. It is worth noticing that these synchronizatons are only possible because the passivation units containing the encoding of n behave as transparent localities, and hence able to interact with its surrounding context. As a result, the encoding of $n-1$ remains inside r_k and the next instruction is invoked. In case the value of the register is equal to zero, the passivation unit r_k is consumed and recreated with the encoding of zero inside. The jump is then performed by invoking the respective instruction.

We are now ready to define the encoding of a configuration of a Minsky machine into $\mathrm{HoP}^{-\mathrm{f}}$.

Definition 5.16 (Encoding of Configurations). *Let N be a Minsky machine with registers $r_0 = m_0$, $r_1 = m_1$ and instructions $(1 : I_1), \ldots, (n : I_n)$. The encoding of a configuration (i, m_0, m_1) of N into* $\mathrm{HoP}^{-\mathrm{f}}$ *is defined by the encodings in Figure 5.4 as*

$$[\![(i, m_0, m_1)]\!]_{\mathrm{M}} = \overline{p_i} \;\; [\![r_0 = m_0]\!]_{\mathrm{M}} \;\; [\![r_1 = m_1]\!]_{\mathrm{M}} \;\; \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathrm{M}} \,,$$

assuming fresh, pairwise different names r_j, u_k, z_k, $p_1, \ldots, p_n$, (for $j \in \{0, 1\}$).

5.5.2 Correctness of the Encoding

We divide the proof of correctness into two properties: completeness (Lemma 5.14) and soundness (Lemma 5.15).

Lemma 5.14 (Completeness). *Let (i, m_0, m_1) be a configuration of a Minsky machine N. Then, if $(i, m_0, m_1) \longrightarrow_{\mathrm{M}} (i', m_0', m_1')$ then, for some finite j and a process P, it holds that $[\![(i, m_0, m_1)]\!]_{\mathrm{M}} \longrightarrow^{j} P \equiv [\![(i', m_0', m_1')]\!]_{\mathrm{M}}$.*

Proof. We proceed by a case analysis on the instruction performed by the Minsky machine. Hence, we distinguish three cases corresponding to the behaviors associated to rules M-Inc, M-Dec, and M-Jmp. Without loss of generality, we restrict our analysis to operations on register r_0.

Case M-Inc: We have a Minsky configuration (i, m_0, m_1) with $(i : \mathrm{INC}(r_0))$. By Definition 5.16, its encoding into $\mathrm{Ho}^{-\mathrm{f}}$ with passivation is as follows:

$$\begin{aligned} [\![(i, m_0, m_1)]\!]_{\mathrm{M}} &= \overline{p_i} \;\; [\![r_0 = m_0]\!]_{\mathrm{M}} \;\; [\![r_1 = m_1]\!]_{\mathrm{M}} \\ &\quad [\![(i : \mathrm{INC}(r_0))]\!]_{\mathrm{M}} \prod_{l=1..n, l \neq i} [\![(l : I_l)]\!]_{\mathrm{M}} \end{aligned}$$

After consuming the program counter p_i we have the following

$$\begin{aligned}[(i,m_0,m_1)]_{\mathrm{M}} \;\; &\longrightarrow \;\; \widetilde{r_0}\{\llparenthesis\, m_0 \,\rrparenthesis_0\} \;\; r_0(x).\,(\overline{c_0}\langle x\rangle \;\; \widetilde{r_0}\{c_0(y).\,(\overline{a_p} \;\; u_0.\,(\overline{a_1} \;\; a_2.\,y))\}) \\ &\qquad a_p.\,\overline{p_{i+1}} \;\; S = P_1\end{aligned}$$

where $S = [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{i=1}^{n}[(i : I_i)]_{\mathrm{M}}$ stands for the rest of the system. The only reduction possible at this point is the synchronization on r_0, which allows the content of the passivation unit r_0 to be communicated:

$$P_1 \longrightarrow \overline{c_0}\langle\llparenthesis\, m_0 \,\rrparenthesis_0\rangle \;\; \widetilde{r_0}\{c_0(y).\,(\overline{a_p} \;\; u_0.\,(\overline{a_1} \;\; a_2.\,y))\} \;\; a_p.\,\overline{p_{i+1}} \;\; S = P_2\,.$$

Now there is a synchronization on c_0, which allows to "inject" the encoding of value m_0 inside the passivation unit r_0 respecting the limitation of the language:

$$P_2 \longrightarrow \widetilde{r_0}\{(\overline{a_p} \;\; u_0.\,(\overline{a_1} \;\; a_2.\,\llparenthesis\, m_0 \,\rrparenthesis_0))\} \;\; a_p.\,\overline{p_{i+1}} \;\; S = P_3\,.$$

The only possible synchronization from P_3 is the one on a_p, which works as an acknowledgment signal, and allows to release the program counter for instruction $i+1$. By performing such a synchronization, and by the encoding of numbers, we obtain the following

$$P_3 \longrightarrow \widetilde{r_0}\{\llparenthesis\, m_0+1 \,\rrparenthesis_0\} \;\; \overline{p_{i+1}} \;\; S = P_4$$

It is then easy to see that $P_4 \equiv [(i+1, m_0+1, m_1)]_{\mathrm{M}}$, as desired.

Case M-Dec: We have a Minsky configuration (i, c, m_1) with $c > 0$ and $(i : \mathrm{DEC}(r_0, s))$. By Definition 5.16, its encoding into $\mathrm{Ho}^{-\mathrm{f}}$ with passivation is as follows:

$$\begin{aligned}[(i,c,m_1)]_{\mathrm{M}} \;\; &= \;\; \overline{p_i} \;\; [r_0 = c]_{\mathrm{M}} \;\; [r_1 = m_1]_{\mathrm{M}} \\ &\qquad [(i : \mathrm{DEC}(r_0,s))]_{\mathrm{M}} \;\; \prod_{l=1..n,\,l\neq i}[(l : I_l)]_{\mathrm{M}}\end{aligned}$$

We begin by consuming the program counter p_i, which leaves the content of $[(i : \mathrm{DEC}(r_0,s))]_{\mathrm{M}}$ exposed. Using the encoding of numbers we have the following:

$$\begin{aligned}[(i,c,m_1)]_{\mathrm{M}} \;\; &\longrightarrow \;\; \widetilde{r_0}\{u_0.\,(\overline{a_1} \;\; a_2.\,\llparenthesis\, c-1 \,\rrparenthesis_0)\} \;\; m(x).\,x \\ &\qquad \widetilde{d}\{\overline{u_0} \;\; a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \;\; \overline{p_{i+1}})\rangle\} \\ &\qquad \widetilde{s}\{\overline{z_0} \;\; a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s})\rangle\} \;\; S = P_1\end{aligned}$$

where $S = [r_1 = m_1]_{\mathrm{M}} \;\; \prod_{i=1}^{n}[(i : I_i)]_{\mathrm{M}}$ stands for the rest of the system. Notice that only reduction possible at this point is the synchronization on u_0, which signals the fact we are performing a decrement instruction. Such a synchronization enables one on a_1. After these two synchronizations we have

$$\begin{aligned}P_1 \;\; &\longrightarrow^2 \;\; \widetilde{r_0}\{a_2.\,\llparenthesis\, c-1 \,\rrparenthesis_0\} \;\; m(x).\,x \;\; \widetilde{d}\{\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \;\; \overline{p_{i+1}})\rangle\} \\ &\qquad \widetilde{s}\{\overline{z_0} \;\; a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s})\rangle\} \;\; S = P_2\,.\end{aligned}$$

Starting in P_2 the only reduction possible is due to the synchronization on m, which gives us the following:

$$\begin{aligned} P_2 \quad \longrightarrow \quad & \widetilde{r_0}\{a_2.\,(\!|\; c-1 \;|\!)_0\} \;\; s(x).\,d(x).\,(\overline{a_2} \;\; \overline{p_{i+1}}) \;\; \widetilde{d}\{\mathbf{0}\} \\ & \widetilde{s}\{\overline{z_0} \;\; a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s})\rangle\} \;\; S = P_3\,. \end{aligned}$$

In P_3 we have that passivation units s and d are consumed, thus we have:

$$P_3 \longrightarrow \widetilde{r_0}\{a_2.\,(\!|\; c-1 \;|\!)_0\} \;\; \overline{a_2} \;\; \overline{p_{i+1}} \;\; S = P_4\,.$$

At this point it is easy to see that, after a synchronization on a_2, we obtain

$$P_4 \longrightarrow\equiv [\![(i+1, c-1, m_1)]\!]_{\mathrm{M}}$$

as desired.

Case M-Jmp: We have a Minsky configuration $(i, 0, m_1)$ and $(i : \mathrm{DEC}(r_0, s))$. By Definition 5.16, its encoding into $\mathrm{Ho}^{-\mathrm{f}}$ with passivation is as follows:

$$\begin{aligned} [\![(i, 0, m_1)]\!]_{\mathrm{M}} \quad = \quad & \overline{p_i} \;\; [\![r_0 = 0]\!]_{\mathrm{M}} \;\; [\![r_1 = m_1]\!]_{\mathrm{M}} \\ & [\![(i : \mathrm{DEC}(r_0, s))]\!]_{\mathrm{M}} \prod_{l=1..n, l\neq i} [\![(l : I_l)]\!]_{\mathrm{M}}\,. \end{aligned}$$

We begin by consuming the program counter p_i, which leaves the content of $[\![(i : \mathrm{DEC}(r_0, s))]\!]_{\mathrm{M}}$ exposed. Using the encoding of numbers we have the following:

$$\begin{aligned} [\![(i, 0, m_1)]\!]_{\mathrm{M}} \quad \longrightarrow \quad & \widetilde{r_0}\{z_0.\,a_z\} \;\; m(x).\,x \\ & \widetilde{d}\{\overline{u_0} \;\; a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \;\; \overline{p_{i+1}})\rangle\} \\ & \widetilde{s}\{\overline{z_0} \;\; a_z.\,\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s})\rangle\} \;\; S = P_1 \end{aligned}$$

where $S = [\![r_1 = m_1]\!]_{\mathrm{M}} \; \prod_{i=1}^{n} [\![(i : I_i)]\!]_{\mathrm{M}}$ stands for the rest of the system. In P_1, the only reduction possible is through a synchronization on z_0, which signals the fact we are performing a jump. Such a synchronization, in turn, enables one on a_z. We then have:

$$\begin{aligned} P_1 \quad \longrightarrow^2 \quad & \widetilde{r_0}\{\mathbf{0}\} \;\; m(x).\,x \\ & \widetilde{d}\{\overline{u_0} \;\; a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \;\; \overline{p_{i+1}})\rangle\} \\ & \widetilde{s}\{\overline{m}\langle d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s})\rangle\} \;\; S = P_2\,. \end{aligned}$$

The only possible reduction from P_2 is by means of a synchronization on m. This gives us:

$$\begin{aligned} P_2 \quad \longrightarrow \quad & \widetilde{r_0}\{\mathbf{0}\} \;\; d(x).\,s(x).\,r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s}) \\ & \widetilde{d}\{\overline{u_0} \;\; a_1.\,\overline{m}\langle s(x).\,d(x).\,(\overline{a_2} \;\; \overline{p_{i+1}})\rangle\} \;\; \widetilde{s}\{\mathbf{0}\} \;\; S = P_3\,. \end{aligned}$$

In P_3 the two passivation units on d and s are consumed, which gives us:

$$P_3 \longrightarrow \widetilde{r_0}\{\mathbf{0}\} \;\; r_0(t).\,(\widetilde{r_0}\{z_0.\,\overline{a_z}\} \;\; \overline{p_s}) \;\; S = P_4\,.$$

At this point, it is easy to see that after a synchronization on r_0 we obtain:

$$P_4 \longrightarrow\equiv [\![(s, 0, m_1)]\!]_{\mathsf{M}}$$

as desired.

□

Lemma 5.15 (Soundness)**.** *Let* (i, m_0, m_1) *be a configuration of a Minsky machine* N*. If* $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow P_1$ *then for every computation of* P_1 *there exists a* P_j *such that* $P_j = [\![(i', m_0', m_1')]\!]_{\mathsf{M}}$ *and* $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m_0', m_1')$*.*

Proof. Consider the reduction $[\![(i, m_0, m_1)]\!]_{\mathsf{M}} \longrightarrow P_1$. An analysis of the structure of process $[\![(i, m_0, m_1)]\!]_{\mathsf{M}}$ reveals that, in all cases, the only possibility for the first step corresponds to the consumption of the program counter p_i. This implies that there exists an instruction labeled with i, that can be executed from the configuration (i, m_0, m_1). We proceed by a case analysis on the possible instruction, considering also the fact that the register on which the instruction acts can hold a value equal or greater than zero.

In all cases, it can be shown that computation evolves deterministically until reaching a process in which a new program counter (that is, some $\overline{p_{i'}}$) appears. The program counter $\overline{p_{i'}}$ is always inside a process that corresponds to $[\![(i', m_0', m_1')]\!]_{\mathsf{M}}$, where $(i, m_0, m_1) \longrightarrow_{\mathsf{M}} (i', m_0', m_1')$. The detailed analysis follows the same lines as the one reported for the proof of Lemma 5.14, and we omit it. □

Corollary 5.7. *Let* N *be a Minsky machine. We have that* $N \nrightarrow_{M}$ *if and only if* $[\![N]\!]_{\mathsf{M}} \nrightarrow$*.*

Proof. Straightforward from Lemmas 5.14 and 5.15. □

Lemma 5.16. *Termination and convergence are undecidable in* HoP$^{-\mathsf{f}}$*.*

Proof. This is an immediate consequence of previous results (Lemmas 5.14 and 5.15, Corollary 5.7). □

5.6 Concluding Remarks

In this chapter we have studied the expressiveness and decidability of higher-order process calculi featuring *limited forwarding*. Our study has been centered around Ho$^{-\mathsf{f}}$, the fragment of HOcore in which output actions can only include previously received processes in composition with closed ones. This communication style is reminiscent of programming scenarios with forms of code mobility in which the recipient is not authorized or capable of accessing/modifying the structure of the received code. We have shown that such a weakening of the forward capabilities of higher-order processes has consequences both on the expressiveness of the language and on the decidability of termination. Furthermore, we analyzed the extension of $\mathrm{Ho}^{-\mathrm{f}}$ with a *passivation* operator as a way of recovering the expressive power lost when moving from HOCORE to $\mathrm{Ho}^{-\mathrm{f}}$.

By exhibiting an encoding of Minsky machines into $\mathrm{Ho}^{-\mathrm{f}}$, we have shown that convergence is undecidable. Unlike the encoding of Minsky machines in HOCORE presented in Chapter 3, the encoding in $\mathrm{Ho}^{-\mathrm{f}}$ is not faithful. Hence, in the terminology of Bravetti and Zavattaro (2009), while HOCORE is Turing complete, $\mathrm{Ho}^{-\mathrm{f}}$ is only *weakly* Turing complete. This discrepancy on the criteria satisfied by each encoding reveals an expressiveness gap between $\mathrm{Ho}^{-\mathrm{f}}$ and HOCORE; nevertheless, it seems clear that the loss of expressiveness resulting from limiting the forwarding capabilities in HOCORE is much less dramatic than what one would have expected.

We have shown that the communication style of $\mathrm{Ho}^{-\mathrm{f}}$ causes a separation result with respect to HOCORE. In fact, because of the limitation on output actions, it was possible to prove that termination in $\mathrm{Ho}^{-\mathrm{f}}$ is decidable. This is in sharp contrast with the situation in HOCORE, for which termination is undecidable. In $\mathrm{Ho}^{-\mathrm{f}}$, it is possible to provide an upper bound on the depth (i.e., the level of nesting of actions) of the (set of) derivatives of a process. In HOCORE such an upper bound does not exist. This was shown to be essential for obtaining the decidability result; for this, we appealed to the approach developed in (Busi et al., 2009), which relies on the theory of well-structured transition systems (Finkel, 1990; Abdulla et al., 2000; Finkel and Schnoebelen, 2001). As far as we are aware, this approach to studying expressiveness issues has not previously been used in the higher-order setting. The decidability of termination is significant, as it might shed light on the development of verification techniques for higher-order processes.

We have also studied the expressiveness and decidability of $\mathrm{HoP}^{-\mathrm{f}}$, the extension of $\mathrm{Ho}^{-\mathrm{f}}$ with a passivation operator. To the best of our knowledge, this is the first expressiveness study involving passivation operators in the context of higher-order process calculi. In $\mathrm{HoP}^{-\mathrm{f}}$ it is possible to encode Minsky machines in a *faithful* manner. Hence, similarly as in HOCORE, in $\mathrm{HoP}^{-\mathrm{f}}$ both termination and convergence are undecidable. This certainly does not imply that both languages have the same expressive power; in fact, an interesting direction for future work consists in assessing the exact expressive power that passivation brings into the picture. This would include not only a comparison between $\mathrm{HoP}^{-\mathrm{f}}$ and HOCORE, but also a comparison between HOCORE and HOCORE extended with passivation. All the languages involved are Turing complete, hence such comparisons should employ techniques different from the ones used here. It is also worth remarking that we have considered a very simple form of passivation, one in which process suspension takes place with a considerable degree of non-determinism.

Studying other forms of passivation, possibly with more explicit control mechanisms, could be interesting from several points of view, including expressiveness.

The $\mathrm{Ho}^{-\mathrm{f}}$ calculus is a sublanguage of HOcore. As such, $\mathrm{Ho}^{-\mathrm{f}}$ inherits the many results and properties of HOcore; most notably, a notion of (strong) bisimilarity which is decidable and coincides with a number of sensible equivalences in the higher-order context. Our results thus complement those in previous chapters and deepen our understanding of the expressiveness of core higher-order calculi as a whole. Furthermore, by recalling that CCS without restriction is not Turing complete and has decidable convergence, the present results shape an interesting expressiveness hierarchy, namely one in which HOcore is strictly more expressive than $\mathrm{Ho}^{-\mathrm{f}}$ (because of the discussion above), and in which $\mathrm{Ho}^{-\mathrm{f}}$ is strictly more expressive than CCS without restriction.

Remarkably, our undecidability result can be used to prove that (weak) barbed bisimilarity is undecidable in the calculus obtained by extending $\mathrm{Ho}^{-\mathrm{f}}$ with restriction. Consider the encoding of Minsky machines used in Section 5.3 to prove the undecidability of convergence in $\mathrm{Ho}^{-\mathrm{f}}$. Consider now the restriction operator $(\nu\widetilde{x})$ used as a binder for the names in the tuple $\widetilde{x}$. Take a Minsky machine N (it is not restrictive to assume that it executes at least one increment instruction) and its encoding P, as given by Definition 5.3. Let $\widetilde{x}$ be the tuple of the names used by P, excluding the name w. We have that N terminates if and only if $(\nu\widetilde{x})P$ is (weakly) barbed equivalent to the process $(\nu d)(\overline{d} \mid d \mid d.(\overline{w} \mid !w.\overline{w}))$.

Related Work. We do not know of other works that study the expressiveness of higher-order calculi by restricting higher-order outputs. The recent work (Bundgaard et al., 2009) studies finite-control fragments of Homer (Hildebrandt et al., 2004), a higher-order process calculus with locations. While we have focused on decidability of termination and convergence, in (Bundgaard et al., 2009) the interest is in decidability of barbed bisimilarity. One of the approaches explored in (Bundgaard et al., 2009) is based on a type system that bounds the size of processes in terms of their syntactic components (e.g. number of parallel components, location nesting). Although the restrictions such a type system imposes might be considered as similar in spirit to the limitation on outputs in $\mathrm{Ho}^{-\mathrm{f}}$ (in particular, location nesting resembles the output nesting $\mathrm{Ho}^{-\mathrm{f}}$ forbids), the fact that the synchronization discipline in Homer depends heavily on the structure of locations makes it difficult to establish a more detailed comparison with $\mathrm{Ho}^{-\mathrm{f}}$.

Also similar in spirit to our work, but in a slightly different context, are some studies on the expressiveness (of fragments) of the Ambient calculus (Cardelli and Gordon, 2000). Ambient and higher-order calculi are related in that both allow the communication of objects with complex structure. Some works on the expressiveness of fragments of Ambient calculi are similar to ours. In particular, (Busi and Zavattaro, 2004) shows that termination is decidable for the fragment without both restriction (as $\mathrm{Ho}^{-\mathrm{f}}$ and HOcore) and movement capabilities, and featuring replication; in contrast, the same property turns out to be undecidable for the fragment with recursion. Hence, the separation between fragments comes from the source of infinite behavior, and not from the structures allowed in output action, as in our case. However, we find that the connections between Ambient-like and higher-order calculi are rather loose, so a proper comparison is difficult also in this case.

Chapter 6

On the Expressiveness of Synchronous and Polyadic Communication

In this chapter we study the expressiveness of synchronous and polyadic communication in higher-order process calculi. We thus consider extensions of HOcore with restriction and polyadic communication. We present both encodabilty and impossibility results: first we show that asynchronous process-passing is expressive enough so as to encode synchronous communication. Then, we show that a similar result for polyadic communication does not hold. In fact, we show that a hierarchy of synchronous higher-order process calculi based on the arity of polyadic communications is induced. Finally, we examine the influence abstraction passing has in the expressiveness of the considered calculi. Central to our results is the fact that the establishment of *private links* —as available in first-order concurrency— is not possible in the absence of name-passing.

Section 6.2 introduces the families of higher-order process calculi we shall be working with. Section 6.4 presents and discusses the encodability result of synchronous into asynchronous communication. Section 6.5 presents separation results for encodings involving polyadic communication, whereas Section 6.6 discusses the power of abstraction passing. Section 6.7 concludes.

The separation results for the expressiveness of polyadic communication have been published as an extended abstract in (Lanese et al., 2009); all the other results and discussions are original to this dissertation.

6.1 Introduction

In this chapter we continue our study of the fundamental properties of higher-order process calculi. We concentrate on *asynchrony* (and its relationship with *synchrony*) and *polyadic communication*. These are two well-understood mechanisms in first-order calculi. Asynchronous communication is of practical relevance since, e.g., it is easier to establish and maintain than synchronous communication. It is also of theoretical interest: numerous works have studied the *asynchronous π-calculus* and the rather surprising effects that the absence of output prefix has over the behavioral theory and expressiveness of the calculus. In a well-known result, Palamidessi showed that the asynchronous π-calculus with separate choice is strictly less expressive than the synchronous π-calculus (Palamidessi, 2003). As for polyadic communication —that is, the passing of tuples of values in communications— it is among the most natural and convenient features for modeling purposes; indeed, it is a stepping stone for the representation of data structures —such as lists and records— as processes.

In the π-calculus without choice, both synchronous and polyadic communication are supported by *encodings* into more basic settings, namely synchronous into asynchronous communication (Boudol, 1992; Honda and Tokoro, 1991), and polyadic into monadic communication (Milner, 1991), respectively. A salient commonality in both encodings is the fundamental rôle played by the *communication of restricted names*. More precisely, both encodings exploit the ability that first-order processes have of *establishing private links* between two or more processes by generating and communicating restricted names. Let us elaborate further on this point by recalling the encoding of the polyadic π-calculus into the monadic one in (Milner, 1991):

$$\begin{aligned}
[\![x(z_1, \dots, z_n).\, P]\!] &= x(w).\, w(z_1). \cdots .\, w(z_n).\, [\![P]\!] \\
[\![\overline{x}\langle a_1, \dots, a_n\rangle.\, P]\!] &= \nu w\, \overline{x}w.\, \overline{w}a_1. \cdots .\, \overline{w}a_n.\, [\![P]\!]
\end{aligned}$$

(where $[\![\cdot]\!]$ is an homomorphism for the other operators). A single n-adic synchronization is encoded as $n+1$ monadic synchronizations. The first synchronization establishes a *private link* w: the encoding of output creates a private name w and sends it to the encoding of input. As a result of the synchronization on x, the scope of w is extruded, and each of $a_1, \dots, a_n$ can then be communicated through monadic synchronizations on w. This encoding is very intuitive, and satisfies a tight operational correspondence property: a term of the polyadic calculus with *one single public* synchronization (i.e., a synchronization on an unrestricted name such as x) is encoded into a term of the monadic calculus with *exactly one public* synchronization on the same name, followed by a number of *internal* synchronizations (i.e., synchronizations on a private name such as w). That is, not only the observable behavior is preserved, but a source term and its encoding in the target language perform the exact same number of visible actions. The crucial advantage of establishing a private link on w is that the encoding is *robust with respect to interferences*: once the private link has been established between two parties, no surrounding —possibly malicious— context can get access to the monadic communications on w.

The establishment of private links is then seen to arise naturally from the interplay of restriction *and* name-passing as available in the π-calculus. In this chapter we aim at understanding whether the settled situation in the first-order setting carries over to the higher-order one. More precisely, we study the extent to which private links can be established in the context of HOcore, a higher-order process calculus *without name passing*. This appears as a particularly intriguing problem: in spite of its minimality, HOcore is very expressive: not only it is Turing complete but also several modelling idioms (disjoint choice, input-guarded replication, lists) are expressible in it as derived constructs. Hence, the answer to this question is far from obvious.

Here we shall consider two extensions of HOcore. The first one—denoted AHO—extends HOcore with restriction and polyadic communication; the second extension—denoted SHO—extends AHO with output prefixes, so as to represent *synchronous* process passing. Since both calculi consider polyadic communication, AHO and SHO actually represent two *families* of higher-order process calculi: given $n \geq 0$, we use SHO^n (resp. AHO^n) to denote the synchronous (resp. asynchronous) higher-order process calculus with n-adic communication.

It is useful to comment on the consequences of considering restricted names in higher-order process calculi *without name-passing*. The most notable one is the *partial effect* that scope extrusions have. Let us explain what we mean by this. In a process-passing setting, received processes can only be executed, forwarded, or discarded. Hence, an input context cannot gain access to the (private) names of those processes it receives; to the context received processes are much like a "black box". Although higher-order communications might lead to scope extrusion of the private names *contained* in the transmitted processes, such extrusions are vacuous: without name-passing, a receiving context can only use the names contained in a process in a restricted way, namely the way decreed by the sender of the process.[1] The sharing of (private) names one obtains from using process-passing only is then incomplete: names can be *sent* as part of processes but they cannot be freely used by a recipient.

With the above discussion in mind, we begin by investigating the relationship between synchrony and asynchrony in process-passing calculi. Our first main result is an encoding of SHO into AHO. Intuitively, a synchronous output is encoded by an asynchronous output that communicates both the communication object and the continuation. This *encodability* result is significant: it reveals that the absence of name passing does *not* necessarily imply that encodings that rely on name-passing and private links are not expressible with process-passing only. In fact, the encoding bears witness to the expressive power intrinsic to (asynchronous) process-passing.

Based on this positive result, we move to examine the situation for polyadic communication in process-passing calculi. We consider variants of SHO with different arity in communications, and study their relative expressive power. Interestingly, we determine that it is indeed the case

[1] In this discussion we understand process-passing that does *not* consider abstraction-passing, i.e. the communication of functions from processes to processes. As we shall see, the situation is rather different with abstraction-passing.

that the absence of name-passing causes an expressiveness loss when considering polyadic communication. Our second main contribution is a non-encodability result: for every $n > 1$, SHO^n cannot be encoded into SHO^{n-1}. This way we obtain a *hierarchy* of higher-order process calculi of strictly increasing expressiveness. Hence, polyadic communication is a striking point of contrast between first-order and higher-order process calculi *without name-passing*.

The crux for obtaining the above hierarchy is a characterization of the *stability conditions* of higher-order processes with respect to their sets of private names. Intuitively, such conditions are meant to capture the following insight: without name-passing, the set of names that are private to a given process remains invariant along computations. As such, two processes that interact respecting the stability conditions and do not share a private name will never be able to establish a private link on it. Focusing on the set of names private to a process is crucial to characterize the private links it can establish. Central to the definability of the stability conditions is a refined account of internal actions that is enforced by the LTS associated to SHO. In fact, the LTS distinguishes the internal actions that result from synchronizations on restricted names from those that result from synchronizations on public names. While the former are the only kind of internal actions, the latter are considered as visible actions.

The separation result for polyadic communication depends on a notion of encoding that is defined in accordance to the stability conditions and requires one visible action in the source language to be matched by at most one visible action in the target language. When compared to proposals for "good" encodings in the literature, this requirement might appear as rather demanding. However, we claim a demanding notion of encoding is indispensable in our case for at least two reasons. First, such a notion allows us to concentrate in compositional encodings that are robust with respect to interferences. As we have discussed, these two properties follow naturally from the ability of establishing of private links in the first-order setting. Arbitrary, potentially malicious interferences are thus a central issue. The requirement on visible actions is intended to ensure that a term and its encoding are exposed to the same *interference points*. We argue that such a requirement is a reasonable way of including arbitrary sources of interferences into the notion of encoding. Second, in the higher-order setting the encoding of synchronous communication into asynchronous one can be seen as a *particular case* of the encoding of polyadic communication into monadic one. This way, for instance, *monadic synchronous* communication corresponds to the class of *biadic asynchronous* communication in which the second parameter (i.e., the continuation of output) is executed only once. This observation and the encodability result for synchronous communication into asynchronous one suggest that the gap between what can be encoded with process-passing and what cannot is rather narrow. Therefore, a notion of encoding more discriminating than usual is necessary in our case to be able to formalize separation results among calculi with different polyadicity.

In the final part of the chapter we consider the extension of SHO with *abstractions*. An abstraction is an expression of the form $(x)P$—it is a parameterized process. An abstraction has a functional type. Applying an abstraction $(x)P$ of type $T \to \diamond$ (where $\diamond$ is the type of all processes) to an argument W of type T yields the process $P\{^W\!/x\}$. The argument W can itself be an abstraction; therefore the *order* of an abstraction, that is, the level of arrow nesting in its type, can be arbitrarily high. The order can also be ω, if there are recursive types. We consider SHO^n_a, the extension of SHO^n with abstractions of order one (i.e., functions from processes to processes). Our last main result shows that abstraction passing provides SHO with the ability of establishing of private links. Indeed, we show that SHO^n can be encoded into SHO^1_a. This can be used to demonstrate that there is no encoding of SHO^n_a into SHO^n. This result thus provides further evidence on the relationship between the ability of establishing private links and absolute expressiveness.

Related Work. While a number of works address the relationship between synchronous and asynchronous communication in first-order calculi (see, e.g., (Palamidessi, 2003; Cacciagrano et al., 2007; Beauxis et al., 2008)), we are not aware of analogous studies for higher-order process calculi. A similar situation occurs for the study of polyadic communication; in the first-order setting the interest has been in characterizing fully-abstract translations of polyadic communication into monadic one (see, e.g., (Quaglia and Walker, 2005; Yoshida, 1996)), but the case of polyadicity in higher-order communication has not been addressed.

The most related work is by Sangiorgi (1996b). There, the expressiveness of the π-calculus with respect to higher-order π is studied by identifying hierarchies of fragments of first-order and higher-order calculi with increasingly expressive power. The first-order hierarchy is based on fragment of the π-calculus in which mobility is *internal*, i.e., where outputs are only on private names —no free outputs are allowed. This hierarchy is denoted as $\pi\mathrm{I}^n$, where the n denotes the degree of mobility allowed; this is formalized by means of *dependency chains* in name creation. In this hierarchy, e.g., $\pi\mathrm{I}^1$ does not allow mobility and corresponds to the core of CCS, and $\pi\mathrm{I}^n$ will allow dependency chains of length at most n. The hierarchy in the higher-order case follows a similar rationale, and is based on the *strictly higher-order* π-calculus, i.e., a higher-order calculus without name-passing features. Also in this hierarchy, the less expressive language (denoted $\mathrm{HO}\pi^1$) corresponds to the core of CCS. Sangiorgi shows that $\pi\mathrm{I}^n$ and $\mathrm{HO}\pi^n$ have the same expressiveness, by exhibiting fully-abstract encodings. In contrast to (Sangiorgi, 1996b), the hierarchy of higher-order process calculi we consider here is not given by the degree of mobility allowed, but by the size of the tuples that can be passed around in polyadic communications.

The distinction between internal and public synchronizations here proposed for our notion of encoding has been used and/or proposed in other contexts. In (Lanese, 2007) labels of internal actions are annotated with the name on which synchronization occurs so as to define *located* semantics which are then used to study concurrent semantics for the π-calculus using standard labeled transition systems. In the higher-order setting (Amadio, 1993) obtains a finitely-branching bisimilarity for CHOCS by means of a reduction into bisimulation for a variant of the π-calculus. In such a variant, processes are only allowed to exchange names of *activation channels* (i.e. the channels that trigger a copy of a process in the representation of higher-order communication with first-order one). The desired finitely-branching bisimilarity is obtained by relying on a labeled transition system in which synchronizations on activation channels are distinguished.

6.2 The Calculi

6.2.1 A Higher-Order Process Calculus with Restriction and Polyadic Communication

Here we define AHO, the extension of HOcore with a restriction operator and polyadic communication. As such, it is asynchronous and does not feature name-passing.

Definition 6.1. *The language of* AHO *processes is given by the following syntax:*

$$P, Q, \ldots \quad ::= \quad a(\widetilde{x}).\, P \;\Big|\; \bar{a}\langle \widetilde{Q} \rangle \;\Big|\; P_1 \;\; P_2 \;\Big|\; \nu r\, P \;\Big|\; x \;\Big|\; \mathbf{0}$$

where x, y range over process variables, and a, b, r, s denote names.

Assuming standard notation and properties for tuples of syntactic elements, polyadicity in process passing is interpreted as expected: an output message $\overline{a}\langle \widetilde{Q} \rangle$ sends the tuple of processes $\widetilde{Q}$ on name a; an input prefixed process $a(\widetilde{x}).\, P$ can receive a tuple $\widetilde{Q}$ on name (or channel) a and continue as $P\{\widetilde{Q}/\widetilde{x}\}$. In both cases, a is said to be the *subject* of the action. We sometimes write $\mid \widetilde{x} \mid$ for the length of tuple $\widetilde{x}$; the length of the tuples that are passed around determines the actual *arity* in polyadic communication. In interactions, we assume inputs and outputs agree on their arity; we shall rely on notions of *types* and *well-typed processes* as in (Sangiorgi, 1996b). Parallel composition allows processes to interact, and $\nu r\, P$ makes r private (or restricted) to the process P. Notions of bound and free names and variables (bn(·), fn(·), bv(·), and fv(·), resp.) are defined in the usual way: an input $a(\widetilde{x}).\, P$ binds the free occurrences of variables in $\widetilde{x}$ in P; similarly, $\nu r\, P$ binds the free occurrences of name r in P. We abbreviate $a(\widetilde{x}).\, P$ as $a.\, P$ when none of the variables in $\widetilde{x}$ is in fv(P), and $\overline{a}\langle \widetilde{\mathbf{0}} \rangle$ as $\overline{a}$. We use notation $\prod^k P$ to represent k copies of process P in parallel.

Definition 6.2. *The* structural congruence *relation for* AHO *processes is the smallest congruence generated by the following laws:* $P \;\; \mathbf{0} \equiv P$*,* $\;P_1 \;\; P_2 \equiv P_2 \;\; P_1$*,* $\;P_1 \;\; (P_2 \;\; P_3) \equiv (P_1 \;\; P_2) \;\; P_3$*,* $\;\nu a\, \nu b P \equiv \nu b\, \nu a P$*,* $\;\nu a\, \mathbf{0} \equiv \mathbf{0}$*,* $\;\nu a\, (P_1 \;\; P_2) \equiv \nu a\, P_1 \;\; P_2$ *—if* $a \notin$ fn(P_2).

$$\textsc{Inp}\quad a(\widetilde{x}).\,P \xrightarrow{a(\widetilde{x})} P \qquad\qquad \textsc{Out}\quad \overline{a}\langle\widetilde{Q}\rangle \xrightarrow{\overline{a}\langle\widetilde{Q}\rangle} \mathbf{0}$$

$$\textsc{Act1}\quad \dfrac{P_1 \xrightarrow{\alpha} P_1' \qquad \mathrm{bv}(\alpha) \cap \mathrm{fv}(P_2) = \emptyset}{P_1 \;\; P_2 \xrightarrow{\alpha} P_1' \;\; P_2}$$

$$\textsc{Tau1}\quad \dfrac{P_1 \xrightarrow{(\nu\widetilde{y})\overline{a}\langle\langle\widetilde{P}\rangle\rangle} P_1' \qquad P_2 \xrightarrow{a(\widetilde{x})} P_2' \qquad \widetilde{y} \cap \mathrm{fn}(P_2) = \emptyset}{P_1 \;\; P_2 \xrightarrow{\tau} \nu\widetilde{y}\,(P_1' \;\; P_2'\{\widetilde{P}/\widetilde{x}\})}$$

$$\textsc{Res}\quad \dfrac{P \xrightarrow{\alpha} P' \qquad r \notin \mathrm{n}(\alpha)}{\nu r\, P \xrightarrow{\alpha} \nu r\, P'}$$

$$\textsc{Open}\quad \dfrac{P \xrightarrow{(\nu\widetilde{y})\overline{a}\langle\langle\widetilde{P''}\rangle\rangle} P' \qquad x \neq a,\, x \in \mathrm{fn}(\widetilde{P''}) - \widetilde{y}}{\nu x\, P \xrightarrow{(\nu x\widetilde{y})\overline{a}\langle\langle\widetilde{P''}\rangle\rangle} P'}$$

Figure 6.1: The LTS of AHO. We have omitted rules Act2 and Tau2, the symmetric counterparts of rules Act1 and Tau1.

The semantics for AHO is given in terms of the LTS given in Figure 6.1. There are three kinds of transitions: internal transitions $P \xrightarrow{\tau} P'$, input transitions $P \xrightarrow{a(\widetilde{x})} P'$, and output transitions $P \xrightarrow{(\nu\widetilde{y})\overline{a}\langle\langle\widetilde{Q}\rangle\rangle} P'$ (with extrusion of the tuple of names $\widetilde{y}$), which have the expected meaning. We use α to range over actions. The subject of action α, denoted as $sub(\alpha)$, is defined as $sub(a(\widetilde{x})) = a$, $sub(\overline{a}\langle\langle\widetilde{Q}\rangle\rangle) = a$, and is undefined otherwise. Notions of bound and free names and variables extend to actions as expected. We sometimes use $\vec{\alpha}$ to denote a sequence of actions $\alpha_1, \ldots, \alpha_n$. Weak transitions are defined in the usual way. We write $\Rightarrow$ for the reflexive, transitive closure of $\xrightarrow{\tau}$. Given an action α, notation $\overset{\alpha}{\Longrightarrow}$ stands for $\Rightarrow\xrightarrow{\alpha}\Rightarrow$. Given a sequence $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, we define $\overset{\vec{\alpha}}{\Longrightarrow}$ as $\overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow}$.

Convention 6.1. *In what follows we shall say that, for some $n > 0$, AHO^n corresponds to the higher-order process calculus obtained from the syntax given in Definition 6.1 in which polyadic communication has arity n.*

The following definition is standard.

Definition 6.3 (Strong and Weak Barbs). *Given a process P and a name a, we write*

- $P \downarrow_a$ *—a* strong input barb*— if P can perform an input action with subject a;*
- $P \downarrow_{\overline{a}}$ *—a* strong output barb*— if P can perform an output action with subject a.*

Given $\mu \in \{a, \overline{a}\}$, we define a weak *barb $P \Downarrow_\mu$ if, for some P', $P \Rightarrow P' \downarrow_\mu$.*

6.2.2 A Higher-Order Process Calculus with Synchronous Communication

We now introduce SHO, the extension of AHO with synchronous communication. As such, processes of SHO are defined in the same way as the processes of AHO (Definition 6.1), except that output is a prefix:

Definition 6.4. *The language of* SHO *processes is given by the syntax in Definition 6.1, excepting that output message $\overline{a}\langle\widetilde{Q}\rangle$ is replaced with $\overline{a}\langle\widetilde{Q}\rangle.\, P$.*

The intended meaning of the output prefix is as expected: $\overline{a}\langle\widetilde{Q}\rangle.\, P$ can send the tuple of processes $\widetilde{Q}$ via name a and then continue as P. All notions on bound variables and names are defined as in AHO.

The LTS for SHO is obtained from that for AHO in Figure 6.1 with two modifications. The first one concerns the shape of output actions: rule Out is replaced with

$$\textsc{SOut}\quad \overline{a}\langle\widetilde{Q}\rangle.\, P \xrightarrow{\overline{a}\langle\widetilde{Q}\rangle} P$$

which formalizes synchronous output. The second modification enforces the distinction between *internal* and *public* synchronizations hinted at in the introduction. This distinction is obtained in two steps. First, by replacing rule Tau1 with the following one:

$$\textsc{PubTau1}\quad \frac{P_1 \xrightarrow{(\nu\widetilde{y})\overline{a}\langle\widetilde{P}\rangle} P_1' \qquad P_2 \xrightarrow{a(\widetilde{x})} P_2' \qquad \widetilde{y} \cap \mathrm{fn}(P_2) = \emptyset}{P_1 \;\; P_2 \xrightarrow{a\tau} \nu\widetilde{y}\,(P_1' \;\; P_2'\{\widetilde{P}/\widetilde{x}\})}$$

(And similarly for Tau2, which is replaced by PubTau2, the analogous of PubTau1.) The second step consists in extending the LTS with the following rule:

$$\textsc{IntRes}\quad \frac{P \xrightarrow{a\tau} P'}{\nu a\, P \xrightarrow{\tau} \nu a\, P}$$

This way we are able to distinguish between *internal* and *public* synchronizations. The former are given by synchronizations on *restricted* names; they are the only source of internal behavior and are denoted as $\xrightarrow{\tau}$. The latter are given by synchronization on *public* names: a synchronization on the public name a leads to the visible action $\xrightarrow{a\tau}$. The distinction between internal and public synchronizations does not have behavioral consequences; it only represents a more refined standpoint of internal behavior that we shall find useful for obtaining results in Section 6.5. As a result, we have four kinds of transitions: in addition to internal and public synchronizations, we have input and output transitions as defined for AHO. Accordingly, we extend the definition of subject of an action for the case of public synchronizations, and decree that $sub(a\tau) = a$.

By varying the arity in polyadic communication, Definition 6.4 actually gives a *family* of higher-order process calculi. We have the following notational convention:

Convention 6.2. *In what follows we shall say that, for some $n > 0$,* SHO^n *corresponds to the higher-order process calculus obtained from the syntax given in Definition 6.4 in which polyadic communication has arity n.*

6.3 The Notion of Encoding

The following definition of encoding is inspired on that of Gorla (2008), who proposed five criteria a "good encoding" should satisfy.

Definition 6.5. *A* language $\mathcal{L}$ *is defined as:*

- *a set of* processes $\mathcal{P}$*;*
- *a labeled transition relation $\longrightarrow$ on $\mathcal{P}$, i.e. a structure $(\mathcal{P}, \mathcal{A}, \longrightarrow)$ for some set $\mathcal{A}$ of* actions *or* labels*.*
- *a weak behavioral equivalence $\approx$ (i.e. a behavioral equivalence that abstracts from internal actions in $\mathcal{A}$).*

A *translation* considers two languages, a *source* and a *target*:

Definition 6.6 (Translation)**.** *Given a source language $\mathcal{L}_s = (\mathcal{P}_s, \longrightarrow_s, \approx_s)$ and a target language $\mathcal{L}_t = (\mathcal{P}_t, \longrightarrow_t, \approx_t)$, a* translation *of $\mathcal{L}_s$ into $\mathcal{L}_t$ is a function $[\cdot] : \mathcal{P}_s \to \mathcal{P}_t$.*

We shall be interested in a class of translations that respect both syntactic and semantic conditions.

Definition 6.7 (Syntactic Conditions on Translations)**.** *Let $[\cdot] : \mathcal{P}_s \to \mathcal{P}_t$ be a translation of $\mathcal{L}_s$ into $\mathcal{L}_t$. We say that $[\cdot]$ is*

1. Compositional*: if for every k-ary operator* op *of $\mathcal{L}_s$ and for all $S_1, \ldots, S_k$ with* $\mathsf{fn}(S_1, \ldots, S_k) = N$*, then there exists a k-ary context $C^N_{\mathsf{op}} \in \mathcal{P}_t$ such that*
$$[\mathsf{op}(S_1, \ldots, S_k)] = C^N_{\mathsf{op}}[[S_1], \ldots, [S_k]].$$
2. Name invariant*: if $[\sigma(P)] = \sigma([P])$, for any injective permutation of names σ.*

Definition 6.8 (Semantic Conditions on Translations)**.** *Let $[\cdot] : \mathcal{P}_s \to \mathcal{P}_t$ be a translation of $\mathcal{L}_s$ into $\mathcal{L}_t$. We say that $[\cdot]$ is* operational corresponding *if the following properties hold:*

1. Completeness/Preservation*: For every $S, S' \in \mathcal{P}_s$ and $\alpha \in \mathcal{A}_s$ such that $S \overset{\alpha}{\Longrightarrow}_s S'$, it holds that $[S] \overset{\beta}{\Longrightarrow}_t \approx_t [S']$, where $\beta \in \mathcal{A}_t$ and $sub(\alpha) = sub(\beta)$.*
2. Soundness/Reflection*: For every $S \in \mathcal{P}_s$, $T \in \mathcal{P}_t$, $\beta \in \mathcal{A}_t$ such that $[S] \overset{\beta}{\Longrightarrow}_t T$ there exists an $S' \in \mathcal{P}_s$ and an action $\alpha \in \mathcal{A}_s$ such that $S \overset{\alpha}{\Longrightarrow}_s S'$, $T \Rightarrow \approx_t [S']$, and $sub(\alpha) = sub(\beta)$.*

Furthermore, we shall require adequacy*: if $P \approx_{\mathrm{s}} Q$ then $[\![P]\!] \approx_{\mathrm{t}} [\![Q]\!]$.*

Notice that adequacy is necessary because we make no assumptions on the nature of $\approx_{\mathrm{s}}$ and $\approx_{\mathrm{t}}$.

Definition 6.9. *We call* encoding *any translation that satisfies both the syntactic conditions in Definition 6.7 and the semantic conditions in Definition 6.8.*

Remark 6.1. *Notice that our definition of encoding intends to capture the fact that an action in the source language might be not matched by the exact same action in the target language.*

Some Properties of Encodings.

Proposition 6.1. *Let a be a name. If $a \in \mathrm{fn}(P)$ then also $a \in \mathrm{fn}([\![P]\!])$.*

Proof. By contradiction. Take two distinct names a and b. Suppose a is free in P. Clearly, we have that

$$P\{{}^{b}\!/\!_{a}\} \neq P \quad (*)$$

Suppose, for the sake of contradiction, that a is not free in $[\![P]\!]$. Under that assumption, one has that $[\![P]\!]\{{}^{b}\!/\!_{a}\} = [\![P]\!]$ as substituting a non-free name with another name is a vacuous operation. Notice that by name invariance one has $[\![P]\!]\{{}^{b}\!/\!_{a}\} = [\![P\{{}^{b}\!/\!_{a}\}]\!]$. Now, considering $(*)$ above, one has the $[\![P]\!]\{{}^{b}\!/\!_{a}\} \neq [\![P\{{}^{b}\!/\!_{a}\}]\!]$, a contradiction. □

Proposition 6.2. *Let $[\![\cdot]\!]$ be an encoding of $\mathcal{L}_{\mathrm{s}}$ into $\mathcal{L}_{\mathrm{t}}$. Then $[\![\cdot]\!]$ satisfies:*

1. Barb preservation*: for every $S \in \mathcal{P}_{\mathrm{s}}$ it holds that $S \Downarrow_{\overline{a}}$ (resp. $S \Downarrow_{a}$) if and only if $[\![S]\!] \Downarrow_{\overline{a}}$ (resp. $[\![S]\!] \Downarrow_{a}$).*

Proof. It follows from operational correspondence in the definition of encoding (Definition 6.9) □

Proposition 6.3 (Composability of Encodings)**.** *If $\mathcal{C}[\![\cdot]\!]$ is an encoding of $\mathcal{L}_1$ into $\mathcal{L}_2$, and $\mathcal{D}[\![\cdot]\!]$ is an encoding of $\mathcal{L}_2$ into $\mathcal{L}_3$ then their composition $(\mathcal{D} \cdot \mathcal{C})[\![\cdot]\!]$ is an encoding of $\mathcal{L}_1$ into $\mathcal{L}_3$.*

Proof. From the definition of encoding (Definition 6.9). The syntactic conditions (compositionality, name invariance) are easily seen to hold for $(\mathcal{D} \cdot \mathcal{C})[\![\cdot]\!]$; the semantic conditions (operational correspondence, adequacy) rely on the fact that $\approx_1, \approx_2$, and $\approx_3$ are equivalences and hence transitive. Note that adequacy is crucial to show the composability for operational correspondence. □

6.4 An Encodability Result for Synchronous Communication

We begin by studying the relationship between synchronous and asynchronous communication. The main result of this section is an encoding of SHO^n into AHO^n.

A naive encoding would simply consist in sending both the communication object and the continuation of the output action in a single synchronization. The continuation is sent explicitly as a parameter, and so a synchronous calculus with polyadicity n would have to be encoded into an asynchronous calculus with polyadicity $n+1$. To illustrate this, consider the naive encoding of SHO^1 into AHO^2:

$$
\begin{aligned}
[\overline{a}\langle P\rangle.\, S] &= \overline{a}\langle [P], [S]\rangle \\
[a(x).\, R] &= a(x,y).\, (y \;\; [R])
\end{aligned}
$$

where $[\cdot]$ is an homomorphism for the other operators in SHO^1. This encoding allows to appreciate how in the higher-order setting the synchronous/asynchronous distinction can be considered as a particular case of the polyadic/monadic distinction. Notice that the fact that the continuation is supposed to be executed only once is crucial for the simplicity of the encoding.

Interestingly, we notice that asynchronous process-passing is expressive enough so as to encode synchronous communication of the *same arity*. Intuitively, the idea is to send a *single process* consisting of a guarded choice between a communication object and the continuation of the synchronous output. For the monadic case the encoding is as follows:

$$
\begin{aligned}
[\overline{a}\langle P\rangle.\, S] &= \nu k, l\, (\overline{a}\langle k.\, (\llbracket P\rrbracket \;\; \overline{k}) + l.\, (\llbracket S\rrbracket \;\; \overline{k})\rangle \;\; \overline{l}) \\
[a(x).\, R] &= a(x).\, (x \;\; \llbracket R\rrbracket)
\end{aligned}
$$

where "+" stands for the encoding of disjoint choice in HOcore, presented in Section 3.2; k, l are two names not in $\mathrm{fn}(P, S)$; and $[\cdot]$ is an homomorphism for the other operators in SHO^1.

The synchronous output action is thus encoded by sending a guarded, disjoint choice between the encoding of the communication object and the encoding of the continuation of the output. The encoding exploits the fact that the continuation should be executed exactly once, while the communication object can be executed zero or more times. Notice that there is only one copy of the trigger that executes the encoding of the continuation (denoted $\overline{l}$ in the encoding above), which guarantees that it is executed exactly once. This can only occur after that the synchronization has taken place, thus ensuring a correct encoding of synchronous communication. Notice that $\overline{l}$ releases both the encoding of the continuation and a trigger for executing the encoding of the communication object (denoted $\overline{k}$); such an execution will only occur when the choice sent by the encoding of output appears at the top level. This way, it is easy to see that a trigger $\overline{k}$ is always available. This idea can be generalized to encode synchronous calculi of arbitrary polyadicity as follows:

Definition 6.10 (Synchronous into Asynchronous). *For some $n > 0$, the encoding of* SHO^n *into* AHO^n *is defined as follows:*

$$\begin{aligned}
[\overline{a}\langle P_1, \ldots, P_n \rangle. S] &= \nu k, l\, (\overline{a}\langle [P_1], \ldots, [P_{n-1}], T_{k,l}[\, [P_n], [S]\,] \rangle \ \ \overline{l}) \\
[a(x_1, \ldots, x_n). R] &= a(x_1, \ldots, x_n). (x_n \ \ [R])
\end{aligned}$$

with

$$T_{k,l}[M_1, M_2] = k. (M_1 \ \ \overline{k}) + l. (M_2 \ \ \overline{k})$$

where $\{k, l\} \cap \mathrm{fn}(P_1, \ldots, P_n, S) = \emptyset$, and $[\cdot]$ is an homomorphism for the other operators in SHO^n*.*

Proposition 6.4 (Correctness of the Encoding). *The translation in Definition 6.10 is an encoding in the sense of Definition 6.9.*

Proof (Sketch). It is immediate to observe that the encoding respects syntactic conditions as in Definition 6.7. To show operational correspondence as in Definition 6.8, one defines a characterization of the "garbage" that the process leaves along reductions. Such garbage is essentially determined by occurrences of the trigger that activates a copy to (the encoding of) the last parameter of the polyadic communication (denoted $\overline{k}$ in Definition 6.10). Such occurrences remain while the summation that the encoding sends is not at the top-level; some triggers might remain even if all summations have been consumed. Crucially, since such triggers are on restricted names, they are harmless for the rest of the process, and so the encoding is correct up to these extra triggers. □

The encoding is significant as it provides compelling evidence on the expressive power that (asynchronous) process-passing has for representing protocols that rely on establishment of private links in the first-order setting. Not only the encoding bears witness of the fact that such protocols can indeed be encoded into calculi with process-passing only; the observation that the encoding of synchronous into asynchronous communication is a particular case of that of polyadic into monadic communication leaves open the possibility that, following a similar structure, an encoding of polyadic communication (as the proposed by Milner) might exist for the case of process-passing. In the next section we prove that this is *not* the case.

6.5 Separation Results for Polyadic Communication

In this section we present the separability results for SHO. First, Then, in Section 6.5.1, we introduce the notion of *distinguished forms*, which allow us to capture a number of *stability* *conditions* of processes with respect to their sets of private names. in Section 6.5.2 we present the hierarchy of SHO calculi based on polyadic communication.

6.5.1 Distinguished Forms

Here we define a number of *distinguished forms* for SHO processes. They are intended to capture the structure of processes along communications, focusing on the private names shared among the participants.

6.5.1.1 Definition

The definition of distinguished forms exploits *contexts*, that is, processes with a hole. We shall consider *multi-hole contexts*, that is, contexts with more than one hole. More precisely, a multi-hole context is n-ary if at most n different holes $[\cdot]_1, \ldots, [\cdot]_n$, occur in it. (A process is a 0-ary multi-hole context.) We will assume that any hole $[\cdot]_i$ can occur more than once in the context expression. Notions of free and bound names for contexts are as expected and denoted $\mathrm{bn}(\cdot)$ and $\mathrm{fn}(\cdot)$, respectively.

Definition 6.11. *Syntax of (guarded, multihole) contexts:*

$$\begin{array}{rcl} C, C', \ldots & ::= & a(x).\,D \;\big|\; \bar{a}\langle D\rangle.\,D \\ D, D', \ldots & ::= & [\cdot] \;\big|\; P \;\big|\; C \;\big|\; D \;\; D \;\big|\; \nu r\, D \end{array}$$

Remark 6.2. *We are always working with* non-binding contexts*, i.e., contexts that do not capture the free variables of the processes that fill their holes.*

Below we define *disjoint forms*, the main distinguished form we shall use in the chapter.

Definition 6.12 (Disjoint Form). *Let $T \equiv \nu\widetilde{n}(P \;\; C[\widetilde{R}])$ be a* SHO^m *process where*

1. *$\widetilde{n}$ is a set of names such that $\widetilde{n} \subseteq \mathrm{fn}(P, \widetilde{R})$ and $\widetilde{n} \cap \mathrm{fn}(C) = \emptyset$;*
2. *C is a k-ary (guarded, multihole) context;*
3. *$\widetilde{R}$ contains k closed processes.*

*We then say that T is in k-*adic disjoint form with respect to $\widetilde{n}$, $\widetilde{R}$, and P.

The above definition decrees an arbitrary arity for the context. We shall sometimes say that processes in such a form are in *n-adic disjoint form*, or NDF. By restricting the arity of the context, this general definition can be instantiated:

Definition 6.13 (Monadic Disjoint Form, MDF). *Suppose a process T that is in disjoint form with respect to some $\widetilde{n}$, $\widetilde{R}$, and P. If $\mid \widetilde{R} \mid= 1$ then T is said to be in* monadic disjoint form *(or* MDF*) with respect to $\widetilde{n}$, R, and P.*

Recall that even if MDFs have monadic contexts, the content of the hole (i.e. the single process R) can appear more than once in the process. It could even be the case the content does not appear at all. This is a special case of MDF, as we define below:

Definition 6.14 (Zero-adic Disjoint Form, ZDF). *Let $T \equiv \nu\widetilde{n}\,(P \quad C[R])$ be in MDF with respect to $\widetilde{n}$, R, and P. If $C[R] \not\equiv \emptyset$ and $R = \mathbf{0}$ then T is said to be in* zero-adic disjoint form (ZDF) *with respect to $\widetilde{n}$ and P. Moreover, T can be rewritten as $T \equiv \nu\widetilde{n}_1 P \quad \nu\widetilde{n}_2 Q$, for some $Q \equiv C[\mathbf{0}]$ and for some disjoint sets of names $\widetilde{n_1}$ and $\widetilde{n_2}$ such that both $\widetilde{n} = \widetilde{n_1} \cup \widetilde{n_2}$ and $\widetilde{n_1} \cap \widetilde{n_2} = \emptyset$ hold.*

The following property will be useful in proofs.

Proposition 6.5 (Encodings preserve ZDFs). *Let $[\cdot]$ be an encoding as in Definition 6.9. If T is in ZDF with respect to some $\widetilde{n}$ and P then $[T]$ is in ZDF with respect to $\widetilde{n}$ and $[P]$.*

Proof. We know that, for some Q and $\widetilde{m}$, $T \equiv \nu\widetilde{n}\,P \quad \nu\widetilde{m}\,Q$ is in ZDF with respect to $\widetilde{n}$ and P, and that $\widetilde{n} \cap \widetilde{m} = \emptyset$. By compositionality (Definition 6.7(1)) we have that, for some context C, $[T] = C[[\nu\widetilde{n}\,P], [\nu\widetilde{m}\,Q]]$. The sensible issue here is to ensure that $[\nu\widetilde{n}\,P]$ and $[\nu\widetilde{m}\,Q]$ do not share private names because of the enclosing context C. There are two cases: the first one is that a name that is free in $\nu\widetilde{n}\,P$ but private to $\nu\widetilde{m}\,Q$ becomes private in both $[\nu\widetilde{n}\,P]$ and $[\nu\widetilde{m}\,Q]$ (and the symmetric case); the second case is that a name that is free in both $\nu\widetilde{n}\,P$ and $\nu\widetilde{m}\,Q$ becomes private in both $[\nu\widetilde{n}\,P]$ and $[\nu\widetilde{m}\,Q]$. Proposition 6.1 ensures that none of these cases is possible; for every name a and process R, such a proposition guarantees that if $a \in \mathrm{fn}(R)$ then also $a \in \mathrm{fn}([R])$. As a consequence, even if the context C could involve restrictions enclosing both $[\nu\widetilde{n}\,P]$ and $[\nu\widetilde{m}\,Q]$, such restriction will not bind names in them. Notice that $C[[\nu\widetilde{n}\,P], [\nu\widetilde{m}\,Q]]$ can be rewritten as $[T] \equiv \nu\widetilde{a}([\nu\widetilde{n}\,P] \quad [\nu\widetilde{m}\,Q] \quad S)$, for some process S. Because of the discussion before, names in $\widetilde{a}$ do not bind names in $[P]$ nor in $[Q]$. Hence, $[T]$ is in ZDF with respect to $[P]$ and $\widetilde{n}$, as desired.

□

6.5.1.2 Properties of Disjoint Forms I: Stability Conditions

We are interested in characterizing the transitions that preserve disjoint forms. We focus on internal and output actions. In what follows we discuss properties that apply to arbitrary NDFs; for the sake of readability, however, in proofs we sometimes restrict ourselves to the case of MDFs, since cases for other disjoint forms are analogous and only differ in notational burden.

The following proposition formalizes that, up-to structural congruence, derivatives of NDFs that have unguarded occurrences of some R_i can be brought back into an NDF by "pushing" such occurrences into the side of P of the NDF.

Proposition 6.6. *Suppose a process $T \equiv \nu\widetilde{n}\,(P \quad C[\widetilde{R}])$ such that*

1. *T complies with conditions (1) and (2) in Definition 6.12;*

2. *$\widetilde{R}$ contains k closed processes and $C[\cdot]$ is a context with one or more holes in evaluation context.*

Then, there exists $T' \equiv T$ such that: (i) $T' = \nu\widetilde{n}\,(P' \quad C'[\widetilde{R}])$; (ii) $\mathsf{fn}(P', \widetilde{R}) = \mathsf{fn}(P, \widetilde{R})$ and $\mathsf{fn}(C') = \mathsf{fn}(C)$; (iii) T' is in DF with respect to $\widetilde{n}$, $\widetilde{R}$, and P'.

Proof. We prove the particular case in which T is in MDF (i.e., we have a single R); the proof is analogous for the other disjoint forms. We then need to show that a MDF T' indeed exists. Since T adheres to condition (1) in Definition 6.12, P and R share conditions on names. Without loss of generality, we can assume that $C[R] \equiv \nu\widetilde{n}_2(\prod^k R \quad C'[R])$ where, for a $k \geq 0$, $\prod^k R$ represents the occurrences of R that are in evaluation context, $\widetilde{n}_2 \subseteq \widetilde{n}$ is the set of private names of C, and $C'[R]$ represents the part of C in which each occurrence of R is behind a prefix with names in $\mathsf{fn}(C)$. That is, $C'[\cdot]$ is the subcontext of C in which top-level holes have been removed. Since R and C do not share private names we know that $C[R] \equiv \prod^k R \quad \nu\widetilde{n}_2\, C'[R]$. Consider the process $T' \equiv \nu\widetilde{n}\,(P' \quad C'[R])$, structurally congruent to T' and where $P' = P \quad \prod^k R$. We verify conditions on names for MDFs hold for T': by the above considerations on C', it holds that $\mathsf{fn}(C') = \mathsf{fn}(C)$; also, since P and R share conditions on names it holds that $\mathsf{fn}(P', R) = \mathsf{fn}(P \quad R, R) = \mathsf{fn}(P, R)$. Finally, observe that in C' all occurrences of R remain guarded. We conclude that T' is indeed in MDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P', as desired. □

Disjoint forms are *stable* with respect to internal synchronizations.

Lemma 6.1. *Let $T \equiv \nu\widetilde{n}\,(P \quad C[\widetilde{R}])$ be a process in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P. If $T \xrightarrow{\tau} T'$ then: $T' \equiv \nu\widetilde{n}\,(P' \quad C'[\widetilde{R}])$; $\mathsf{fn}(P', \widetilde{R}) \subseteq \mathsf{fn}(P, \widetilde{R})$ and $\mathsf{fn}(C') \subseteq \mathsf{fn}(C)$; T' is in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P'.*

Proof. We proceed by a case analysis on the communicating partners in the transition.

Transition internal to P. We have a transition $P \xrightarrow{\tau} P'$, and hence $T' \equiv \nu\widetilde{n}\ (P' \quad C[\widetilde{R}])$. The transition is private to P, and as such, $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$. Names in C remain unchanged; we then have that T' is in MDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P', as desired.

Transition internal to $C[\widetilde{R}]$. We have a transition $C[\widetilde{R}] \xrightarrow{\tau} D[\widetilde{R}]$. Since C and $\widetilde{R}$ do not share private names, the transition can only correspond to an internal synchronization on the names private to C. Process $D[\widetilde{R}]$ can have two possible forms, depending on whether or not the prefixes involved in (and consumed by) the transition are guarding some occurrence of R_i. We thus have two cases.

1. In the case $D[\widetilde{R}]$ has no unguarded occurrences of $\widetilde{R}$ (i.e. there are no holes at the top level of the context), we have $D \equiv C'[\widetilde{R}]$, for a context C' that is exactly as C except from two prefixes. The transition concerns only names private to C; hence, $\mathrm{fn}(C') \subseteq \mathrm{fn}(C)$ and the other conditions on names are not affected. We then have that $T' = \nu\widetilde{n}\ (P \quad C'[\widetilde{R}])$ is in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P, as desired.
2. In the case occurrences of some R_i end up unguarded after the transition, with the aid of Proposition 6.6 we infer that T' is structurally congruent to a MDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P, and we are done.

Transition internal to some R_i. This is not possible as by definition of disjoint form, every occurrence of $\widetilde{R}$ in $C[\widetilde{R}]$ is underneath a prefix.

Communication between P and $C[\widetilde{R}]$. This is not possible since by definition of disjoint form, P and C do not share private names. No R_i can evolve, thus there cannot be a communication between P and any R_i.

□

Corollary 6.1. *Let T be a process in ZDF with respect to some $\widetilde{n}$ and P. If $T \xrightarrow{\tau} T'$, then T' is in ZDF with respect to $\widetilde{n}$ and P too.*

The lemma below asserts that disjoint forms are stable also under output actions that do not involve extrusion of names. To see this, consider a MDF T: the only risk for it after an output action is that the R in $C[R]$ could be communicated, therefore “downgrading” the MDF into a ZDF. Since, as we have seen, ZDFs are a special case of MDFs, this is not a problem and MDFs are preserved. Below we say a process P is *contained* in a process Q if and only if there exists a context C such that $Q \equiv C[P]$.

Lemma 6.2. *Let $T \equiv \nu\widetilde{n}\,(P \quad C[\widetilde{R}])$ be a process in NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P. If $T \xrightarrow{\overline{a}\langle Q\rangle} T'$ then: there exist P' and C' so that $T' \equiv \nu\widetilde{n}\,(P' \quad C'[\widetilde{R}])$; both $\mathrm{fn}(P', \widetilde{R}) \subseteq \mathrm{fn}(P, \widetilde{R})$ and $\mathrm{fn}(C') \subseteq \mathrm{fn}(C)$ hold; T' is in MDF with respect to $\widetilde{n}$, $\widetilde{R}$, and P'.*

Proof. By a case analysis on the source of the action. We prove the particular case in which T is in MDF; the proof is analogous for the other disjoint forms.

- If $P \xrightarrow{\overline{a}\langle Q\rangle} P'$ then $T' \equiv \nu\widetilde{n}\,(P' \quad C[R])$. Since P' is contained in P, we have $\mathrm{fn}(P', R) \subseteq \mathrm{fn}(P, R)$. Conditions on names in $\mathrm{fn}(C)$ are unchanged, and we have that T' is in MDF with respect to $\widetilde{n}$, R, and P', as desired.

- If $C[R] \xrightarrow{\overline{a}\langle Q\rangle} D[R]$ then we reason on k, the number of guarded occurrences of R in $D[R]$. The thesis is immediate for $k > 0$; if $k = 0$ then $D[R]$ is actually in ZDF with respect to $\widetilde{n}$ and P; by recalling that a ZDF is a special case of MDF we are done.

□

The following property formalizes the consequences public synchronizations have on ZDFs.

Lemma 6.3. *Let T be a* SHO^n *process in ZDF with respect to $\widetilde{n}$ and P. Suppose $T \xrightarrow{a\tau} T'$ where $\xrightarrow{a\tau}$ is a public n-adic synchronization. Then T' is in n-adic disjoint form with respect to $\widetilde{n}$, some $\widetilde{R}$, and P.*

Proof. The proof proceeds by a case analysis on the rule used to infer $\xrightarrow{a\tau}$. We concentrate on the case in which $\xrightarrow{a\tau}$ is a monadic public synchronization, and arises from interaction of two processes that do not share private names; the other cases are similar or simpler. There are two cases, corresponding to rules Tau1 and Tau2. We analyze the first one. Without loss of generality, we can assume $T \equiv \nu\widetilde{n}_1 P \quad \nu\widetilde{n}_2 Q$, which is in ZDF with respect to $\widetilde{n}_1 \cup \widetilde{n}_2$ and P. In T, we have that $P = \overline{a}\langle R\rangle.\, P' \quad P''$, $Q = a(x).\, Q' \quad Q''$, and $\widetilde{n}_1$, $\widetilde{n}_2$ are two disjoint sets of names. We then have $\nu\widetilde{n}_1\, P \xrightarrow{(\nu\widetilde{n}'_1)\overline{a}\langle R\rangle} \nu\widetilde{n}_1\, P'$ (with $\widetilde{n}'_1 \subseteq \widetilde{n}_1$) and $\nu\widetilde{n}_2\, Q \xrightarrow{a(x)} \nu\widetilde{n}_2\, Q'$. That is, we are assuming the case in which the output on a extrudes some private names $\widetilde{n}'_1$. Using rule Tau1 we obtain $\nu\widetilde{n}_1 P \quad \nu\widetilde{n}_2\, Q \xrightarrow{a\tau} \nu\widetilde{n}_1\, P' \quad \nu\widetilde{n}'_1\widetilde{n}_2\, Q'\{R/x\} = T'$. By noticing that $\widetilde{n}'_1 \subseteq \widetilde{n}$ we have that $T' \equiv \nu\widetilde{n}_1\, (P' \quad \nu\widetilde{n}_2\, Q'\{R/x\})$, so T' can be brought into a MDF with respect to $\widetilde{n}_1$, R, and some P'. First, consider the context that is obtained by replacing each occurrence of x in Q with a single hole. Call that context $C[\cdot]$; since we have monadic communication, C is *monadic*. We can then see that $\nu\widetilde{n}_2\, Q'\{R/x\}$ corresponds to $C[R]$. The resulting process can be written as $\nu\widetilde{n}_1\, (P' \quad C[R])$; in case there are unguarded occurrences of R in $C[R]$ (because of top-level occurrences of x in Q), with the aid of Proposition 6.6, the process can be rewritten as a MDF with respect to $\widetilde{n}_1$, R, and some P'' containing both P' and a number of occurrences of R.

The case for Tau2 is completely analogous, and only differs in the fact that the process after the public synchronization is in MDF with respect to $\widetilde{n}_2$ (rather than to $\widetilde{n}_1$). □

6.5.1.3 Properties of Disjoint Forms II: Origin of Actions

We now give some properties regarding the order and origin of internal and output actions of processes in DFs.

Definition 6.15. *Let $T = \nu\widetilde{n}\,(A \quad C[\widetilde{R}])$ be an NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \xrightarrow{\alpha} T'$ for some action α.*

- *Let α be an output action. We say that α* originates in *A if $A \xrightarrow{\alpha} A'$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$, and that α* originates in *C if $C[\widetilde{R}] \xrightarrow{\alpha} C'[\widetilde{R}]$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$.*

- *Let $\alpha = \tau$. We say that α* originates in *A if, for some $a \in \widetilde{n}$, $A \xrightarrow{a\tau} A'$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$, and that α* originates in *C if $C[\widetilde{R}] \xrightarrow{\tau} C'[\widetilde{R}]$ occurs as a premise in the derivation of $T \xrightarrow{\alpha} T'$.*

Proposition 6.7. *Let $T = \nu\widetilde{n}\,(A \quad C[\widetilde{R}])$ be an NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \xrightarrow{\alpha} T'$, where α is either an output action or an internal synchronization. Then α originates in* either *A or C.*

Proof. The thesis is immediate for the case of output actions. For internal synchronizations the thesis follows by noting that by definition internal synchronizations take place on private names only. By definition of MDF, A and C do not share private names, and all occurrences of $\widetilde{R}$ in context C are guarded, so they cannot interact with A. As a result, there is no way A and C can interact through an internal synchronization; such an action must originate in either A or C. □

Notice that both A and C can have the same action α (for instance, an output action on a public name that is shared among them). This, however, does not mean that a single instance of α originates in both A and C.

The following proposition says that the only consequence an internal transition originated in C might have on the structure of an NDF is to release new copies of the processes in $\widetilde{R}$:

Proposition 6.8. *Let $T = \nu\widetilde{n}\,(A \quad C[\widetilde{R}])$ be a NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \xrightarrow{\tau} T'$, where τ originates in C. Then, for some $k_1, \ldots, k_n \geq 0$, $T' \equiv \nu\widetilde{n}\,(A \quad C'[\widetilde{R}] \quad \prod^{k_1} R_1 \quad \cdots \quad \prod^{k_n} R_n)$.*

Proof. Immediate by recalling that by definition of MDF occurrences of $\widetilde{R}$ appear guarded in $C[\widetilde{R}]$, and by noticing that an internal synchronization consumes two (complementary) prefixes. The number of copies of any R_i (for $i \in 1..n$) is greater than zero if the prefixes involved in the synchronization guard an occurrence of R_i. □

The following lemma states the conditions under which two actions of a disjoint form can be safely *swapped.*

Lemma 6.4 (Swapping Lemma). *Let $T = \nu\widetilde{n}\,(A \quad C[\widetilde{R}])$ be an NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Consider two actions α and β that can be either an output action or an internal synchronization. Suppose that α originates in A, β originates in C, and that there exists a T' such that $T \xrightarrow{\alpha}\xrightarrow{\beta} T'$. Then $T \xrightarrow{\beta}\xrightarrow{\alpha} T'$ also holds, i.e., action β can be performed before α without affecting the final behavior.*

Proof. We proceed by a case analysis on α and β, analyzing their possible combinations. Since we have two kinds of actions (output actions and internal synchronizations), we have four cases to check. All of them are easy, and follow by the semantics of parallel composition. Consider, for instance, in the case in which $\alpha = \tau$ through a synchronization on private name

a, and $\beta = \tau$ through a synchronization on private name a. Then, for some complementary actions $\alpha_0, \overline{\alpha_0}$ on (private) name a, and complementary actions $\beta_0, \overline{\beta_0}$ on (private) name b, we have that

$$\begin{aligned} T &\equiv \nu\widetilde{n}\,(\alpha_0.\,A_1 \;\; \overline{\alpha_0}.\,A_2 \;\; A' \;\; \beta_0.\,C_1[\widetilde{R}] \;\; \overline{\beta_0}.\,C_2[\widetilde{R}] \;\; C'[\widetilde{R}]) \text{ and} \\ T' &\equiv \nu\widetilde{n}\,(A_1 \;\; A_2 \;\; A' \;\; C_1[\widetilde{R}] \;\; C_2[\widetilde{R}] \;\; C'[\widetilde{R}]) \end{aligned}$$

By definition of internal synchronizations, a is a name private to A and b is a name private to C. Since by definition of MDF A and C do not share private names, then there is no possibility for interferences between the prefixes $\alpha_0, \overline{\alpha_0}, \beta_0$, and $\overline{\beta_0}$. Hence, it is safe to perform $T \xrightarrow{\beta}\xrightarrow{\alpha} T'$, and the thesis holds. □

Notice that the converse of the Swapping Lemma does not hold: since an action β originated in C can enable an action α originated in A (e.g., an action enabled by an extra copy of R), these cannot be swapped. We now generalize the Swapping Lemma to a sequence of internal synchronizations and output actions.

Lemma 6.5 (Commuting Lemma). *Let $T = \nu\widetilde{n}\,(A \;\; C[\widetilde{R}])$ be a NDF with respect to $\widetilde{n}$, $\widetilde{R}$, and A. Suppose $T \overset{\vec{\alpha}}{\Longrightarrow} T'$, where $\vec{\alpha}$ is a sequence of output actions and internal synchronizations only. Let $\vec{\alpha}_C$ (resp. $\vec{\alpha}_A$) be the sequence of actions that is exactly as $\vec{\alpha}$ but in which actions originated in A (resp. C) or its derivatives are not included. Then, there exists a process T_1 such that*

1. $T \overset{\vec{\alpha}_C}{\Longrightarrow} T_1 \overset{\vec{\alpha}_A}{\Longrightarrow} T'$.

2. $T_1 \equiv \nu\widetilde{n}\,(A \;\; \prod^{m_1} R_1 \;\; \cdots \;\; \prod^{m_k} R_n \;\; C'[\widetilde{R}])$, *for some* $m_1, \ldots, m_k \geq 0$.

Proof. We proceed by an induction on k, the number of actions originated in C that occur after an action originated in A in the sequence $\vec{\alpha}$. The base case is when $k = 0$; that is, when all the actions after T_1 are originated in A, and we are done. The inductive step requires a second induction on j, the number of actions originated in A which precede a single action originated in C. This induction follows easily exploiting the Swapping Lemma (Lemma 6.4). The fact that, for each $i \in 1..n$, T_1 involves a number $m_i \geq 0$ of copies of R_i is an immediate consequence of Proposition 6.8. □

6.5.2 A Hierarchy of Synchronous Higher-Order Process Calculi

We define an expressiveness hierarchy for the higher-order process calculi in the family given by SHO. The hierarchy is defined in terms of the impossibility of encoding SHO^n into SHO^{n-1}, according to the definition given in Section 6.3. We begin by showing the impossibility result that sets the basic case of the hierarchy, namely that biadic process passing cannot be encoded into monadic process passing (Lemma 6.6). The proof exploits the notion of MDF and its associated stability properties. We then state the general result, i.e. the impossibility of encoding SHO^{n+1} into SHO^{n} (Lemma 6.7); this is done by generalizing the proof of Lemma 6.6.

Lemma 6.6. *There is no encoding of* SHO^2 *into* SHO^1*.*

Proof. Assume, towards a contradiction, that an encoding $[\![\cdot]\!] : \mathsf{SHO}^2 \to \mathsf{SHO}^1$ does indeed exist. In what follows, we use i, j to range over $\{1, 2\}$, assuming that $i \neq j$.

Assume processes $S_i = \overline{m_i} \quad m_i.\,\overline{s_i}$ and $S_j = \overline{m_j} \quad m_j.\,\overline{s_j}$. Consider the SHO^2 process $P = E^{(2)} \quad F^{(2)}$, where $E^{(2)}$ and $F^{(2)}$ are defined as follows:

$$\begin{aligned} E^{(2)} &= \nu m_1, m_2\,(\overline{a}\langle\langle\rangle S_1,\, S_2\rangle.\,\mathbf{0}) \\ F^{(2)} &= \nu b\,(a(x_1, x_2).\,(\overline{b}\langle\langle\rangle \overline{b_1}.\,x_1\rangle.\,\mathbf{0} \quad \overline{b}\langle\langle\rangle \overline{b_2}.\,x_2\rangle.\,\mathbf{0} \quad b(y_1).\,b(y_2).\,y_1)) \end{aligned}$$

where both $b_1, b_2 \notin \mathrm{fn}(E^{(2)})$ (with $b_1 \neq b_2$) and $s_1, s_2 \notin \mathrm{fn}(F^{(2)})$ (with $s_1 \neq s_2$) hold. Let us analyze the behavior of P. We first have a public synchronization on a:

$$P \xrightarrow{a\tau} \nu m_1, m_2, b\,(\overline{b}\langle\langle\rangle \overline{b_1}.\,S_1\rangle.\,\mathbf{0} \quad \overline{b}\langle\langle\rangle \overline{b_2}.\,S_2\rangle.\,\mathbf{0} \quad b(y_1).\,b(y_2).\,y_1) = P_0\,.$$

In P_0 we have two private synchronizations on name b that implement an internal choice: both processes $\overline{b_1}.\,S_1$ and $\overline{b_2}.\,S_2$ are consumed but only one of them will be executed. We then have either $P_0 \xrightarrow{\tau}\xrightarrow{\tau} \overline{b_1}.\,S_1 = P_1$ or $P_0 \xrightarrow{\tau}\xrightarrow{\tau} \overline{b_2}.\,S_2 = P_1'$. Starting in P_1 and P_1' we have the following sequences of actions:

$$\begin{aligned} P_1 &\xrightarrow{\overline{b_1}} P_2 \xrightarrow{\tau}\xrightarrow{\overline{s_1}} \mathbf{0} \\ P_1' &\xrightarrow{\overline{b_2}} P_2' \xrightarrow{\tau}\xrightarrow{\overline{s_2}} \mathbf{0}\,. \end{aligned}$$

In both cases, a private synchronization on m_i precedes an output action on s_i. All the above can be summarized as follows:

$$P \xrightarrow{a\tau} P_0 \xrightarrow{\tau}\xrightarrow{\tau} P_1 \xrightarrow{\overline{b_1}} P_2 \xrightarrow{\tau}\xrightarrow{\overline{s_1}} \mathbf{0} \tag{6.1}$$

$$P \xrightarrow{a\tau} P_0 \xrightarrow{\tau}\xrightarrow{\tau} P_1' \xrightarrow{\overline{b_2}} P_2' \xrightarrow{\tau}\xrightarrow{\overline{s_2}} \mathbf{0}\,. \tag{6.2}$$

These sequences of actions might help to appreciate the effects of the internal choice on b, discussed above. Such a choice has direct influence on: (i) the output action on b_i, (ii) the internal synchronization on m_i, and (iii) the output action on s_i. Notice that each of these actions enables the following one, and that an output on b_i precludes the possibility of actions on b_j, m_j, and s_j.

Consider now the behavior of $[\![P]\!]$ —the encoding of P— with the aid of (6.1) and (6.2) above. By definition of encoding (in particular, completeness) we have the following two, mutually exclusive, possibilities for behavior:

$$[\![P]\!] \xRightarrow{a\tau} \approx [\![P_0]\!] \Rightarrow \approx [\![P_1]\!] \xRightarrow{\overline{b_1}} \approx [\![P_2]\!] \xRightarrow{\overline{s_1}} \approx \mathbf{0} \quad \text{and} \tag{6.3}$$

$$[\![P]\!] \xRightarrow{a\tau} \approx [\![P_0]\!] \Rightarrow \approx [\![P_1']\!] \xRightarrow{\overline{b_2}} \approx [\![P_2']\!] \xRightarrow{\overline{s_2}} \approx \mathbf{0}\,. \tag{6.4}$$

We notice that the first (weak) transition, namely

$$[P] \overset{a\tau}{\Longrightarrow} \approx [P_0] \, ,$$

is the same in both possibilities. Let us analyze it, by relying on Definition 6.3. For SHO^1 processes T, T', and T_0, it holds

$$[P] \Rightarrow T \xrightarrow{a\tau} T' \Rightarrow T_0 \approx [P_0] \, . \tag{6.5}$$

We examine the distinguished forms in the processes in (6.5). We notice that P is in ZDF with respect to $\{m_1, m_2, b\}$ and $E^{(2)}$: m_1, m_2 do not appear in $F^{(2)}$, and b does not appear in $E^{(2)}$. From Proposition 6.5 we know that $[P]$ is also in ZDF with respect to $\{m_1, m_2, b\}$ and $[E^{(2)}]$. Since DFs are preserved by internal actions (Corollary 6.1), we know that T is also a ZDF with respect to $\{m_1, m_2, b\}$ and A, the derivative of $[E^{(2)}]$. In the general case, Lemma 6.3 ensures that a public synchronization causes a ZDF to become a MDF. In this case, we have a communication from $E^{(2)}$ to $F^{(2)}$ which is mimicked by the encoding; we then have that T' is in MDF with respect to $\{m_1, m_2\}$, some $R \not\equiv \mathbf{0}$, and A', the derivative of A after the public synchronization. Finally, since T' evolves into T_0 by means of internal synchronizations only, by Lemma 6.1, we know that T_0 is also in MDF with respect to $\{m_1, m_2\}$, R, and A_0, the derivative of A'. Indeed, for some context C_0 (with private name b), we have that

$$T_0 = \nu m_1, m_2 \, (A_0 \;\; C_0[R]) \; .$$

Notice that (6.5) ensures that process $T_0 \approx [P_0]$. Hence, by definition of $\approx$, T_0 should be able to match each action possible from $[P_0]$ by performing either the sequence of actions given in (6.3) or the one in (6.4). We have just seen that T_0 is in MDF with respect to $\{m_1, m_2\}$, R, and A_0. Crucially, both (6.3) and (6.4) involve only internal synchronizations and output actions. Therefore, by Lemmas 6.1 and 6.2, every derivative of T_0 intended to mimic the behavior of $[P_0]$ (and its derivatives) is a process in MDF with respect to $\{m_1, m_2\}$, R, and some A_i.

We now use this information on the structure of the derivatives of T_0 to analyze the bisimulation game for $T_0 \approx [P_0]$. We use the observability predicates (barbs) as in Definition 6.3. We know from (6.3) and (6.4) that $[P_0]$ evolves into either $[P_1]$ or $[P_1']$ after a weak transition. The encoding preserves the mutually exclusive, internal choice that was discussed for the source term P_0; in the encoding such a choice is governed by the encoding of $F^{(2)}$. Also, as in the source term, the output barb on b_i (resp. b_j) available in $[P_1]$ (resp. $[P_1']$) is enough to recognize the result of such a choice. Process T_0 should be capable of mimicking this internal choice, and there should exist derivatives T_1 and T_1' of T_0 such that both $T_0 \Rightarrow T_1$ with $T_1 \approx [P_1]$ and $T_0 \Rightarrow T_1'$ with $T_1' \approx [P_1']$ hold.

Consider now the behavior from $[P_1]$, one of the two possible derivatives of $[P_0]$ (given in (6.3)). After a weak output transition on b_1, the process evolves into one that is behaviorally equivalent to $\llbracket P_2 \rrbracket$. This output barb gives evidence on the internal choice that took place in $\llbracket P_0 \rrbracket$. Recall that such a choice was a mutually exclusive choice: therefore, once an output barb on b_1 is performed, the possibility of an output barb on b_2 is precluded. By definition of $\approx$, process T_1 should be able to perform a weak output transition on b_1, thus evolving into a process T_2 behaviorally equivalent to $\llbracket P_2 \rrbracket$. The behavior from $\llbracket P_1' \rrbracket$ (the other derivative of $\llbracket P_0 \rrbracket$, given in (6.4)) is similar: after a weak output transition on b_2, the process evolves into a process behaviorally equivalent to $\llbracket P_2' \rrbracket$. The SHO^1 process T_1' should mimic this behavior as expected, and evolve into a T_2' such that $T_2' \approx \llbracket P_2' \rrbracket$. Since MDFs are preserved by output action (Lemma6.2) both T_2 and T_2' are in MDF with respect to $\{m_1, m_2\}$, R, and some A_i.

To complete the bisimulation game, we have that T_2 and T_2' should be able to match the internal synchronizations and output actions that are performed by $\llbracket P_2 \rrbracket$ and $\llbracket P_2' \rrbracket$, respectively. Summing up we have the following behavior from T_0:

$$T_0 \Rightarrow T_1 \xRightarrow{\overline{b_1}} T_2 \xRightarrow{\overline{s_1}} \approx \mathbf{0} \quad \text{and} \tag{6.6}$$

$$T_0 \Rightarrow T_1' \xRightarrow{\overline{b_2}} T_2' \xRightarrow{\overline{s_2}} \approx \mathbf{0}. \tag{6.7}$$

where, by definition of $\approx$, $\llbracket P_i \rrbracket \approx T_i$ for $i \in \{0, 1, 2\}$ and $\llbracket P_j' \rrbracket \approx T_j'$ for $j \in \{1, 2\}$. Call C_2 and C_2' to the derivatives of C_0 in T_2 and T_2', respectively. It is worth noticing that by conditions on names, output actions on s_1 and s_2 cannot originate in C_2 and C_2'.

The behavior of T_0 described in (6.6) and (6.7) can be equivalently described as $T_0 \xRightarrow{\alpha_1} \mathbf{0}$ and $T_0 \xRightarrow{\alpha_2} \mathbf{0}$, where α_1 contains outputs on b_1 and s_1, and α_2 contains outputs on b_2 and s_2, respectively. Using the Commuting Lemma (Lemma 6.5) on T_0, we know there exist processes T_1^*, and T_2^* such that

1. $T_1^* \equiv \nu \widetilde{n}\,(A_0 \;\; \prod^{m} R \;\; C_1^*[R])$ and $T_2^* \equiv \nu \widetilde{n}\,(A_0 \;\; \prod^{m'} R \;\; C_2^*[R])$, for some $m, m' \geq 0$. Recall that these processes are obtained by performing every action originated in C_0 (which can only be output actions and internal synchronizations); as a result, we have that $C_1^*[R] \not\rightarrow$ and $C_2^*[R] \not\rightarrow$.

2. T_1^* (resp. T_2^*) can only perform an output action on s_1 (resp. s_2) and internal actions. Considering this, we have that $T_1^* \Downarrow_{\overline{s_1}}$, $T_1^* \not\Downarrow_{\overline{s_2}}$ and $T_2^* \Downarrow_{\overline{s_2}}$, $T_2^* \not\Downarrow_{\overline{s_1}}$ should hold.

From item (1) above it is easy to observe that the only difference between T_1^* and T_2^* is in m and m', the number of copies of R released as a result of executing first all actions originating in C_0. We then find that the number of copies of R has direct influence on performing an output action on s_1 or on s_2; in turn, this has influence on the bisimulation game between $\llbracket P_2 \rrbracket$ and T_2, and that between $\llbracket P_2' \rrbracket$ and T_2'. We consider three possible cases for the value of m and m':

Case 1: $m = m'$. This is not a possibility, since it would imply that both T_1^* and T_2^* have the same possibilities of behavior, i.e., that outputs on both s_1 and s_2 are possible from T_1^* and T_2^*. Clearly, this breaks the bisimilarity condition.

Case 2: $m > m'$. Consider the process T_1^*. We have already seen that in order to play correctly the bisimulation game, it must be the case that $T_1^* \Downarrow_{\overline{s_1}}$ and $T_1^* \not\Downarrow_{\overline{s_2}}$. Process T_1^* has more copies of R than T_2^*; we can thus rewrite it as

$$T_1^* \equiv \nu\widetilde{n}\,(A_0 \ \prod^{m'} R \ \prod^{m-m'} R \ \ C_1^*[R])\,.$$

Considering that $C_1^*[R] \not\rightarrow$ and $C_2^*[R] \not\rightarrow$, we can formally state that the $m - m'$ copies of R in T_1^* are the only behavioral difference between T_1^* and T_2^*, i.e.

$$T_1^* \approx T_2^* \ \prod^{m-m'} R\,. \tag{6.8}$$

Let us analyze the consequences of this relationship between T_1^* and T_2^*. As argued before, it must be the case that $T_1^* \Downarrow_{\overline{s_1}}$ and $T_2^* \Downarrow_{\overline{s_2}}$ should hold. Notice that because of (6.8), if $T_2^* \Downarrow_{\overline{s_2}}$ then $T_1^* \Downarrow_{\overline{s_2}}$ holds. This would break the bisimilarity game between $[\![P_2]\!]$ and T_2, since $[\![P_2]\!] \not\Downarrow_{\overline{s_2}}$. Even in the (contradictory) case that $T_2^* \Downarrow_{\overline{s_2}}$ would not hold, the bisimilarity game between $[\![P_2]\!]$ and T_2 would succeed, but the game between $[\![P_2']\!]$ and T_2' would fail, as $[\![P_2']\!]$ could perform an output on s_2 that T_2 could not match. Hence, in the case $m > m'$ the bisimilarity game would fail.

Case 3: $m < m'$. This case is completely symmetric to Case 2.

This analysis reveals that there is no way a MDF can faithfully mimic the observable behavior of a SHO^2 process when such a behavior depends on internal choices implemented with private names. We then conclude that there is no encoding $[\![\cdot]\!] : \mathrm{SHO}^2 \to \mathrm{SHO}^1$. □

The scheme used in the proof of Lemma 6.6 can be generalized for calculi with arbitrary polyadicity. Therefore we have the following.

Lemma 6.7. *For every $n > 1$, there is no encoding of* SHO^n *into* SHO^{n-1}.

Proof. The proof proceeds by contradiction, assuming an encoding $[\![\cdot]\!] : \mathrm{SHO}^n \to \mathrm{SHO}^{n-1}$ indeed exists. Consider the SHO^n process $P = E^{(n)} \ \ F^{(n)}$, where $E^{(n)}$ and $F^{(n)}$ are defined as follows:

$$\begin{aligned} E^{(n)} &= \nu m_1, \ldots, m_n\,(\overline{a}\langle\langle\rangle S_1, \ldots, S_n\rangle.\,\mathbf{0}) \\ F^{(n)} &= \nu b\,(a(x_1, \ldots, x_n).\,(\overline{b}\langle\langle\rangle \overline{b_1}.\,x_1\rangle.\,\mathbf{0} \ \cdots \ \overline{b}\langle\langle\rangle \overline{b_n}.\,x_n\rangle.\,\mathbf{0} \ \ b(y_1).\cdots.b(y_n).\,y_1) \end{aligned}$$

where, for each $l \in 1..n$, $S_l = \overline{m_l} \ \ m_l.\,\overline{s_l}$. Also, $b_1, \ldots, b_n$ are pairwise different names not in $\mathrm{fn}(E^{(n)})$ and $s_1, \ldots, s_n$ are pairwise different names not in $\mathrm{fn}(F^{(n)})$.

Using this P, the analysis follows the same principles and structure than the proof of Lemma 6.6. After a public synchronization on a, P evolves into some P_0. In P_0 there are n internal synchronizations on the private name b, which implement an internal, mutually exclusive choice and lead to the execution of one (and only one) of the $\overline{b_l}.\,S_l$. In the encoding side, using Proposition 6.5, the SHO^{n-1} process $[\![P]\!]$ can be shown to be in ZDF with respect to $\{m_1, \ldots, m_n, b\}$ and $[\![E^{(n)}]\!]$; using Corollary 6.1 and the generalization of Lemma 6.3 to the case of a public $(n-1)$-adic synchronization, $[\![P_0]\!]$ can be shown to be behaviorally equivalent to a process T_0 that is in $(n-1)$-adic disjoint form with respect to $\{m_1, \ldots, m_n\}$, some $R_1, \ldots, R_{n-1}$, and some A_0.

The analysis of the bisimulation game $T_0 \approx [\![P_0]\!]$ is similar as before; the only difference is that now there are n alternatives for an output action on some b_i which enables an output action on s_i. Process T_0 should be able to match any such actions; this exploits the fact that along the bisimulation game the $(n-1)$-adic disjoint form is preserved (by Lemmas 6.1 and 6.2). The Commuting Lemma (Lemma 6.5, which holds for arbitrary NDFs) can be then applied to show that the $n-1$-adic disjoint form T_0 might perform some observable behavior that $[\![P_0]\!]$ is not able to perform. In particular, if $[\![P_0]\!]$ executes only some $\overline{b_l}.\,S_l$, T_0 could exhibit *also* barbs associated to some $\overline{b_k}.\,S_k$, where $k \in 1..n$ and $k \neq l$. This leads to a contradiction, and the thesis holds. □

Remark 6.3 (A hierarchy for *asynchronous* calculi)**.** *The expressiveness hierarchy characterized by Lemma 6.7 for calculi in* SHO *holds for calculi in* AHO *as well. In fact, a detailed proof would simply consist in adapting the definition of guarded contexts (Definition 6.11), the stability lemmas (Lemmas 6.1 and 6.2), the conditions under which the Swapping Lemma holds (Lemma 6.4), and the counterexample used in Lemma 6.6. Roughly speaking, there are no substantial differences between the synchronous and the asynchronous case: having one less prefix does not change the main structure of the proof; the definition of disjoint form becomes somewhat weaker, as copies of the process inside context would be only released after an input action.*

6.6 The Expressive Power of Abstraction Passing

In this section we show that abstraction passing, i.e., parameterizable processes, is strictly more expressive than process passing. We consider SHO^n_a, the extension of SHO^n with the communication of abstractions of one level of arrow nesting, i.e., functions from processes into processes. The language of SHO^n_a processes is obtained by extending the syntax of SHO processes (Definition (6.4) in the following way:

$$P, Q, \ldots ::= \cdots \;\Big|\; (x)P \;\Big|\; P_1 \lfloor P_2 \rfloor$$

That is, we consider abstractions of the form $(x)P$ and *applications* of the form $P_1\lfloor P_2 \rfloor$, that allows to assign an argument P_2 to an abstraction P_1. As usual, $(x_1)\ldots(x_n)P$ is abbreviated as $(x_1, \ldots, x_n)P$. The operational semantics of SHO^n_a is that of SHO, extended with the following rule:

$$\textsc{App} \quad \frac{}{(x)P\lfloor Q \rfloor \xrightarrow{\tau} P\{Q/x\}}\,.$$

Moreover, for SHO^n_a we rely in notions of types and as in (Sangiorgi, 1996b), and consider only well-typed processes.

Example 6.1 (Private Link Establishment with Abstraction Passing). *Let us introduce a very simple example of the way in which abstraction passing is able to model private link establishment on a name. Consider the* SHO^1_a *process* $P = S \;\; R$*, where* S *and* R *are defined as follows:*

$$\begin{aligned} S &= \nu s\,(\overline{a}\langle (y)\overline{s}\langle y \rangle\rangle.\, s(x).\, x) \\ R &= a(x).\, x\lfloor Q \rfloor\,. \end{aligned}$$

We then have that a private link between S *and* R *is created once they synchronize on* a*; the private link is used to send* Q *from the derivative of* R *to that of* S*:*

$$\begin{aligned} P &\xrightarrow{a\tau} \nu s\,(s(x).\, x \;\; (y)\overline{s}\langle y \rangle\lfloor Q \rfloor) \\ &\xrightarrow{\tau} \nu s\,(s(x).\, x \;\; \overline{s}\langle Q \rangle) \\ &\xrightarrow{\tau} Q\,. \end{aligned}$$

We now show that abstraction passing increases the expressive power of pure process passing in SHO. The result is based on the encoding below.

Definition 6.16 (Monadic abstraction-passing can encode polyadic communication). *The encoding* $[\cdot] : \mathsf{SHO}^2 \to \mathsf{SHO}^1_a$ *is defined as:*

$$\begin{aligned} [\overline{a}\langle\langle\rangle P_1, P_2\rangle.\, R] &= a(z).\,([R] \;\; \nu m, n, c\,(\overline{n} \;\; z\lfloor n.\,(\overline{c} \;\; \overline{m}) + m.\,([P_1] \;\; \overline{m})\rfloor \;\; c.\, z\lfloor [P_2] \rfloor)) \\ [a(x_1, x_2).\, Q] &= \nu b\,(\overline{a}\langle\langle\rangle(y)\overline{b}\langle y \rangle\rangle \;\; b(x_1).\,(x_1 \;\; b(x_2).\,[Q])) \end{aligned}$$

where $[\cdot]$ *is an homomorphism for the other operators in* SHO^2*.*

Definition 6.16 can be generalized so as to obtain an encoding $[\cdot] : \mathsf{SHO}^n \to \mathsf{SHO}^1_a$, for any $n > 1$. This encoding leads to the following separation result:

Lemma 6.8. *There is no encoding of* SHO^n_a *into* SHO^n*.*

Proof. Let us just consider the case $n = 1$; the other cases are similar. Suppose, for the sake of contradiction, there is an encoding $\mathcal{A}[\cdot] : \mathsf{SHO}^1_a \to \mathsf{SHO}^1$. By Definition 6.16, we know there is an encoding $\mathcal{B}[\cdot] : \mathsf{SHO}^2 \to \mathsf{SHO}^1_a$. Since the composition of two encodings is an encoding (Proposition 6.3), this means that $(\mathcal{A} \cdot \mathcal{B})[\cdot]$ is an encoding of SHO^2 into SHO^1. However, by Lemma 6.7 we know such an encoding does not exist, and we reach a contradiction. □

6.7 Concluding Remarks

Summary. In first-order process calculi such as the π-calculus both (a)synchronous and polyadic communication are well-understood mechanisms; they rely on the ability of establishing *private links* for process communications that are robust with respect to external interferences. Such an ability is natural to first-order process calculi, as it arises from the interplay of restriction and name passing. In this chapter we have studied synchronous and polyadic communication and their representability in higher-order process calculi *with restriction* but *without name-passing*. Central to our study is the invariance of the set of private names of a process along certain computations. We have studied two *families* of higher-order process calculi: the first one, called AHO, extends HOcore with restriction and polyadic communication; the second, called SHO, replaces asynchronous communication in AHO with synchronous communication. Each define calculi with different arity in communications, denoted AHO^n and SHO^n, respectively. Our first contribution was an *encodability* result of SHO^n into AHO^n. Such an encoding bears witness of the expressive power of the process passing communication paradigm and gives insights on how to represent certain scenarios using process passing only. With this positive result, we moved to analyze polyadic communication. We showed that in the case of polyadicity the absence of name-passing does entail a loss in expressiveness; this is represented by the non-existence of an encoding of SHO^n into SHO^{n-1}. This *non-encodability* result is our second main contribution; it determines a *hierarchy* of higher-order process calculi based on the arity allowed in process passing communications. Finally, we showed that unlike process passing, *abstraction passing* provides a way of establishing private links. As a matter of fact, we showed an encoding of SHO^n into SHO^1 extended with abstraction passing, and used such a result to prove our final contribution: the non-existence of an encoding of abstraction passing into process passing of any arity.

More on the Notion of Encoding. It has become increasingly accepted that a unified, all-embracing notion of encoding that serves all purposes is unlikely to exist, and that the exact definition of encoding should depend on the particular purpose. This way, for instance, the kind of criteria adopted in encodability results is usually different from those generally present in separation results. In this chapter we have adopted a notion of encoding that is arguably more demanding than those previously proposed in the literature for separation results. We argue that such a definition is in line with our overall goal, that of assessing the expressiveness of higher-order concurrency with respect to (a)synchrony and polyadicity and, most importantly, in the absence of name passing.

Interferences are a major concern in our setting, essentially because the absence of name passing leaves us without suitable mechanisms for establishing private links. Devising a definition of encoding so as to incorporate a notion of potentially malicious context (including techniques for reasoning over *every possible* context) appears very challenging. To this end, we combine suitable elements from the operational semantics and from the definition of encoding. It could be rightly argued that not all interferences are necessarily harmful, and in this sense our approach to interference handling would appear too coarse. We would like to stress on the difficulties inherent to only *considering* interferences; attempting to both *considering* and *handling* them in a selective way seems much more challenging. Also, even if we do not actually prove that encodings behave correctly under every possible context, we think that our approach is an initial effort in that direction.

Notice that we do not claim our notion of encoding should be taken as a reference for other separation results; it simply intends to capture the —rather strong— correctness requirements (i.e. compositionality and robustness with respect to interferences) which we consider appropriate and relevant in the restricted setting we are working on. Similarly, we believe that a strict comparison between our notion of encoding and recent proposals for "good" encodings would not be fair: while it is clear that the "quality" of an encoding will always be an issue, such proposals should be taken primarily as a reference. Our interest is not in introducing a new notion of encoding but in deepening our understanding of the process-passing paradigm and its expressive power. Consequently, we feel that our results should not be judged solely on the basis of conformance to the requirements of some "good" notion of encoding.

Future Work. There are a number of directions worth investigating. An immediate issue is to explore whether the hierarchy of expressiveness for polyadic communication presented in Section 6.5 holds for a less contrained definition of encoding. Here we have focused on deriving the impossibility result based on the invariance of private names along certain computations; it remains to be explored if other approaches to the separation result, in particular those based on *experiments* and *divergence* as in testing semantics (De Nicola and Hennessy, 1984), could allow for a proof with a less constrained definition of encoding. We wish to insist that the challenge is to find a notion that enforces the same correctness guarantees as the ones we have aimed to enforce here. Clearly, more relaxed conditions in the definition of encoding would give more significance to our results. Unfortunately, up to now we have been unable to prove the separation results using a less constrained definition.

We have discussed two *dimensions* of expressiveness: a *horizontal* dimension given by the hierarchy based on polyadic communication in Section 6.5, and a *vertical* dimension that is given by the separation result based on abstraction passing in Section 6.6. The horizontal hierarchy has been obtained by identifying a distinguised form over higher-order processes with process-passing only, and by defining a number of stability conditions over such forms. While the horizontal hierarchy has been defined for any arity greater than zero, the result in Section 6.6 only provides one "level" in the vertical hierarchy, i.e. the separation between calculi without abstraction passing and calculi with only passing of abstractions of order one. (Recall that a very similar hierarchy based on abstraction passing has been obtained in (Sangiorgi, 1996b).)

We believe that an approach based on distinguished forms and stability conditions can be given so as to characterize the other levels of the vertical hierarchy. As a matter of fact, we have preliminary results in such an extended approach: we have an alternative proof for Proposition 6.8 which relies on an extension of the notion of Disjoint Form (see Definition 6.12) that represents the more complex structure (i.e. an additional level of nesting of processes) that processes with abstraction passing might exhibit. As in the case of the separation result in Section 6.5, the alternative proof for Lemma 6.8 exploits both the dependencies induced by nesting of processes in the distinguished form and the fact that private names "remain disjoint" to a certain extent. The proof we have at present requires *three* communication partners that feature *two* public synchronizations among them (one of which communicates an abstraction of level one) in order to arrive to the distinguished form for the abstraction passing case. This is in contrast to the proof of Lemma 6.7 which requires only two communication partners and a single public synchronization. Consequently, the alternative proof involves many more details and subtleties than the one of Lemma 6.6. Our current intuition is that in order to prove the separation between calculi in higher levels of the vertical hierarchy we will require a *varying* number of communication partners (and hence, of public synchronizations); the exact number should be proportional to the order of the abstractions in the calculi involved. Hence, the complexity of the separation is expected to increase as we "move up" in the hierarchy.

Chapter 7

Conclusions and Perspectives

7.1 Concluding Remarks

Expressiveness and decidability have been little studied in the context of calculi for higher-order concurrency. In this dissertation we take a direct and minimal approach to the expressiveness and decidability of higher-order process calculi. This approach finds justification in the fact that higher-order process calculi with specialized operators or modeling features often do not admit a satisfactory interpretation into some first-order setting. The results in this dissertation concern issues which can be regarded as *basic* in process calculi, namely (a)synchrony, polyadicity, forwarding. As such, our contributions might have a potential repercussion in the definition of a large class of higher-order process calculi.

A first achievement of our research is the introduction of HOCORE as a *core* calculus for higher-order concurrency. In fact, HOCORE provides a convenient framework to study fundamental issues of higher-order process calculi: it is minimal enough so as to be theoretically tractable and, at the same time, it is expressive enough so as to represent interesting phenomena in higher-order concurrency. In our opinion, HOCORE can be regarded as the simplest, non-trivial process calculus featuring higher-order concurrency.

The most salient feature of HOCORE is the *absence of name passing* in communications. Given the prominent rôle of name passing within calculi for concurrency, and the fact that most known higher-order process calculi feature both name and process passing, this could be well considered as the main design decision in our research. The absence of name passing is necessary to isolate the behavior associated to process passing; as such, it allows to obtain more accurate assessments of the expressiveness of the process-passing paradigm. In this sense, the results in this dissertation not only deepen our understanding of process-passing communication but they can also be interpreted as indirect evidence of the expressiveness and significance of the name-passing communication discipline.

It is worth observing that even in the absence of name passing, process-passing is a very expressive paradigm. This is demonstrated by the fact that HOcore, $\text{Ho}^{-\text{f}}$ (the fragment of HOcore with limited forwarding), and $\text{HoP}^{-\text{f}}$ (the extension of $\text{Ho}^{-\text{f}}$ with passivation) are all shown to be Turing complete by exhibiting encodings of Minsky machines. Remarkably, HOcore and its variants do not have operators for infinite behavior. All the encodings of Minsky machines presented in this dissertation are compact and intuitive, and exploit convenient modelling idioms —such as input-guarded replication and disjoint sum— which can be expressed succinctly with process-passing only (see Chapters 3 and 5). Another result on the expressive power of process-passing presented in the dissertation is the encoding of synchronous into asynchronous communication presented in Chapter 6 for AHO, the extension of HOcore with restriction and polyadic communication.

Closely related to the absence of name passing is the treatment of *restriction*. Restriction is a practically important construct, as it provides a way of enforcing modelling principles such as encapsulation and abstraction into specifications. Along the dissertation, the absence/presence of restriction has shown to have a notable influence on our developments. The absence of restriction makes HOcore a *public* calculus in which behavior is completely exposed; in Chapter 4 we use this to show that Input-Output bisimilarity characterizes τ actions, a necessary step in showing decidability of strong bisimilarity. Also, the absence of restriction was useful when deriving an axiomatization of strong bisimilarity, as it allowed to adapt previous results by Moller (1989), Milner and Moller (1993), and Hirschkoff and Pous (2007) for (first-order) calculi without restriction. Also in Chapter 4 we analyzed *top-level restrictions*, and showed that when HOcore is extended with four of such restrictions strong bisimilarity is no longer decidable.

The results in Chapter 5 are also insightful with respect to restriction. There, we show that in $\text{Ho}^{-\text{f}}$ termination is decidable whereas convergence is undecidable. While undecidablity of convergence is shown by exhibiting a *unfaithful* encoding of Minsky machines, we are able to prove decidability of termination by appealing the theory of well-structured transition systems. To our knowledge, ours is the first application of such theory in the higher-order setting. $\text{Ho}^{-\text{f}}$ is a calculus without restriction and yet it has the same decidability properties of $\text{CCS}_{!}^{\nu}$, the variant of CCS with restriction and replication as the only source of infinite behavior. Indeed, also in $\text{CCS}_{!}^{\nu}$ termination is decidable and convergence is undecidable (Busi et al., 2009). We find this surprising because it is well-known that in fragments of CCS *without restriction* decision problems such as termination and convergence of processes are decidable.[1] This observation on the presence of restriction also bears witness of the

[1] In fact, Goltz (1988) has shown that the fragment of CCS without restriction and relabeling can be translated into a strongly bisimilar finite Petri net. Since termination and convergence are decidable for finite Petri nets (see, e.g., (Esparza and Nielsen, 1994))—and strong bisimilarity preserves both properties—we can conclude that both termination and convergence are decidable in such a fragment of CCS.

expressiveness of process-passing. Furthermore, the expressiveness results for $\text{HoP}^{-\text{f}}$ given in Chapter 5 can be alternatively interpreted from the point of view of restriction. In fact, passivation as we define it here adds a subtle notion of structure to higher-order processes. This reminds us of the rôle of restriction in expressiveness studies for other process calculi. In that sense, passivation can be considered as a very relaxed form of restriction. It is worth noticing that the addition of passivation to $\text{Ho}^{-\text{f}}$ allows to describe a *faithful* encoding of Minsky machines, thus showing that both convergence and termination are undecidable in $\text{HoP}^{-\text{f}}$. The notion of structure on processes induced by passivation units is therefore crucial to both expressiveness and decidability of $\text{HoP}^{-\text{f}}$.

Finally, in Chapter 6 we consider extensions of HOcore with full (i.e., ordinary) restriction. There, we discussed how the scope extrusion one obtains with restriction but without name-passing is *incomplete* in that (restricted) names can be passed around inside processes but cannot be effectively used within of receiving context. As a result, with only process-passing it is not possible to establish *private links* as those used in encodings of synchronous and polyadic communication in first-order calculi. This insight is central to the separation results for SHO (i.e., the synchronous variant of AHO). By combining selected features from a more informative LTS and a rather demanding notion of encoding we were able to show that SHO^{n} (i.e. the instance of SHO with n-adic communication) cannot be encoded into SHO^{n-1}, thus determining a *hierarchy* of higher-order process calculi based on polyadicity. This result suggests that in the absence of the name-passing, the impact of adding restriction diminishes. The last result in Chapter 6 shows that the ability of establishing private links in SHO (that is, the ability of fully exploiting restriction and restricted names) is obtained when extending SHO with abstraction-passing. This is an insightful result, in that abstraction is arguably one of the most practically useful constructs in the higher-order π-calculus.

Along the dissertation we describe the way in which basic modeling idioms such as lists, counters, and constructs for choice and guarded infinite behavior can be expressed in core higher-order process calculi. Nevertheless, this does not seem enough so as to consider core higher-order process calculi as adequate as *modelling languages* in concrete application areas. In our opinion, higher-order process calculi for specialized application areas should arise from the careful combination of higher-order constructs and name-passing features.

7.2 Ongoing and Future Work

Along the dissertation we have already pointed out a number of strands for future work. We conclude by commenting on those directions we find particularly promising; some of them are object of current work.

More on Expressiveness of Passivation. In Chapter 5 we have examined the expressiveness associated to suspension operators by studying the influence a passivation operator has on the absolute expressive power of a higher-order process calculus with restricted output actions. In this respect, much remains to be done. In fact, we have considered a particular form of passivation, one that allows to both suspend a process and then restart it later. Other forms of passivation (more precisely, other semantics for passivation) are also possible and could be relevant. A natural concern is that, as we pointed out before, passivation as defined here has a very non-deterministic character. It is reasonable to assume that the kind of passivation required for applications in dynamic system reconfiguration to be more controlled.

We have conducted preliminary studies on the expressiveness of passivation in the context of $\mathrm{aCCS}_!^{-\nu}$, the asynchronous fragment of CCS without restriction and with replication (Di Giusto et al., 2009b). In the absence of process-passing a passivation action entails essentially the *destruction* of the content of the passivation unit. That is, suspension represents a “kill” action over a process. We think that this “destructive passivation” is perhaps the simplest kind of passivation one could think of. In (Di Giusto et al., 2009b) we show that even destructive passivation is enough to increase the expressive power of $\mathrm{aCCS}_!^{-\nu}$. Using the same theoretical machinery as in Chapter 5 (i.e. unfaithful encodings of Minsky machines and well-structured transition systems) we show that in $\mathrm{aCCS}_!^{-\nu}$ extended with passivation: (i) in contrast to the situation in $\mathrm{aCCS}_!^{-\nu}$, convergence is undecidable; and (ii) similarly as in $\mathrm{aCCS}_!^{-\nu}$, termination is decidable.

Higher-Order and Ambient-like Calculi. Ambient-like and higher-order process calculi are similar in that they involve complex objects in interactions. Also, in both cases the associated behavioral theory can be hard to define, and employs similar techniques. However, some other characteristics suggest deep differences. Communication in Ambients resembles a “move” operation whereas in higher-order settings it is better assimilated to a “copy” operation. Most notably, there is a subtle discrepancy when it comes to *binding*: most (higher-order) process calculi adopt static binding only, whereas Ambient-like formalisms exhibit features of both static and dynamic binding.

Based on the above, we find it interesting to formally compare Ambient-like and higher-order calculi. From the point of view of *expressiveness*, this is relevant for at least two reasons. First, in the light of the above differences, an encoding of Ambient calculi would represent a significant test of expressiveness for the process-passing paradigm. Second, it would shed light on the intrinsic nature of the Ambient primitives which have proven so successful.

In (Di Giusto and Pérez, 2009) we have reported on initial results of our investigation: an encoding of Ambient calculus into a higher-order process calculi with localities (implemented as passivation units) and a form of dynamic binding. The encoding is useful to understand the nature of Ambient communication; it also allows us to conjecture that an encoding into a higher-order process calculus with static binding does not exist. It would be interesting to see whether this encodability result can be exploited/strengthened so as to have a more conclusive assessment of the expressiveness of the higher-order paradigm with respect to the Ambient calculi. Also, it is not clear how to proceed in order to transform our conjecture into a formal separation result. It could be that passivation and dynamic binding —crucial in the encoding in (Di Giusto and Pérez, 2009)— could give hints in this case, but this remains to be explored in detail.

Dimensions of Mobility. Together with Roland Meyer, we are studying the relationship between decidability results for the π-calculus and those presented in Chapter 5 for $\mathsf{Ho}^{-\mathsf{f}}$. In his PhD Thesis, Meyer (2008) studied the notion of *structural stationarity* in the π-calculus. Roughly speaking, structural stationarity means bounding processes so as to obtain decidability results and hence perform automatic verification techniques on them. In the π-calculus, structural stationarity arises by giving bounds on two *dimensions* of infinite behavior: *depth* (i.e., the nesting of restrictions inside a process) and *breadth* (i.e., the degree of parallelism of a process). It would be interesting to determine precisely what structural stationarity means for higher-order processes, and its exact relationship with that in the first-order setting. In principle, such a relationship should allow to transfer and generalize decidability results from one setting to the other.

References

Abadi, M. and Fournet, C. 2001. Mobile values, new names, and secure communication. In *Proc. of POPL'01*. ACM, 104–115. 43

Abadi, M. and Gordon, A. D. 1999. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput. 148,* 1, 1–70. 34, 43

Abdulla, P. A., Cerans, K., Jonsson, B., and Tsay, Y.-K. 2000. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput. 160,* 1-2, 109–127. 50, 93, 124

Abramsky, S. 1989. The lazy lambda calculus. In *Research Topics in Functional Programming*, D. Turner, Ed. Addison Wesley, Reading, MA., 65–116. 31, 34

Amadio, R. M. 1993. On the reduction of chocs bisimulation to pi-calculus bisimulation. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 715. Springer, 112–126. Extended version available in the author's web site. 52, 132

Amadio, R. M. 1994. Translating core facile. Tech. Rep. ECRC-94-3, ECRC, Munich. 52

Amadio, R. M., Castellani, I., and Sangiorgi, D. 1998. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci. 195,* 2, 291–324. 74

Amadio, R. M., Leth, L., and Thomsen, B. 1995. From a concurrent lambda-calculus to the pi-calculus. In *Proc. of FCT*. Lecture Notes in Computer Science, vol. 965. Springer, 106–115. 52

Aranda, J., Giusto, C. D., Nielsen, M., and Valencia, F. D. 2007. Ccs with replication in the chomsky hierarchy: The expressive power of divergence. In *Proc. of APLAS*. Lecture Notes in Computer Science, vol. 4807. Springer, 383–398. 48

Aranda, J. A. 2009. On the expressivity of infinite and local behaviour in fragments of the pi-calculus. Ph.D. thesis, École Polytechnique de Paris and Universidad del Valle Colombia. 39, 49

Astesiano, E., Giovini, A., and Reggio, G. 1988. Generalized bisimulation in relational specifications. In *Proc. of STACS*. Lecture Notes in Computer Science, vol. 294. Springer, 207–226. 35

Baldamus, M. 1998. Semantics and logic of higher-order processes: Characterizing late context bisimulation. Ph.D. thesis, Computer science department, Berlin University of Technology. 36

Barendregt, H. 1984. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland. 51

Beauxis, R., Palamidessi, C., and Valencia, F. D. 2008. On the asynchronous nature of the asynchronous pi-calculus. In *Concurrency, Graphs and Models*. Lecture Notes in Computer Science, vol. 5065. Springer, 473–492. 12, 46, 131

Bernstein, K. L. 1998. A congruence theorem for structured operational semantics of higher-order languages. In *Proc. of LICS'98*. IEEE Computer Society, 153–164. 37

Bloom, B. 1994. Chocolate: Calculi of higher order communication and lambda terms. In *Proc. of POPL'94*. ACM, 339–347. 31

Boreale, M., De Nicola, R., and Pugliese, R. 1999. Basic observables for processes. *Inf. Comput. 149,* 1, 77–98. 33

Boudol, G. 1989. Towards a lambda-calculus for concurrent and communicating systems. In *Proc. of TAPSOFT, Vol.1*. Lecture Notes in Computer Science, vol. 351. Springer, 149–161. Also appeared as INRIA Research Report No. RR-0885, August 1988. 5, 29, 30, 31, 35

Boudol, G. 1992. Asynchrony and the π-calculus (note). Tech. rep., Rapport de Recherche 1702, INRIA, Sophia-Antipolis. 11, 42, 128

Boudol, G. 1998. The pi-calculus in direct style. *Higher-Order and Symbolic Computation 11,* 2, 177–208. A preliminary version appeared in Proc. of POPL'97. 31, 32

Bougé, L. 1988. On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. *Acta Inf. 25,* 2, 179–201. 42, 50

Bravetti, M. and Zavattaro, G. 2009. On the expressive power of process interruption and compensation. *Math. Struct. in Comp. Sci. 19,* 3, 565–599. 48, 92, 124

Bundgaard, M., Glenstrup, A. J., Hildebrandt, T. T., Højsgaard, E., and Niss, H. 2008. Formalizing higher-order mobile embedded business processes with binding bigraphs. In *Proc. of COORDINATION*. Lecture Notes in Computer Science, vol. 5052. Springer, 83–99. 118

Bundgaard, M., Godskesen, J. C., Haagensen, B., and Hüttel, H. 2009. Decidable fragments of a higher order calculus with locations. *Electr. Notes Theor. Comput. Sci. 242,* 1, 113–138. 10, 54, 125

Bundgaard, M., Hildebrandt, T. T., and Godskesen, J. C. 2006. A cps encoding of name-passing in higher-order mobile embedded resources. *Theor. Comput. Sci. 356,* 3, 422–439. 10, 54

Busi, N., Gabbrielli, M., and Zavattaro, G. 2003. Replication vs. recursive definitions in channel based calculi. In *Proc. of ICALP*. Lecture Notes in Computer Science, vol. 2719. Springer, 133–144. 64

Busi, N., Gabbrielli, M., and Zavattaro, G. 2009. On the expressive power of recursion, replication and iteration in process calculi. *Math. Struct. in Comp. Sci. 19,* 6, 1191–1222. 18, 47, 49, 50, 64, 93, 95, 124, 156

Busi, N., Gorrieri, R., and Zavattaro, G. 2000. On the expressiveness of linda coordination primitives. *Inf. Comput. 156,* 1-2, 90–121. Full version of a paper in Proc. of EXPRESS'97. 47, 50

Busi, N. and Zandron, C. 2009. Computational expressiveness of genetic systems. *Theor. Comput. Sci. 410,* 4-5, 286–293. 39

Busi, N. and Zavattaro, G. 2000. On the expressiveness of event notification in data-driven coordination languages. In *Proc. of ESOP*. Lecture Notes in Computer Science, vol. 1782. Springer, 41–55. 47

Busi, N. and Zavattaro, G. 2004. On the expressive power of movement and restriction in pure mobile ambients. *Theor. Comput. Sci. 322,* 3, 477–515. 47, 64, 125

Cacciagrano, D., Corradini, F., and Palamidessi, C. 2007. Separation of synchronous and asynchronous communication via testing. *Theor. Comput. Sci. 386,* 3, 218–235. 12, 43, 131

Cao, Z. 2006. More on bisimulations for higher order *pi*-calculus. In *Proc. of FoSSaCS'06*. Lecture Notes in Computer Science, vol. 3921. Springer, 63–78. 35, 74

Carbone, M. and Maffeis, S. 2003. On the expressive power of polyadic synchronisation in pi-calculus. *Nord. J. Comput. 10,* 2, 70–98. 51

Cardelli, L. and Gordon, A. D. 2000. Mobile ambients. *Theor. Comput. Sci. 240,* 1, 177–213. A preliminary version appeared in Proc. of FOSSACS'98. 5, 7, 32, 47, 125

Chandra, A. K. and Manna, Z. 1976. On the power of programming features. *Comput. Lang. 1,* 3, 219–232. 40

CHRISTENSEN, S. 1993. PhD thesis CST–105–93. Ph.D. thesis, Dept. of Computer Science, University of Edinburgh. 47

CHRISTENSEN, S., HIRSHFELD, Y., AND MOLLER, F. 1994. Decidable subsets of CCS. *Comput. J. 37,* 4, 233–242. 89

COLLBERG, C. S., THOMBORSON, C. D., AND LOW, D. 1998. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Proc. of POPL'98*. ACM Press, 184–196. 93

DE BOER, F. S. AND PALAMIDESSI, C. 1990. Concurrent logic programming: Asynchronism and language comparison. In *Proc. of NACLP*. The MIT Press, Series in Logic Programming, 175–194. 41

DE BOER, F. S. AND PALAMIDESSI, C. 1991. Embedding as a tool for language comparison: On the csp hierarchy. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 527. Springer, 127–141. 42

DE BOER, F. S. AND PALAMIDESSI, C. 1994. Embedding as a tool for language comparison. *Inf. Comput. 108,* 1, 128–157. 41, 42

DE BRUIJN, N. G. 1972. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae 34*, 381–392. 82

DE NICOLA, R. 2006. From process calculi to Klaim and back. *Electr. Notes Theor. Comput. Sci. 162*, 159–162. 33

DE NICOLA, R., FERRARI, G. L., AND PUGLIESE, R. 1998. Klaim: A kernel language for agents interaction and mobility. *IEEE Trans. Software Eng. 24,* 5, 315–330. 33

DE NICOLA, R. AND HENNESSY, M. 1984. Testing equivalences for processes. *Theor. Comput. Sci. 34*, 83–133. 15, 46, 153

DE SIMONE, R. 1985. Higher-level synchronising devices in meije-sccs. *Theor. Comput. Sci. 37*, 245–267. 39, 52

DEMANGEON, R., HIRSCHKOFF, D., AND SANGIORGI, D. 2010. Termination in higher-order concurrent calculi. In *Proc. of FSEN'09*. Lecture Notes in Computer Science, vol. 5961. Springer, 81–96. 11

DI GIUSTO, C. AND PÉREZ, J. A. 2009. Move vs copy: Towards a formal comparision of ambients and higher-order process calculi. In *Proc. of ICTCS'09: the Eleventh Italian Conference on Theoretical Computer Science*. 158, 159

Di Giusto, C., Pérez, J. A., and Zavattaro, G. 2009a. On the expressiveness of forwarding in higher-order communication. In *Proc. of ICTAC*. Lecture Notes in Computer Science, vol. 5684. Springer, 155–169. 14, 91

Di Giusto, C., Pérez, J. A., and Zavattaro, G. 2009b. On the expressiveness of suspension in higher-order process calculi. In preparation. 158

Dovier, A., Piazza, C., and Policriti, A. 2004. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci. 311,* 1-3, 221–256. 81

Dsouza, A. and Bloom, B. 1995. On the expressive power of ccs. In *Proc. of FSTTCS*. Lecture Notes in Computer Science, vol. 1026. Springer, 309–323. 52

Dufourd, C., Finkel, A., and Schnoebelen, P. 1998. Reset nets between decidability and undecidability. In *Proc. of ICALP*. Lecture Notes in Computer Science, vol. 1443. Springer, 103–115. 50

Ene, C. and Muntean, T. 1999. Expressiveness of point-to-point versus broadcast communications. In *Proc. of FCT*. Lecture Notes in Computer Science, vol. 1684. Springer, 258–268. 43, 50

Esparza, J. and Nielsen, M. 1994. Decidability issues for petri nets - a survey. *Bulletin of the EATCS 52*, 244–262. 156

Felleisen, M. 1991. On the expressive power of programming languages. *Sci. Comput. Program. 17,* 1-3, 35–75. A preliminary version appeared in Proc. of ESOP'90. 40, 46

Ferreira, W., Hennessy, M., and Jeffrey, A. 1998. A theory of weak bisimulation for core cml. *J. Funct. Program. 8,* 5, 447–491. 36

Finkel, A. 1990. Reduction and covering of infinite reachability trees. *Inf. Comput. 89,* 2, 144–179. 50, 93, 124

Finkel, A. and Schnoebelen, P. 2001. Well-structured transition systems everywhere! *Theor. Comput. Sci. 256,* 1-2, 63–92. 50, 93, 105, 106, 124

Fournet, C., Gonthier, G., Lévy, J.-J., Maranget, L., and Rémy, D. 1996. A calculus of mobile agents. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 1119. Springer, 406–421. 32

Giacalone, A., Mishra, P., and Prasad, S. 1989. Facile: A symmetric integration of concurrent and functional programming. In *Proc. of TAPSOFT, Vol.2*. Lecture Notes in Computer Science, vol. 352. Springer, 184–209. 30

Godskesen, J. C. and Hildebrandt, T. T. 2005. Extending howe's method to early bisimulations for typed mobile embedded resources with local names. In *Proc. of FSTTCS*. Lecture Notes in Computer Science, vol. 3821. Springer, 140–151. 36

Goltz, U. 1988. On representing ccs programs by finite petri nets. In *Proc. of MFCS*. Lecture Notes in Computer Science, vol. 324. Springer, 339–350. 156

Gorla, D. 2006. Comparing calculi for mobility via their relative expressive power. Tech. Rep. 09/2006, Dipartimento di Informatica, Universita di Roma - La Sapienza. 46

Gorla, D. 2008. Towards a unified approach to encodability and separation results for process calculi. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 5201. Springer, 492–507. Extended version available as Tech. Rep. 10/2008, Dip. Informatica, Universita di Roma - La Sapienza. 43, 46, 135

Hennessy, M., Rathke, J., and Yoshida, N. 2005. safedpi: a language for controlling mobile code. *Acta Inf. 42,* 4-5, 227–290. 34

Higman, G. 1952. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3) 2,* 7, 326–336. 106

Hildebrandt, T., Godskesen, J. C., and Bundgaard, M. 2004. Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Tech. Rep. TR-2004-52, IT University of Copenhagen. 9, 10, 29, 33, 36, 118, 125

Hirschkoff, D., Lozes, E., and Sangiorgi, D. 2002. Separability, expressiveness, and decidability in the ambient logic. In *Proc. of LICS'02*. IEEE Computer Society Press, 423–432. 47

Hirschkoff, D. and Pous, D. 2007. A distribution law for CCS and a new congruence result for the *pi*-calculus. In *Proc. of FoSSaCS'07*. Lecture Notes in Computer Science, vol. 4423. Springer, 228–242. 78, 79, 156

Honda, K. and Tokoro, M. 1991. An object calculus for asynchronous communication. In *Proc. of ECOOP*. Lecture Notes in Computer Science, vol. 512. Springer, 133–147. 11, 42, 128

Honda, K., Vasconcelos, V. T., and Kubo, M. 1998. Language primitives and type discipline for structured communication-based programming. In *Proc. of ESOP*. Lecture Notes in Computer Science, vol. 1381. Springer, 122–138. 4, 43

Honda, K. and Yoshida, N. 1994a. Combinatory representation of mobile processes. In *Proc. of POPL'94*. ACM, 348–360. 51

Honda, K. and Yoshida, N. 1994b. Replication in concurrent combinators. In *Proc. of TACS*. Lecture Notes in Computer Science, vol. 789. Springer, 786–805. 51

Honda, K. and Yoshida, N. 1995. On reduction-based process semantics. *Theor. Comput. Sci. 151,* 2, 437–486. 74

Howe, D. J. 1996. Proving congruence of bisimulation in functional programming languages. *Inf. Comput. 124,* 2, 103–112. 36, 70

Jeffrey, A. and Rathke, J. 2005. Contextual equivalence for higher-order pi-calculus revisited. *Logical Methods in Computer Science 1,* 1, 1–22. 35, 36, 67

Koutavas, V. and Hennessy, M. 2009. First-order reasoning for higher-order concurrency. Tech. rep., Trinity College Dublin. July. 36

Kucera, A. and Jancar, P. 2006. Equivalence-checking on infinite-state systems: Techniques and results. *TPLP 6,* 3, 227–264. 89, 90

Landin, P. J. 1966. The Next 700 Programming Languages. *Communications of the ACM 9,* 3 (March), 157–166. 40

Lanese, I. 2007. Concurrent and located synchronizations in *pi*-calculus. In *Proc. of SOFSEM*. Lecture Notes in Computer Science, vol. 4362. Springer, 388–399. 131

Lanese, I., Pérez, J. A., Sangiorgi, D., and Schmitt, A. 2008. On the expressiveness and decidability of higher-order process calculi. In *Proc. of LICS'08*. IEEE Computer Society, 145–155. 14

Lanese, I., Pérez, J. A., Sangiorgi, D., and Schmitt, A. 2009. On the expressiveness and decidability of polyadicity in higher-order process calculi (extended abstract). In *Proc. of ICTCS'09: the Eleventh Italian Conference on Theoretical Computer Science*. 14, 127

Laneve, C. and Victor, B. 2003. Solos in concert. *Mathematical Structures in Computer Science 13,* 5, 657–683. 51

Lenglet, S., Schmitt, A., and Stefani, J.-B. 2008. Bisimulations in calculi featuring passivation and restriction. Technical Report, Sardes Project, INRIA Rhône Alpes. 11, 37

Lenglet, S., Schmitt, A., and Stefani, J.-B. 2009a. Howe's method for calculi with passivation. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 5710. Springer, 448–462. 37

Lenglet, S., Schmitt, A., and Stefani, J.-B. 2009b. Normal bisimulations in calculi with passivation. In *Proc. of FOSSACS*. Lecture Notes in Computer Science, vol. 5504. Springer, 257–271. 37

Maffeis, S., Abadi, M., Fournet, C., and Gordon, A. D. 2008. Code-carrying authorization. In *Proc. of ESORICS*. Lecture Notes in Computer Science, vol. 5283. Springer, 563–579. 34

Maffeis, S. and Phillips, I. 2005. On the computational strength of pure ambient calculi. *Theor. Comput. Sci. 330,* 3, 501–551. 47, 48

Mayr, R. 2000. Process rewrite systems. *Inf. Comput. 156,* 1-2, 264–286. 48

Meredith, L. G. and Radestock, M. 2005a. Namespace logic: A logic for a reflective higher-order calculus. In *Proc. of TGC*. Lecture Notes in Computer Science, vol. 3705. Springer, 353–369. 34

Meredith, L. G. and Radestock, M. 2005b. A reflective higher-order calculus. *Electr. Notes Theor. Comput. Sci. 141,* 5, 49–67. 34

Meyer, R. 2008. Structural stationarity in the pi-calculus. Ph.D. thesis, Department of Computing Science, University of Oldenburg. 159

Milner, R. 1989. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall. 15, 17, 19, 25, 67

Milner, R. 1991. The Polyadic pi-Calculus: A Tutorial. Tech. Rep. ECS-LFCS-91-180, University of Edinburgh. 23, 42, 128

Milner, R. 1992. Functions as processes. *Mathematical Structures in Computer Science 2,* 2, 119–141. 4, 52

Milner, R. and Moller, F. 1993. Unique decomposition of processes. *Theor. Comput. Sci. 107,* 2, 357–363. 78, 79, 156

Milner, R., Parrow, J., and Walker, D. 1992. A calculus of mobile processes, i. *Inf. Comput. 100,* 1, 1–40. A preliminary version appeared as Technical Report ECS-LFCS-89-85, LFCS, University of Edinburgh, June 1989. 4, 15, 22, 31

Milner, R. and Sangiorgi, D. 1992. Barbed bisimulation. In *Proc. 19th ICALP*, W. Kuich, Ed. Lecture Notes in Computer Science, vol. 623. Springer Verlag, 685–695. 20, 46, 74

Minsky, M. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall. 47, 49

Mitchell, J. C. 1993. On abstraction and the expressive power of programming languages. *Sci. Comput. Program. 21,* 2, 141–163. 40

Moller, F. 1989. Axioms for concurrency. Ph.D. thesis, University of Edinburgh, Dept. of Comp. Sci. PhD thesis CST–59–89. 78, 156

MOLLER, F. 1996. Infinite results. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 1119. Springer, 195–216. 48

MOSTROUS, D. AND YOSHIDA, N. 2007. Two session typing systems for higher-order mobile processes. In *Proc. of TLCA*. Lecture Notes in Computer Science, vol. 4583. Springer, 321–335. 34

MOSTROUS, D. AND YOSHIDA, N. 2009. Session-based communication optimisation for higher-order mobile processes. In *Proc. of TLCA*. Lecture Notes in Computer Science, vol. 5608. Springer, 203–218. 34

MOUSAVI, M. R., GABBAY, M., AND RENIERS, M. A. 2005. SOS for Higher Order Processes. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 3653. Springer, 308–322. 37

MOUSAVI, M. R., RENIERS, M. A., AND GROOTE, J. F. 2007. SOS formats and meta-theory: 20 years after. *Theor. Comput. Sci. 373,* 3, 238–272. 37

NECULA, G. C. AND LEE, P. 1998. Safe, untrusted agents using proof-carrying code. In *Mobile Agents and Security*. Lecture Notes in Computer Science, vol. 1419. Springer, 61–91. 93

NESTMANN, U. 1996. On determinacy and and nondeterminacy in concurrent programming. Ph.D. thesis, Univ. Erlangen. 43, 44

NESTMANN, U. 2000. What is a "good" encoding of guarded choice? *Inf. Comput. 156,* 1-2, 287–319. A preliminary version appeared in EXPRESS'97. 43, 46

NESTMANN, U. AND PIERCE, B. C. 2000. Decoding choice encodings. *Inf. Comput. 163,* 1, 1–59. Extended abstract in Proc. of CONCUR'96. 43, 45

NIELSON, F. 1989. The typed lambda-calculus with first-class processes. In *Proc. of PARLE (2)*. Lecture Notes in Computer Science, vol. 366. Springer, 357–373. 5, 29, 30

NYGAARD, M. AND WINSKEL, G. 2002. Hopla-a higher-order process language. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 2421. Springer, 434–448. 33

OSTROVSKY, K., PRASAD, K. V. S., AND TAHA, W. 2002. Towards a primitive higher order calculus of broadcasting systems. In *Proc. of PPDP*. ACM, 2–13. 34

PALAMIDESSI, C. 2003. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science 13,* 5, 685–719. Extended abstract in Proc. of POPL'97. 12, 43, 45, 46, 50, 128, 131

PARROW, J. 1990. The expressive power of parallelism. *Future Gener. Comput. Syst. 6,* 3, 271–285. 51

Parrow, J. 2000. Trios in concert. In *Proof, Language, and Interaction*. The MIT Press, 623–638. 51

Parrow, J. 2008. Expressiveness of process algebras. *Electr. Notes Theor. Comput. Sci. 209*, 173–186. 37, 39

Pierce, B. and Sangiorgi, D. 1996. Typing and subtyping for mobile processes. *Journal of Mathematical Structures in Computer Science 6,* 5, 409–454. An extended abstract appeared in *Proc. LICS 93*, IEEE Computer Society Press. 23

Post, E. L. 1946. A variant of a recursively unsolvable problem. *Bull. of the Am. Math. Soc 52*, 264–268. 85

Prasad, S., Giacalone, A., and Mishra, P. 1990. Operational and algebraic semantics for facile: A symmetric integration of concurrent and functional programming. In *Proc. of ICALP*. Lecture Notes in Computer Science, vol. 443. Springer, 765–778. 30

Priami, C. 1995. Stochastic pi-calculus. *Comput. J. 38,* 7, 578–589. 42, 43

Quaglia, P. and Walker, D. 2005. Types and full abstraction for polyadic pi-calculus. *Inf. Comput. 200,* 2, 215–246. 43, 131

Radestock, M. and Eisenbach, S. 1996. Semantics of a higher-order coordination language. In *Proc. of COORDINATION*. Lecture Notes in Computer Science, vol. 1061. Springer, 339–356. 34

Raja, N. and Shyamasundar, R. K. 1995a. Combinatory formulations of concurrent languages. In *Proc. of ASIAN*. Lecture Notes in Computer Science, vol. 1023. Springer, 156–170. 51

Raja, N. and Shyamasundar, R. K. 1995b. The quine-bernays combinatory calculus. *Int. J. Found. Comput. Sci. 6,* 4, 417–430. 51

Reppy, J. H. 1991. CML: A higher-order concurrent language. In *PLDI*. 293–305. 30

Reppy, J. H. 1992. Higher-order concurrency. Ph.D. thesis, Cornell University. 30

Riecke, J. G. 1993. Fully abstract translations between functional languages. *Mathematical Structures in Computer Science 3,* 4, 387–415. A preliminary report appeared in Proc. of POPL'91. 41

Sangiorgi, D. 1992. Expressing mobility in process algebras: First-order and higher-order paradigms. Ph.D. thesis, University of Edinburgh. 4, 5, 6, 10, 21, 23, 25, 26, 28, 29, 35, 42, 43, 47, 52, 53, 55, 67, 84

Sangiorgi, D. 1993. From π-calculus to Higher-Order π-calculus — and back. In *Proc. TAPSOFT'93*, M.-C. Gaudel and J.-P. Jouannaud, Eds. Lecture Notes in Computer Science, vol. 668. Springer Verlag, 151–166. 21, 27

Sangiorgi, D. 1994. The lazy lambda calculus in a concurrency scenario. *Information and Computation 111,* 1, 120–153. 34

Sangiorgi, D. 1996a. Bisimulation for Higher-Order Process Calculi. *Inf. Comput. 131,* 2, 141–178. 35, 36, 55, 84

Sangiorgi, D. 1996b. π-calculus, internal mobility and agent-passing calculi. *Theor. Comput. Sci. 167,* 2, 235–274. 10, 26, 52, 55, 88, 131, 132, 151, 154

Sangiorgi, D. 1996c. A theory of bisimulation for the π-calculus. *Acta Informatica 33*, 69–97. An extract appeared in *Proc. CONCUR '93*, Lecture Notes in Computer Science 715, Springer Verlag. 35

Sangiorgi, D. 1998. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science 8*, 447–479. 36

Sangiorgi, D. 2001. Asynchronous process calculi: the first-order and higher-order paradigms (tutorial). *Theor. Comput. Sci. 253*, 311–350. 27

Sangiorgi, D. 2009. *An introduction to bisimulation and coinduction*. Draft. 16, 19

Sangiorgi, D., Kobayashi, N., and Sumii, E. 2007. Environmental bisimulations for higher-order languages. In *Proc. of LICS'07*. IEEE Computer Society, 293–302. 36, 67

Sangiorgi, D. and Walker, D. 2001. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press. 4, 8, 10, 21, 22, 25, 27, 53, 64, 74, 84, 88

Sato, N. and Sumii, E. 2009. The higher-order, call-by-value applied pi-calculus. In *Proc. of APLAS'09: the Seventh Asian Symposium on Programming Languages and Systems*. Lecture Notes in Computer Science. Springer. To Appear. 11, 34, 36

Schmitt, A. and Stefani, J.-B. 2002. The m-calculus: a higher-order distributed process calculus. Tech. Rep. 4361, INRIA. Jan. 32

Schmitt, A. and Stefani, J.-B. 2003. The m-calculus: a higher-order distributed process calculus. In *Proc. of POPL'03*. ACM, 50–61. 32

Schmitt, A. and Stefani, J.-B. 2004. The kell calculus: A family of higher-order distributed process calculi. In *Proc. of Global Computing*. Lecture Notes in Computer Science, vol. 3267. Springer, 146–178. 9, 29, 32, 118

SHAPIRO, E. Y. 1989. The family of concurrent logic programming languages. *ACM Comput. Surv. 21,* 3, 413–510. 41

SHAPIRO, E. Y. 1991. Separating concurrent languages with categories of language embeddings (extended abstract). In *Proc. of STOC'91*. ACM, 198–208. 41

SHAPIRO, E. Y. 1992. Embeddings among concurrent programming languages (preliminary version). In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 630. Springer, 486–503. 42

SHEPHERDSON, J. C. AND STURGIS, H. E. 1963. Computability of recursive functions. *J. ACM 10,* 2, 217–255. 47

SIPSER, M. 2005. *Introduction to the Theory of Computation*. PWS Publishing Company. 85

THOMSEN, B. 1989. A calculus of higher order communicating systems. In *Proc. of POPL'89*. ACM Press, 143–154. 5, 25, 36, 55, 172

THOMSEN, B. 1990. Calculi for higher order communicating systems. Ph.D. thesis, Dept. of Comp. Sci., Imperial College. 10, 25, 29, 30, 31, 35, 42, 54, 84

THOMSEN, B. 1993. Plain CHOCS: A second generation calculus for higher order processes. *Acta Inf. 30,* 1, 1–59. 5, 31, 36, 55

THOMSEN, B. 1995. A theory of higher order communicating systems. *Inf. Comput. 116,* 1, 38–57. Extended version of Thomsen (1989). 30

VAANDRAGER, F. W. 1992. Expressive results for process algebras. In *Proc. of REX Workshop on 'Semantics: Foundations and Application'*. Lecture Notes in Computer Science, vol. 666. Springer, 609–638. Also available as CWI Report CS-R9301, 1993. 52

VERSARI, C., BUSI, N., AND GORRIERI, R. 2009. An expressiveness study of priority in process calculi. *Math. Struct. in Comp. Sci. 19,* 6, 1161–1189. 51

VIGLIOTTI, M. G. 2004. Reduction semantics for ambient calculi. Ph.D. thesis, Imperial College London. 50

VIGLIOTTI, M. G., PHILLIPS, I., AND PALAMIDESSI, C. 2007. Tutorial on separation results in process calculi via leader election problems. *Theor. Comput. Sci. 388,* 1-3, 267–289. 50

VIVAS, J.-L. 2001. Dynamic Binding of Names in Calculi for Mobile Processes. Ph.D. thesis, KTH - Royal Instituye of Technology. 53

VIVAS, J.-L. AND DAM, M. 1998. From higher-order pi-calculus to pi-calculus in the presence of static operators. In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 1466. Springer, 115–130. 8, 53

Vivas, J.-L. and Yoshida, N. 2002. Dynamic channel screening in the higher order pi-calculus. *Electr. Notes Theor. Comput. Sci. 66,* 3. Extended version available as Technical Report 2002-22, MCS, University of Leicester. 8, 53

Walker, D. 1995. Objects in the pi-calculus. *Inf. Comput. 116,* 2, 253–271. 4

Winskel, G. and Zappa Nardelli, F. 2004. New-hopla: A higher-order process language with name generation. In *Proc. of IFIP TCS*. Kluwer, 521–534. 33

Xu, X. 2007. On the bisimulation theory and axiomatization of higher-order process calculi. Ph.D. thesis, Shanghai Jiao Tong University. 10

Yoshida, N. 1996. Graph types for monadic mobile processes. In *Proc. of FSTTCS*. Lecture Notes in Computer Science, vol. 1180. Springer, 371–386. 43, 131

Yoshida, N. 2002. Minimality and separation results on asynchronous mobile processes - representability theorems by concurrent combinators. *Theor. Comput. Sci. 274,* 1-2, 231–276. 39, 51

Yoshida, N. and Hennessy, M. 1999. Suptyping and locality in distributed higher order processes (extended abstract). In *Proc. of CONCUR*. Lecture Notes in Computer Science, vol. 1664. Springer, 557–572. 53

Zavattaro, G. 2009. Personal communication. 49