The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes

Björn Victor

A Dissertation submitted
for the Degree of Doctor of Philosophy
Department of Computer Systems
Uppsala University

June 1998

DoCS 98/98 ISSN 0283-0574

Dissertation for the Degree of Doctor of Philosophy in Computer Systems presented at Uppsala University in 1998.

ABSTRACT

Victor, B. 1998: The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. *DoCS* 98/98. 158 pp. Uppsala. ISSN 0283-0574.

The fusion calculus is presented as a significant step towards a canonical calculus of concurrency. It simplifies and extends the π-calculus of Milner, Parrow and Walker.

The fusion calculus contains the polyadic π-calculus as a proper subcalculus and thus inherits all its expressive power. In addition fusion contains actions akin to updating a shared state, and a scoping construct for bounding their effects. Therefore it is easier to represent computational models with shared state, including concurrent constraint formalisms. It is also easy to represent the so called strong reduction strategies in the λ-calculus, involving reduction under abstraction. In the π-calculus these tasks require elaborate encodings.

The fusion calculus simplifies the π-calculus by reducing the number of binding operators and the number of bisimulation equivalences, and by making input and output symmetric like in pure CCS. We attain a calculus where concepts from other models of computation are more easily expressed than in the π-calculus, thereby taking a step towards a unified yet simple model of computation.

In this thesis we present a broad foundational theory of the fusion calculus. We define its labelled and unlabelled operational semantics, and treat strong and weak bisimulation equivalences for both semantics in some detail, including complete axiom systems for finite terms. The equivalences are given symbolic characterizations, leading to algorithms and an automatic tool for equivalence checking. We demonstrate the expressive power of the fusion calculus to give simple encodings of foundational calculi for functional and concurrent constraint programming.

Björn Victor, Department of Computer Systems, Uppsala University, Box 325, S-751 05 Uppsala, Sweden, E-mail: `Bjorn.Victor@DoCS.UU.SE`

ISSN 0283-0574

Printed in Sweden by Reprocentralen HSC, Uppsala 1998.
Distributor: Department of Computer Systems, Uppsala University, Box 325, S-751 05 Uppsala, Sweden

To Ellen and Helena

Acknowledgements

My advisor Joachim Parrow has helped me through many dark passages along the way to this thesis; always encouraging and helpful, and full of ideas for improvements. His patience and generosity has allowed me to grow as a researcher, and our cooperation in writing papers has been excellent. His thorough reviewing of this thesis has been invaluable.

My friend Faron Moller has also been supportive and has helped me stay on the track. Apart from being great company for having fun, his scientific strictness and high standards has been an example to me, although hard to live up to. I am very happy to have him as a colleague and friend, as a co-author of my first paper and language expert for most of them.

Davide Sangiorgi once invited me to visit the Laboratory for Foundations of Computer Science in Edinburgh, which was a most inspiring and rewarding experience. During that visit, I worked with Faron and Davide on the initial development of the Mobility Workbench, a tool which has survived all the way to this thesis. Davide has always been encouraging, and has also provided much comment and feedback on the manuscript of this thesis.

Another experience crucial for my development was a visit to the Programming Systems Lab in Saarbrücken, where I had long inspiring discussions about mobile processes and constraints, in particular with Martin Müller. This visit provided the inspiration for the later development of the fusion calculus.

Lars-åke Fredlund, Parosh Abdulla and Bengt Jonsson deserve special thanks for reading and commenting on late versions of the manuscript. I hope to pay Lars-åke back the same way later…

The Department of Computer Systems, DoCS, has been a great place to spend so many years at, as a student, system manager and graduate student. The special "DoCS atmosphere" and the blend of experimental and theoretical research has been an important factor to help me complete my graduate studies. I have so many friends here!

Finally I must express my deepest gratitude to my wife Helena, whose patience and support even exceeds Joachim's. Her love has kept me up, even though these last months have been harder for her than for me. I must not forget our daughter Ellen (how could I?), who has not complained too much about my writing a book in English, but who has really missed me. Now I'm back!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The *art* of computer programming, as it has been practised since the first stored computer program was executed 50 years ago, on the 21st of June 1948 [WK48], is a magic craft of controlling utterly complex systems without knowing why they work. With the increasing use of computer programs to control important aspects of life, including airplanes and nuclear power stations, we should be concerned with the correctness and reliability of programs and computer systems. Evidence such as the explosion of the Ariane 5 space rocket on June 4, 1996, the faulty division instruction of the Intel Pentium processor, discovered in October 1994, or the Mars Pathfinder malfunctions in July 1997 tell us that even when the consequences of errors are extremely costly ($8 billion for Ariane, $475 million for replacing Pentium processors) the state of the art cannot safeguard us from such problems. It is simply too difficult for humans to completely understand such intricate systems.

The magic craft must thus be replaced by a *science* of computer programming. Like a construction engineer uses a mathematical model to ensure the strength and solidness of his designs prior to manufacturing them, a computer engineer should use mathematical models to ensure the correctness of his programs prior to their production.

We need to reason about the meaning of computer programs with mathematical exactness in order to make analysis precise and trustworthy, and we need tools to facilitate the use of such methods. This thesis develops a mathematical model for describing and analysing programs, demonstrates its application, and presents an automatic tool to support its use.

⋆ ⋆ ⋆

The fully-fledged science of computer programming may be an inaccessible utopia, but basic theoretical computer science has already contributed to the practical craft. Complexity theory, where the degree of difficulty for classes of problems are studied, is applied to construct cryptographic methods for safer computer systems (which are guaranteed to be more difficult to break). The

λ-calculus, initially introduced by Church in 1941 as a foundation for logics and mathematics and later successfully used as a theory of computations, has been instrumental in the study of so called types, which is nowadays a very important ingredient in understanding practical programming languages such as Java. The λ-calculus is also the source of parametric functions or procedures, which allow specification of computations which depend on their input arguments. On the more direct application of mathematical methods for computer systems, so called formal methods, it is appropriate to quote one of the major surveys of their impact on industry [CGR93]:

> A major conclusion of the study is that formal methods, while still immature in certain important respects, are beginning to be used seriously and successfully by industry to design and develop computer systems.

The point here is that computing systems are getting so complex and their application so widespread that we need a precise understanding of hardware and software, all the way from integrated circuits to languages and their implementations, in order to develop the systems of tomorrow. This understanding must be based on basic mathematical principles — methods such as intuition, rules of thumb, and empirically verified solutions ("this usually works very well") are past their limit. While hardware such as processors is now hundreds of times faster than ten years ago, computer system development is not keeping pace, but uses the same basic methods as before.

A foundational model of computing systems should be small, with few basic concepts and a reasonably simple theory, so that it can be understood and easily applied to rigorous description and analysis of systems. For purely computational aspects Turing machines [Tur36] is a canonical model in the sense that any computable function can be described; the λ-calculus [Chu41] does the same but captures aspects such as function definition and application in a clearer way. It can be seen as both a programming language where computations can be described, and as a mathematical theory where rigorous analysis can be performed.

Both Turing machines and the λ-calculus describe non-interactive behaviours: given an input, they produce an output. *Interactive* systems, sometimes called reactive systems, react to input and produce output continuously during execution. This may make the interacting party react to this output and produce new input to the system. The interacting party may be a person or perhaps other programs which execute concurrently with the system. Such interactive systems are difficult to describe succinctly with the mentioned foundational models.

In this thesis we study *concurrent* systems, which is a case of interactive systems where concurrent activities (henceforth called processes) interact as parts of a large system. For such systems there is yet no single canonical model; this is partly because there is such a diversity of aspects to study in concurrent systems. For example, the interaction or communication between components in a concurrent program can be described in different ways. One major view uses *shared variables*, and another uses *message passing.*

In the shared variable setting, the processes communicate by writing and reading variables shared among them. Changes made by one process are immediately visible to all others. Formal semantics and analysis methods in this paradigm normally use a separate global state encompassing all processes, representing the shared memory where variables reside.

In the message passing view communication is more explicit: a process sends a message to another over a named communication channel. In the basic case communication is local in the sense that only two processes take part: one as sender and one as recipient. Analysis is often simpler than for shared memory models since formal semantics and analysis methods do not need to use a separate global state.

Message passing and shared variables can be used to implement each other: a global state can be interpreted as a process which can be read and written by passing read and write messages to it, and message passing can be done by using one or more shared variables for each communication channel. The difference between the views on communication is not their expressiveness, or *what* can be described in them, but *how* different phenomena can be described in a clear and convenient way. The shared variables view is close to high-level programming languages for tightly coupled parallel computers, where memory is shared between processors; on the other hand, the message passing view is closer to loosely coupled distributed systems, where processors wanting to communicate must pass messages through a data network such as the Internet.

Within the message passing paradigm there is another dichotomy between synchronous and asynchronous communication. Synchronous communication is like handshaking; it requires both the sender and receiver to be present for communication to take place. Asynchronous communication is in a sense buffered: a message can be sent even if the recipient does not participate at the same instant.

An interesting variant of concurrent programming is concurrent constraint programming, where results are computed incrementally by adding partial information in the form of constraints on variables, such as $x > 7$, to a shared constraint store. The store can be interrogated to see if it entails a formula, such as $x < y$. The constraint store differs from a normal memory store, which holds values (such as 42) of variables; here a variable can have a *partial* value (such as "between 17 and x"). The predominant view on communication in concurrent constraint programming can be seen as a blend of message passing and shared variables: the constraint store is a separate global state, and the queries and updates of constraints are performed by *ask* and *tell* messages passed to the store. The ask and tell operations are similar to ordinary read and write operations, but logically and operationally more complex because of the nature of the store.

The work described in this thesis brings the shared memory and message passing perspectives together in one framework. Shared state such as a constraint store can be described more directly while staying in the basic message passing framework. This new power of expression is obtained by simplifying the message passing framework instead of, as might be supposed, complicating it.

1.1 Background

Concurrent programming is studied in this thesis by the use of *process calculi*, where the view on interaction is based on message passing. There are several families of such process calculi [Mil80, Hoa85, BW90]. The process calculus we develop has its roots in the Calculus of Communicating Systems (CCS) of Milner [Mil80, Mil89]. CCS in its pure form deals only with processes which communicate by synchronizing over communication ports, and which may behave indeterministically. Many extensions of CCS have been constructed to express and analyze different additional aspects of programs, as for example real-time concepts or probabilistic properties. In some extensions values of various types can be transmitted in the communication, for example expressions of integers or booleans. Such data types are separate entities from the processes and communication channels of the process calculus and require separate theories for formal treatment.

The particular aspect of concurrent programming we are interested in is mobility. We study systems of so called mobile processes where the structure of communication, saying which processes can communicate with which, is dynamically changing. Changing communication links can be seen as moving processes in a "virtual space" of interlinked processes, and thus the phrase "mobile processes".

Mobility in concurrent systems was probably first studied in the actor systems of Hewitt and Agha [Hew77, Agh86]. While this model has had considerable success and set off research in system architecture and object-oriented programming languages, its foundational theory lacks in concepts such as behavioural equivalence and compositionality.

The first to formulate an algebraic calculus of mobile processes, where the behaviours of processes are built structurally by a small set of well-defined operators, were Engberg and Nielsen [EN86]. Their Extended CCS (ECCS) was complicated by a heavy syntactic definition, differentiating between constants, variables and names as values, with complex interplay between them.

The direct ancestor of our new calculus is the π-calculus of Milner, Parrow and Walker [MPW92], which is based on the work of Engberg and Nielsen. In the π-calculus the only values passed over the named communication channels are again such channels. There is no difference in type between channels and values, and both are referred to as names. Name-passing turns out to be sufficient to encode passing of other types of data, such as booleans, integers, lists and even processes, resulting in one unified theory for processes, communication and data values. This insight that names suffice as values produced a simpler and leaner calculus which is easier to understand and use than ECCS. Calculi for mobile processes are sometimes also referred to by the more transparent phrase "name-passing calculi", but while the ability to pass names as values can be used to describe mobile processes there are also calculi for mobile processes where *only* processes (and not names) can be passed, e.g. Thomsen's CHOCS [Tho90].

When reasoning about interactive processes an important question we want to be able to answer is the following: when do two processes have equivalent interaction behaviour, so that we can replace one by the other in any concurrent system, and not be able to tell the difference? This is a theoretical kind of question, and it is very important in practise. We must know what it means for behaviours to be equal or unequal, otherwise we do not really know what behaviour means, and then we have no way of explaining what our system does. For this purpose we use the concept of bisimulation.

Bisimulation equivalence was originally developed by Park [Par81] and used for concurrent systems by Milner [Mil83]. It has become one of the most commonly used concepts of behavioural equivalence in process calculi. In essence, for two processes to be bisimilar, each interaction of one process must be mimicked by the other, reaching processes which are again bisimilar. In pure CCS this has a simple definition but in extensions such as the π-calculus care must be taken to properly define what "again bisimilar" means. As we shall see, in the π-calculus there are no less than three major variations.

Mobile processes appear in traditional application areas of process calculi such as operating systems, where they can be used to model resource allocation, and communication protocols, e.g. for mobile telephony. They also appear in models of programming, where names or processes can be used to represent pointers or objects which can be passed between processes. An example from the traditional areas where the π-calculus has been used is Orava and Parrow's analysis of communication protocols for mobile telephony [OP92]; examples of the latter type include data structures such as booleans or lists [Mil91a], object-oriented programming [Wal95], logic programming and unification [Ros93, Li94], and the programming language Pict [PT98], which uses the π-calculus as its foundational model. The π-calculus has also been used to model higher-order communication [San93] where processes are passed as values, and to encode the λ-calculus [Mil92] in clear-cut ways.

Over the nine years since the first papers on the π-calculus [MPW92] were published as technical reports in 1989, a variety of proof methods based on bisimulations [MPW92, San96c, San93, San95b, PS96b], algebraic rewrite systems [PS95, Lin95a] and model checking [Dam96] have been developed, and automated tools for these have started to emerge [VM94, FMQ95, FFG$^+$97, FGM$^+$98].

Constraint programming traces its origins back to [Mon74]. With the development of the concurrent constraint programming framework by Saraswat [Sar93], concurrency and constraints were connected and techniques from process calculi could be applied to introduce, e.g., indeterminacy. Semantics based on traces and closure operators were given by Saraswat, Rinard, and Panangaden [SRP91] and De Boer and Palamidessi [dBP91]; truly concurrent models were given by, e.g., Montanari and Rossi [MR95] and Gupta, Jagadeesan and Saraswat [GJS96].

Smolka developed the γ-calculus [Smo94] to achieve a framework which combines first-order concurrent constraint programming with higher-order functional programming and concurrent objects as a foundation of the programming language Oz [Smo95a]. The γ-calculus was inspired by the π-calculus; it is a name-passing calculus and thus supports mobile processes and, like the π-calculus, higher-order functional programming and concurrent objects. It adds logical variables and an elimination rule, which allows a basic form of constraint solving. While the first connection between concurrent constraints and mobile processes was made here, behavioural equivalences and proof methods were not a primary concern. The ρ-calculus of Niehren and Müller [NM95] is parameterized by an arbitrary constraint system, and can be seen as a generalization of the γ-calculus.

1.2 Results

The main contribution of this thesis, the *fusion calculus*, is a significant step towards a canonical calculus of concurrent processes. It simplifies and extends the π-calculus.

The fusion calculus contains the π-calculus as a proper subcalculus and thus inherits all its expressive power. In addition the fusion calculus contains actions akin to updating a shared state, and a scoping construct for bounding their effects. Therefore it is easier to represent computational models with shared state, including concurrent constraint formalisms. It is also easy to represent the so called strong reduction strategies in the λ-calculus, involving reduction under abstraction. In the π-calculus these tasks require elaborate encodings.

The dramatic main point of this thesis is that we achieve these improvements by *simplifying* the π-calculus rather than by adding features to it. The fusion calculus has only one binding operator where the π-calculus has two (input and restriction). It has a complete symmetry between input and output actions where the π-calculus has not. There is only one sensible variety of bisimulation congruence where the π-calculus has at least three. Proofs about the fusion calculus therefore are shorter and clearer. The price to be paid is the introduction of a new kind of actions, "fusion" actions. These actions are easily and cleanly incorporated into the formal framework.

In this thesis we present a broad foundational theory of the fusion calculus. We define its labelled and unlabelled operational semantics, and treat strong and weak bisimulation equivalences for both semantics in some detail, including complete axiom systems for finite terms. The equivalences are given symbolic characterizations, leading to algorithms and tools for equivalence checking. We apply the expressive power of the fusion calculus to give simple encodings of foundational calculi for functional and concurrent constraint programming.

In summary, the fusion calculus simplifies the π-calculus by reducing the number of binding operators and the number of bisimulation equivalences, and by making input and output symmetric like in pure CCS. We attain a calculus where concepts from other models of computation are more easily expressed than in the π-calculus, thereby taking a step towards a unified yet *simple* model of computation.

1.3 Overview

In the next chapter we give a technical background to the work in the rest of the thesis by presenting the π-calculus: its syntax and semantics, its bisimulation equivalences, and its variants. We also alleviate the search for further reading by introducing our bibliography on calculi for mobile processes, containing over 300 references.

In Chapter 3 we introduce the fusion calculus. The fundamental concepts of *fusion* and *scope* are presented, followed by the syntax of the calculus. We then define the operational semantics by three types of transition systems. The first type comes from process calculi; here transitions may carry labels. The second type originates from rewrite systems such as the λ-calculus; here transitions have no labels. The third type is a variant on the first, where the condition for a transition to be enabled is recorded in the label. In this chapter we prove that the π-calculus is a proper subcalculus of the fusion calculus.

Chapter 4 defines and motivates the strong bisimulation equivalence and we provide a simple characterization of its induced congruence, *hyperequivalence*. The weak hyperequivalence,where internal computations are not considered relevant for the behaviour, and its corresponding congruence are described and analysed. Congruences based on the unlabelled transition semantics are defined and proven to coincide with the labelled characterizations of the congruences. Based on the symbolic transition semantics we give symbolic characterizations of the equivalences, motivated by efficient implementation of algorithms in automatic tools. At the end of the chapter we return to the π-calculus and extend previous work by Sangiorgi by defining one of the bisimulation equivalences in the presence of a type of if-then-else construct.

Algebraic proof systems are presented in Chapter 5. We present axiom systems for hyperequivalence, and investigate variants of the weak hypercongruence by giving additional axioms, or "tau-laws", for them. Finally the π-calculus bisimulation of the previous chapter is axiomatized. All axiom systems are proved complete for terms with finite behaviour.

The expressive power of the calculus is demonstrated in Chapter 6. We do this first by giving a straight-forward encoding of the strong lazy λ-calculus, which admits reduction under λ abstraction. We also present encodings of three basic variants of the ρ-calculus, which is a foundational calculus for concurrent constraint programming with higher-order functions and concurrent objects. We contrast the simplicity of these encodings with an encoding of the similar γ-calculus in the π-calculus, where logical variables must be encoded by separate processes.

In Chapter 7 we present an algorithm for automatic verification of fusion calculus equivalences, and its implementation in a tool called the Mobility Workbench (MWB). The MWB is presented along with one simple example of its use for the fusion calculus and one more complex example of its use for the π-calculus.

The thesis concludes in Chapter 8 with a summary of the contribution and a discussion of open ends and future work.

1.4 Contribution

The material in this thesis is based on material from the following articles:

1. *The Mobility Workbench — A Tool for the π-Calculus* by Björn Victor and Faron Moller; published in the proceedings of the 6th International Conference on Computer Aided Verification (CAV'94) [VM94].

2. *Constraints as Processes* by Björn Victor and Joachim Parrow; published in the proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96) [VP96].

3. *The Update Calculus* by Joachim Parrow and Björn Victor; published in the proceedings of the 6th international conference on Algebraic Methods and Software Technology (AMAST'97) [PV97].

4. *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes* by Joachim Parrow and Björn Victor; to appear in the proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98) [PV98a].

5. *Concurrent Constraints in the Fusion Calculus* by Björn Victor and Joachim Parrow; to appear in the proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP'98) [VP98].

6. *The Tau-Laws of Fusion* by Joachim Parrow and Björn Victor; submitted for publication [PV98b].

7. *Symbolic Characterizations and Algorithms for Hyperequivalence and Open Bisimulation* by Björn Victor; submitted for publication [Vic98].

8. *Calculi for Mobile Processes: Bibliography and Web Pages* by Uwe Nestmann and Björn Victor; published in the *Bulletin of the EATCS*, number 64.

All proofs in the thesis are by me except for those in Section 5.2 which are mainly due to Joachim Parrow.

Chapter 2

Background on the π-Calculus

This chapter gives technical background on the aspects of calculi for mobile processes which are addressed in this thesis, and thus it provides a starting point for the other chapters. We present the π-calculus with its syntax, different versions of semantics and concepts of equivalence, and also present some of the variants of the π-calculus.

We also briefly point at several other aspects on mobile processes which are not in the focus of the thesis, and introduce the extensive bibliography on calculi for mobile processes found at the end of the thesis.

For a tutorial on the π-calculus with extensive examples and motivations we refer to Milner [Mil91a].

2.1 The π-Calculus

The π-calculus is a calculus of mobile processes, i.e., the communication structure can change dynamically over the execution of the processes. The mobility

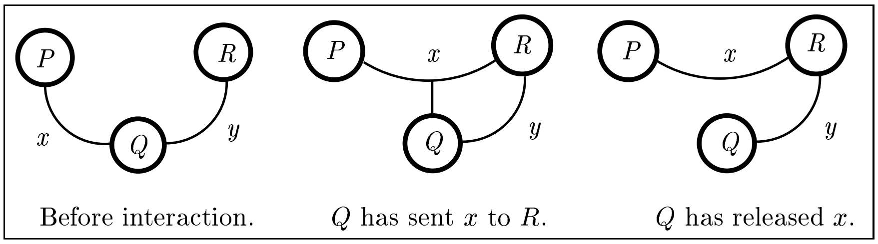

Figure 2.1: Mobile processes.

is achieved by allowing names of communication ports to be passed as values in interactions between processes, thereby changing the scope of the port. An example is the system of three processes P, Q and R in Figure 2.1, where P shares a communication port named x with Q (i.e., they both know the same port name x), but R does not know this name (left picture). Q can send x to R over the port named y, and thereafter P, Q and R can communicate directly over x (middle picture). If Q drops the name x, only P and R know it, and the "movement" of x is complete (right picture). This passing of names can be seen as passing pointers to objects or processes. For example, in the figure we can see x as a pointer to the process P. The name x can be passed over to R, which can then activate P by using x. In this sense we can say that P has "moved" from Q to R.

The π-calculus postulates a set of *names*, ranged over by $u, v, \ldots, z$, used to represent both communication ports, values, variables etc. We define a *polyadic* calculus where an output or input along a port may carry zero or more objects (as opposed to *monadic* calculi where exactly one object is carried). An interaction occurs between two processes, a sender performing an output and a receiver performing an input. The effect of the interaction is local to the receiver: its input variables become instantiated with the objects emitted by the sender. An example is:

$$\overline{u}\,vw\,.\,P \mid u(xy)\,.\,Q \quad \xrightarrow{\tau} \quad P \mid Q\{v/x, w/y\}$$

The *output prefix* $\overline{u}\,vw\,.\,P$ sends two objects v and w along u before continuing as the process P, and the *input prefix* $u(xy)\,.\,Q$, which binds x and y, instead receives two objects. Their interaction gives rise to a transition with the label τ, signifying internal activity, and a substitution of the received objects for the bound. As the binding is local to Q, the substitution is also local to Q. Additionally these prefixes imply temporal precedence: P and Q may not perform any transitions until the prefixes have been performed.

Another kind of binding occurs in the *restriction* operator: in $(\boldsymbol{\nu} x)P$ the name x in P is local to P (or "new in P"). This means that it is distinct from any name occurring outside P and from other names in P, and moreover that as P and the environment evolve during execution it will remain distinct.

We now present the formal syntax and semantics of the π-calculus. We write $\tilde{x}$ for a (possibly empty) finite sequence $x_1 \cdots x_n$ of names, and $|\tilde{x}|$ for the length of the sequence $\tilde{x}$.

Definition 1 *The* prefixes, *ranged over by* β, *and the* agents, *ranged over by* $P, Q, \ldots$, *are defined by*

$$\begin{array}{llllll}
\beta ::= & u(\tilde{x}) & \text{(Input)} & P ::= & \mathbf{0} & \text{(Inaction)} \\
& \overline{u}\tilde{x} & \text{(Output)} & & \beta \,.\, Q & \text{(Prefix)} \\
& \tau & \text{(Silent)} & & Q + R & \text{(Summation)} \\
& & & & Q \mid R & \text{(Composition)} \\
& & & & (\boldsymbol{\nu} x)Q & \text{(Restriction)} \\
& & & & [x = y]Q & \text{(Match)} \\
& & & & [x \neq y]Q & \text{(Mismatch)} \\
& & & & A(\tilde{x}) & \text{(Identifier)}
\end{array}$$

The *actions*, ranged over by α, consist of the prefixes and of output actions on the form $(\boldsymbol{\nu} z_1) \cdots (\boldsymbol{\nu} z_n)\overline{u}\,\tilde{x}$, abbreviated $(\boldsymbol{\nu}\tilde{z})\overline{u}\,\tilde{x}$, where $n \geq 0$ and all names in $\tilde{z}$ are also in $\tilde{x}$. If $n > 0$ we say it is a *bound output* action.

The input and output actions are called *communication* actions. In these, the names $\tilde{x}$ are the *objects* of the action, and the name u is the *subject*. The silent action τ has neither subject nor objects.

The null agent $\mathbf{0}$ is inactive and can not perform any actions. Prefixing an agent Q means that the agent must perform an action before acting like Q. We often omit a trailing $\mathbf{0}$ and write α for $\alpha \,.\, \mathbf{0}$ if no confusion can arise. Summation is alternative choice: when one of the summands performs an action, the other summand is discarded. Composition lets agents act in parallel.

A Match $[x = y]Q$ acts like Q if x and y are the same name; a Mismatch $[x \neq y]Q$ acts like Q if x and y are *not* the same. We use M, N to range over both match and mismatch operators. We write "match sequence" for a sequence of both match and mismatch operators, ranged over by $\tilde{M}, \tilde{N}$, and we say that $\tilde{M}$ *implies* $\tilde{N}$, written $\tilde{M} \Rightarrow \tilde{N}$, if the conjunction of all matches and mismatches in $\tilde{M}$ logically implies all elements in $\tilde{N}$, and that $\tilde{M} \Leftrightarrow \tilde{N}$ if $\tilde{M}$ and $\tilde{N}$ imply each other. We also use A to range over a possibly infinite set of identifiers, each with an associated nonnegative arity k and agent definition $A(z_1, \ldots, z_k) \stackrel{def}{=} P$ with pairwise distinct z_i such that $\mathrm{fn}(P) \subseteq \{z_1, \ldots, z_n\}$. The intuition is that $A(\tilde{z}) \stackrel{def}{=} P$ declares a possibly recursive agent with formal parameters $\tilde{z}$, and $A(\tilde{x})$ gives actual parameters $\tilde{x}$ to an instance of P.

We write $(\boldsymbol{\nu}\tilde{x})P$ for $(\boldsymbol{\nu} x_1) \cdots (\boldsymbol{\nu} x_n)P$. The names $\tilde{x}$ are said to be *bound* in $(\boldsymbol{\nu}\tilde{x})P$ and in $u(\tilde{x}) \,.\, P$. The bound names of Q are denoted $\mathrm{bn}(Q)$. The *free names* in P, denoted $\mathrm{fn}(P)$, are the names in P with a non-bound occurrence. We will not distinguish between alpha-variants of agents, i.e., agents differing only in the choice of bound names. We write $\mathrm{n}(M)$ for the names in a match or mismatch M and $\mathrm{n}(\alpha)$ for the (free and bound) names of an action α.

We use σ, ρ, ϱ to range over substitutions of names. When $\tilde{y}$ are pairwise distinct we write $\{\tilde{x}/\tilde{y}\}$ for a substitution which simultaneously replaces each y_i by x_i in an agent or action, alpha-converting the agent or action to avoid capture of bound names. For example, $u(x) \,.\, \overline{x}y \,.\, \mathbf{0}\{x/y\} = u(z) \,.\, \overline{z}x \,.\, \mathbf{0}$. We write $P\sigma\rho$ for $(P\sigma)\rho$.

Inspired by the Chemical Abstract Machine of Berry and Boudol [BB92], the semantics of the π-calculus is presented based on a structural congruence and a transition relation. This allows us to separate the physical structure of agents from their interaction. The structural congruence equates all agents we will never want to distinguish for any semantic reason.

Definition 2 *The* structural congruence, $\equiv$, *between agents is the least congruence satisfying the abelian monoid laws for Summation and Composition:*

$$\begin{array}{ccc} P+Q \equiv Q+P, & P+(Q+R) \equiv (P+Q)+R, & P+\mathbf{0} \equiv P \\ P \mid Q \equiv Q \mid P, & P \mid (Q \mid R) \equiv (P \mid Q) \mid R, & P \mid \mathbf{0} \equiv P \end{array}$$

and the scope laws

$$(\boldsymbol{\nu}x)\mathbf{0} \equiv \mathbf{0}, \quad (\boldsymbol{\nu}x)(\boldsymbol{\nu}y)P \equiv (\boldsymbol{\nu}y)(\boldsymbol{\nu}x)P, \quad (\boldsymbol{\nu}x)(P+Q) \equiv (\boldsymbol{\nu}x)P + (\boldsymbol{\nu}x)Q,$$
$$(\boldsymbol{\nu}x)MP \equiv M(\boldsymbol{\nu}x)P \;\; \textit{if } x \notin \mathrm{n}(M)$$

and also the scope extension *law* $P \mid (\boldsymbol{\nu}z)Q \equiv (\boldsymbol{\nu}z)(P \mid Q)$ *where* $z \notin \mathrm{fn}(P)$, *and the law for process identifiers:* $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ *if* $A(\tilde{x}) \stackrel{def}{=} P$.

Definition 3 *The family of* transitions $P \xrightarrow{\alpha} Q$ *is the least family satisfying the laws in Table 2.1. In this definition structurally equivalent agents are considered the same, i.e., if* $P \equiv P'$ *and* $Q \equiv Q'$ *and* $P \xrightarrow{\alpha} Q$ *then also* $P' \xrightarrow{\alpha} Q'$.

The scope extension law allows us to make the scope of a restriction smaller or larger, as long as we do not capture a free name. Let us see for example how to infer a transition such as

$$u(x) \,.\, P \mid (\boldsymbol{\nu}x)\overline{u}x \,.\, Q \xrightarrow{\tau} (\boldsymbol{\nu}x)(P \mid Q)$$

We show rather meticulously how to do this:

$$\begin{array}{rcll}
(\boldsymbol{\nu}x)\overline{u}x \,.\, Q & \equiv & (\boldsymbol{\nu}y)\overline{u}y \,.\, Q\{y/x\} & \\
 & & \alpha \text{ conversion, } y \notin \mathrm{fn}(Q) & \\
u(x) \,.\, P \mid (\boldsymbol{\nu}x)\overline{u}x \,.\, Q & \equiv & u(x) \,.\, P \mid (\boldsymbol{\nu}y)\overline{u}y \,.\, Q\{y/x\} & (2.1) \\
 & & \text{congruence property of } \equiv & \\
u(x) \,.\, P \mid (\boldsymbol{\nu}y)\overline{u}y \,.\, Q\{y/x\} & \equiv & (\boldsymbol{\nu}y)(u(x) \,.\, P \mid \overline{u}y \,.\, Q\{y/x\}) & (2.2) \\
 & & \text{scope extension, } y \notin \mathrm{fn}(P) & \\
u(x) \,.\, P \mid \overline{u}y \,.\, Q\{y/x\} & \xrightarrow{\tau} & P\{y/x\} \mid Q\{y/x\} & \\
 & & \text{by the } \pi\text{COM rule} & \\
(\boldsymbol{\nu}y)(u(x) \,.\, P \mid \overline{u}y \,.\, Q\{y/x\}) & \xrightarrow{\tau} & (\boldsymbol{\nu}y)(P\{y/x\} \mid Q\{y/x\}) & \\
 & & \text{by the } \pi\text{PASS rule} & \\
(\boldsymbol{\nu}y)(P\{y/x\} \mid Q\{y/x\}) & \equiv & (\boldsymbol{\nu}x)(P\{y/x\} \mid Q\{y/x\})\{x/y\} & (2.3) \\
 & & \alpha \text{ conversion, } x \notin \mathrm{fn}(P\{y/x\}, Q\{y/x\}) & \\
(\boldsymbol{\nu}x)(P\{y/x\} \mid Q\{y/x\})\{x/y\} & \equiv & (\boldsymbol{\nu}x)(P \mid Q) & (2.4) \\
 & & \text{by def of substitutions} & \\
u(x) \,.\, P \mid (\boldsymbol{\nu}x)\overline{u}x \,.\, Q & \xrightarrow{\tau} & (\boldsymbol{\nu}x)(P \mid Q) & \\
 & & \text{by (2.1),(2.2),(2.3),(2.4)} &
\end{array}$$

πPREF	$\dfrac{-}{\alpha . P \xrightarrow{\alpha} P}$	πSUM	$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
πPAR	$\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	πCOM	$\dfrac{P \xrightarrow{u(\tilde{x})} P',\ \ Q \xrightarrow{\overline{u}\tilde{y}} Q',\ \ \lvert\tilde{x}\rvert = \lvert\tilde{y}\rvert}{P \mid Q \xrightarrow{\tau} P'\{\tilde{y}/\tilde{x}\} \mid Q'}$
πPASS	$\dfrac{P \xrightarrow{\alpha} P',\ z \notin \mathrm{n}(\alpha)}{(\boldsymbol{\nu} z)P \xrightarrow{\alpha} (\boldsymbol{\nu} z)P'}$	πOPEN	$\dfrac{P \xrightarrow{(\boldsymbol{\nu}\tilde{y})\overline{u}\,\tilde{x}} P',\ z \in \tilde{x} - \tilde{y},\ u \neq z}{(\boldsymbol{\nu} z)P \xrightarrow{(\boldsymbol{\nu} z\tilde{y})\overline{u}\,\tilde{x}} P'}$
πMATCH	$\dfrac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$	πMISMATCH	$\dfrac{P \xrightarrow{\alpha} P',\ x \neq y}{[x \neq y]P \xrightarrow{\alpha} P'}$

Table 2.1: Laws of action for the π-calculus.

We will in some situations want to abstract from the silent τ action, regarding it as internal and unobservable. Then we use the *weak* transitions:

Definition 4 *The* weak *transitions* $P \stackrel{\alpha}{\Longrightarrow} Q$ *are defined as follows:*

- *if* $\alpha = \tau$*, then* $P \stackrel{\alpha}{\Longrightarrow} Q$ *means* $P \Longrightarrow Q$*;*
- *if* $\alpha \neq \tau$*, then* $P \stackrel{\alpha}{\Longrightarrow} Q$ *means* $P \Longrightarrow\xrightarrow{\alpha}\Longrightarrow Q$*;*

where $P \Longrightarrow Q$ *iff* $P(\xrightarrow{\tau})^* Q$.

2.2 Bisimulation Equivalences in the π-Calculus

The notion of equivalence is based on the well known idea of bisimulation, which has many pleasant properties, e.g., it has simple proof techniques and is decidable for large classes of agents. We look at three variants of strong bisimulation in the π-calculus: *early* and *late* bisimulation were defined in the first papers on the π-calculus [MPW92], and *open* bisimulation was later defined by Sangiorgi [San96c].

Before investigating the π-calculus bisimulations we show the basic bisimulation as defined by Park and Milner and used in CCS. To make the definition sensible in the π-calculus, where actions can carry bound names, we add the side condition $\mathrm{bn}(\alpha) \cap \mathrm{fn}(P, Q) = \emptyset$ so that free names are not captured.

Definition 5 *A binary symmetric process relation* $\mathcal{S}$ *is a* basic bisimulation *if* $(P, Q) \in \mathcal{S}$ *implies*

- *if* $P \xrightarrow{\alpha} P'$ *where* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(P, Q) = \emptyset$,
  then there exists a Q' *s.t.* $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in \mathcal{S}$.

Two agents P and Q are basic bisimilar*, written* $P \stackrel{\cdot}{\sim}_B Q$*, if they are related by a basic bisimulation.*

However, this bisimulation is too weak to be useful by itself in the π-calculus. The problem is the way basic bisimulation treats input actions. Intuitively the objects of an input action are placeholders for whatever is being received, and can be instantiated by arbitrary names, but basic bisimulation only requires that the continuations are bisimilar for instantiation where the received objects are different from the other names of the agents. This means, e.g., that

$$u(x) \, . \, [x = y]\overline{x}y \;\; \stackrel{\cdot}{\sim}_B \;\; u(x) \, . \, \mathbf{0}$$

even though $u(x) \, . \, [x = y]\overline{x}y \mid \overline{u}\,y \xrightarrow{\mathbf{1}}\xrightarrow{\overline{y}y}$ while $u(x) \, . \, \mathbf{0} \mid \overline{u}\,y \xrightarrow{\mathbf{1}} \mathbf{0}$. So basic bisimilarity is not preserved even by parallel composition.

Requiring the continuations after input actions to be bisimilar for whatever names are received can be specified in two ways:

Definition 6 *A binary symmetric process relation* $\mathcal{S}$ *is a* late bisimulation *if* $(P, Q) \in \mathcal{S}$ *implies*

1. *if* $P \xrightarrow{u(\tilde{x})} P'$ *where* $\tilde{x} \cap \mathrm{fn}(Q) = \emptyset$,
   then there exists Q' *s.t.* $Q \xrightarrow{u(\tilde{x})} Q'$ *and for all* $\tilde{y}$, $(P'\{\tilde{y}/\tilde{x}\}, Q'\{\tilde{y}/\tilde{x}\}) \in \mathcal{S}$;

2. *if* $P \xrightarrow{\alpha} P'$ *where* α *is not an input action and* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$,
   then there exists Q' *s.t.* $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in \mathcal{S}$.

P and Q are late bisimilar *or* late bisimulation equivalent*, written* $P \stackrel{\cdot}{\sim}_l Q$*, if* $(P, Q) \in \mathcal{S}$ *for some late bisimulation* $\mathcal{S}$.

By swapping the quantifiers in the input case, another variant becomes:

Definition 7 *A binary symmetric process relation* $\mathcal{S}$ *is an* early bisimulation *if* $(P, Q) \in \mathcal{S}$ *implies*

1. *if* $P \xrightarrow{u(\tilde{x})} P'$ *where* $\tilde{x} \cap \mathrm{fn}(Q) = \emptyset$,
   then for all $\tilde{y}$ *there exists* Q' *s.t.* $Q \xrightarrow{u(\tilde{x})} Q'$ *and* $(P'\{\tilde{y}/\tilde{x}\}, Q'\{\tilde{y}/\tilde{x}\}) \in \mathcal{S}$;

2. *if* $P \xrightarrow{\alpha} P'$ *where* α *is not an input action and* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$,
   then there exists Q' *s.t.* $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in \mathcal{S}$.

P and Q are early bisimilar *or* early bisimulation equivalent*, written* $P \stackrel{\cdot}{\sim}_e Q$*, if* $(P, Q) \in \mathcal{S}$ *for some early bisimulation* $\mathcal{S}$.

Liberally speaking, early bisimulation lets the simulating agent peek at the names being input, and for each set of names it must select a simulating transition. Late bisimulation on the other hand requires the simulating agent to find one transition which simulates for any names.

Late and early bisimulation equivalences are not full congruences (they are not preserved by input prefix), but such relations can be obtained by requiring that two agents are bisimilar under any substitution of names.

Definition 8 *P and Q are (early/late) congruent if for all substitutions σ, it is the case that $P\sigma$ and $Q\sigma$ are (early/late) bisimilar. We write $P \sim_e Q$ if P and Q are early congruent and $P \sim_l Q$ if P and Q are late congruent.*

Proposition 2.1 [MPW92] *The (early/late) congruence is the largest congruence in (early/late) bisimulation equivalence.*

Sangiorgi proposed open bisimulation equivalence as a way of integrating the substitution closure into the bisimulation definition. To handle restriction, which declares a new name which is different from all other known names, *distinction* relations are used. If (x, y) are in a distinction relation then x and y must be kept different.

Definition 9 *A* distinction *is a finite symmetric and irreflexive relation on names. A substitution σ* respects *a distinction D if $(x, y) \in D$ implies $\sigma(x) \neq \sigma(y)$.*

Subsuming the symmetry, we will write $\{(x, y)\}$ for the distinction $\{(x, y), (y, x)\}$.

Definition 10 *The set $\mathcal{S} = \{S_D\}_\mathcal{D}$ of binary symmetric process relations is an* open bisimulation *if for each $S_D \in \mathcal{S}$ and all substitutions σ which respect D, $(P, Q) \in S_D$ implies*

1. *if $P\sigma \xrightarrow{(\boldsymbol{\nu}\tilde{x})\overline{u}\,\tilde{y}} P'$ with $\tilde{x} \cap \mathrm{fn}(Q) = \emptyset$,*
   then $Q\sigma \xrightarrow{(\boldsymbol{\nu}\tilde{x})\overline{u}\,\tilde{y}} Q'$ such that $(P', Q') \in S_{D'}$,
   where $D' = D\sigma \cup (\tilde{x} \times \tilde{x} - \mathbf{1}) \cup (\tilde{x} \times \mathrm{fn}(P\sigma, Q\sigma))$;

2. *if $P\sigma \xrightarrow{\gamma} P'$ otherwise, with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$,*
   then $Q\sigma \xrightarrow{\gamma} Q'$ such that $(P', Q') \in S_{D\sigma}$.

P and Q are open D-bisimilar *or* open D-bisimulation equivalent, *written $P \sim_D^o Q$, if $(P, Q) \in S_D \in \mathcal{S}$ for some open bisimulation $\mathcal{S}$. We say that P and Q are* open bisimulation equivalent, *written $P \sim^o Q$, if $P \sim_\emptyset^o Q$.*

Notice that for the calculus without restriction, open bisimulation equivalence coincides with the largest substitution closed basic bisimulation [San96c], i.e., the largest basic bisimulation relation $\mathcal{S}$ with the property that $(P, Q) \in \mathcal{S}$ implies $(P\sigma, Q\sigma) \in \mathcal{S}$ for any substitution σ.

Proposition 2.2 [San96c] *Open bisimulation equivalence is a congruence.*

Comparing the instantiation requirements of open to late and early, we see that open requires not only that the simulation works for any values of input names, but for *all* substitutions except those involving a restricted name.

Open bisimulation equivalence is strictly stronger than late bisimulation equivalence, which in turn is strictly stronger than early bisimulation equivalence. For a simple example distinguishing open equivalence from late congruence, consider the following [San96c]:

$$\tau\,.\,\tau+\tau \quad \overset{\not\sim^{o}}{\dot\sim_{l}} \quad \tau\,.\,\tau+\tau+\tau\,.\,[x=y]\tau$$

An example distinguishing late from early is similar (based on an example in [MPW92]):

$$x(u)\,.\,\tau+x(u) \quad \overset{\not\dot\sim_{l}}{\dot\sim_{e}} \quad x(u)\,.\,\tau+x(u)+x(u)\,.\,[u=z]\tau$$

While early and late bisimulation is defined for the full π-calculus as described in Section 2.1, open bisimulation equivalence is undefined for the calculus with the mismatch operator. We will return to this issue in Chapter 4.

The early and late bisimulation equivalences and congruences have been given complete axiomatizations in [PS95], and open bisimulation equivalence was axiomatized in [San96c].

The *weak* open bisimulation and D-bisimilarity are obtained by replacing $Q\sigma \xrightarrow{(\boldsymbol{\nu}\tilde{x})\overline{u}\,\tilde{y}} Q'$ with $Q\sigma \xRightarrow{(\boldsymbol{\nu}\tilde{x})\overline{u}\,\tilde{y}} Q'$ and $Q\sigma \xrightarrow{\alpha} Q'$ with $Q\sigma \xRightarrow{\alpha} Q'$ in Definition 10. Weak early bisimulation is obtained in the same way. For weak late bisimulation the case for input actions must be $Q \Longrightarrow\xrightarrow{u(\tilde{x})} Q'$ for the resulting relation to be transitive (see [San96c] for a discussion). The weak early and late bisimulation congruences have been axiomatized by Lin [Lin95a].

2.2.1 Alternative Semantics for the π-Calculus

For a subcalculus, the semantics can also be given by a reduction relation (an unlabelled transition system). The fragment of the π-calculus considered uses only *guarded* summation, and match/mismatch operators can only be combined with prefixes. We call the subcalculus π_g and define it formally by the following BNF equation:

$$P ::= \mathbf{0} \;\big|\; \sum \tilde{M}_i\alpha_i\,.\,P_i \;\big|\; P_1 \mid P_2 \;\big|\; (\boldsymbol{\nu}x)P \;\big|\; A(\tilde{y})$$

The relation $\rightarrow$ is the least relation satisfying the rules in Table 2.2, where again structurally equivalent agents are considered the same, i.e., if $P \equiv P'$ and $Q \equiv Q'$ and $P \rightarrow Q$ then also $P' \rightarrow Q'$.

$$(\cdots + \tilde{M}u(\tilde{x})\,.\,P) \mid (\tilde{N}\overline{u}\,\tilde{y}\,.\,Q + \cdots) \to P\{\tilde{y}/\tilde{x}\} \mid Q$$
$$\text{if } |\tilde{x}| = |\tilde{y}| \text{ and } \tilde{M} \Leftrightarrow \tilde{N} \Leftrightarrow \mathsf{true}$$

$$\frac{P \to P'}{[x=x]P \to P'} \qquad \frac{P \to P',\ x \neq y}{[x \neq y]P \to P'} \qquad \frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

$$\frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \frac{P \to P'}{P + Q \to P'} \qquad \frac{P \to P'}{(\boldsymbol{\nu} x)P \to (\boldsymbol{\nu} x)P'}$$

Table 2.2: Reduction rules for the π-calculus.

Together with an observation predicate the reduction relation can be used to define *barbed* bisimulation equivalence. The idea here is to consider two agents equivalent if their reductions simulate and they have the same observations as defined below, and closing under arbitrary contexts to obtain a congruence.

Definition 11 *The* observation relation *is the least relation satisfying the rules below.*

$x(\tilde{y})\,.\,P \downarrow x$	
$\overline{x}\,\tilde{y}\,.\,P \downarrow x$	
$[x=x]P \downarrow y$	*if* $P \downarrow y$
$[x \neq z]P \downarrow y$	*if* $P \downarrow y$ *and* $x \neq z$
$A(\tilde{y}) \downarrow z$	*if* $A(\tilde{x}) \stackrel{def}{=} P$, $\lvert\tilde{y}\rvert = \lvert\tilde{x}\rvert$ *and* $P\{\tilde{y}/\tilde{x}\} \downarrow z$
$(P \mid Q) \downarrow x$	*if* $P \downarrow x$ *or* $Q \downarrow x$
$(P + Q) \downarrow x$	*if* $P \downarrow x$ *or* $Q \downarrow x$
$(\boldsymbol{\nu} x)P \downarrow y$	*if* $P \downarrow y$ *and* $x \neq y$

Intuitively, $P \downarrow x$ if $P \xrightarrow{\alpha} P'$ with $\mathrm{n}(\mathrm{subj}(\alpha)) = x$. We sometimes write $P \to^{*}\downarrow x$ for $\exists P'$ s.t. $P \to^{*} P'$ and $P' \downarrow x$.

Definition 12 *A* barbed bisimulation *is a symmetric binary relation $\mathcal{S}$ between agents such that $P \mathrel{\mathcal{S}} Q$ implies:*

1. *If $P \to P'$ then $Q \to Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*
2. *If $P \downarrow x$ for some x, then $Q \downarrow x$.*

P is barbed bisimilar to Q, written $P \stackrel{\centerdot}{\sim}_b Q$, if $P \mathrel{\mathcal{S}} Q$ for some barbed bisimulation $\mathcal{S}$.

Barbed bisimulation is a very weak equivalence: e.g., $u\,.\,v\,.\,\mathbf{0} \stackrel{\centerdot}{\sim}_b u\,.\,\mathbf{0}$, so we close under contexts to obtain a congruence:

Definition 13 *Two agents P and Q are* barbed congruent*, written $P \sim_b Q$, if for all contexts $C[\cdot]$, it holds that $C[P] \stackrel{\centerdot}{\sim}_b C[Q]$.*

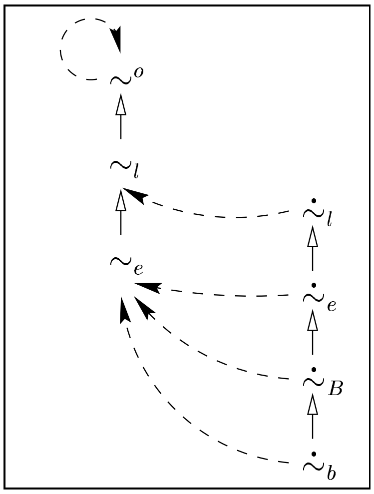

Figure 2.2: Summary of the bisimulation equivalences and congruences in the π-calculus.

For the π-calculus, barbed congruence coincides with early congruence [San93].

In previous presentations of reduction semantics for the π-calculus no mismatch operator has been used and the match operator can be handled by adding the rule $[x = x]P \equiv P$ to the structural congruence. The mismatch operator cannot be handled this way: we cannot have a rule $[x \neq y]P \equiv P$ if x is different from y since, e.g., $u(x) \,.\, [x \neq y]P$ should not be equivalent to $u(x) \,.\, P$. It becomes cumbersome to define rules for deriving reductions from agents such as $\tilde{M}_1 u(\tilde{x}) \,.\, P \mid \tilde{M}_2(R \mid \tilde{M}_3 \overline{u}\tilde{y} \,.\, Q)$, so we choose to consider only the subcalculus π_g and to treat match and mismatch uniformly. π_g is sufficient for our needs in Chapter 6.

Variants of the labelled transition semantics can also be given where the transitions are decorated with a symbolic condition which must be satisfied for the transition to be enabled. E.g.,

$$[x = y]\alpha \,.\, P \xrightarrow{[x=y],\alpha} P \qquad [x \neq y]u(\tilde{z}) \,.\, P \mid \overline{v}\tilde{w} \,.\, Q \xrightarrow{[x \neq y, u=v],\tau} P\{\tilde{w}/\tilde{z}\} \mid Q$$

We will study such *symbolic semantics* for the fusion calculus in Section 3.4.

In Figure 2.2 we show a summary of the bisimulation equivalences and congruences in the π-calculus. Barbed bisimilarity ($\dot\sim_b$) only considers reductions and observations. Basic bisimilarity ($\dot\sim_B$) considers all transitions, but no instantiations. Early ($\dot\sim_e$) and late ($\dot\sim_l$) considers instantiations of input objects; the corresponding congruences ($\sim_e$ and $\sim_l$) are obtained by closing under substitutions. Open bisimilarity ($\sim^o$) considers instantiations of all names except bound output objects.

In the diagram the solid arrows represent inclusion, and the dashed arrows represent "largest congruence contained in".

2.3 Variants of the π-Calculus

The *higher-order* π-calculus [San93], or HOπ, extends the π-calculus by allowing not only names but also agents and agent abstractions to be passed as values, thus making the mobility of processes more explicit. As an example of interaction in HOπ, consider

$$a(X) \,.\, (\overline{z} \,.\, X \mid \overline{x} \,.\, R) \mid \overline{a}\langle x \,.\, Q\rangle \,.\, P \xrightarrow{\tau} (\overline{z} \,.\, x \,.\, Q \mid \overline{x} \,.\, R) \mid P$$

Even though HOπ may at first seem much more expressive that the ordinary (first-order) π-calculus, an important point is that it can be faithfully encoded in the π-calculus [San93]. HOπ is still very useful for reasoning about process mobility at a more abstract level.

The *asynchronous* π-calculus [HT91, Bou92, ACS96], or π_a, is a variant of the π-calculus where an output prefix is always on the form $\overline{u}\,\tilde{x} \,.\, \mathbf{0}$. An output action cannot guard other actions, so communication is asynchronous in the sense that it is not possible for the sender to determine when the output is consumed by an input, or for an asynchronous observer to directly observe input actions, and therefore π_a needs different equivalences than the ones defined above. One of the most notable differences is that $u(x) \,.\, \overline{u}\,x + \tau$ is equivalent to τ in the asynchronous setting (try the context $\overline{u}\,y \mid \cdot$ to see why), while this obviously does not hold in the synchronous calculus.

In programming language design the distinction between synchronous and asynchronous communication is important, since the latter is easier to implement, being closer to the primitives available in actual distributed systems.

While the asynchronous π-calculus has been successfully used to, e.g., formally confirm the validity of optimization techniques in the Pict programming language [NP96], it has been shown to be strictly less expressive than the standard (synchronous) π-calculus [Pal97]. The basis of this result is that in the asynchronous calculus, certain symmetries in distributed algorithms cannot be broken, and it implies that the full π-calculus cannot be encoded in the asynchronous π_a up to any reasonable equivalence. The key is that while output-guarded summation is possible in the full π-calculus, it cannot be encoded in π_a.

Quite obviously the asymmetry between input and output of the π-calculus is taken even further in the asynchronous π-calculus. A seemingly different direction is taken in the πI-calculus, the π-calculus with only *internal mobility* [San96b], where symmetry is achieved by requiring outputs to always have bound objects: all output prefixes are on the form $(\boldsymbol{\nu}\tilde{x})\overline{u}\,\tilde{x} \,.\, P$. This simplifies the theory of the π-calculus rather dramatically: the basic bisimulation (Definition 5) is a congruence, and the weak equivalence is obtained by replacing $Q \xrightarrow{\alpha} Q'$ with $Q \overset{\alpha}{\Longrightarrow} Q'$. No early/late/open variants appear, and the strong bisimulation equivalence has a very simple axiomatization. Match and mismatch operators become superfluous.

Interestingly, the asynchronous π_a, which admits external mobility (output of free names) can be encoded in πI which admits only internal mobility (output of bound names) [Bor96]. This result is only up to a rather weak equivalence, barbed bisimulation (see Definition 12), but in a sense allows external mobility to be programmed by only internal mobility.

Modelling systems with shared state is difficult in the π-calculus, where effects of interactions are local: sending a value to a process has effect only in the recipient process, while updating a shared variable has effect in all processes.

To model shared state in a process calculus, two approaches have been considered. Smolka [Smo94] extends the calculus with logical variables and suitable operators for updating them separately from their binding. In his γ-calculus, an example of interaction is

$$\exists x(x = y \mid P \mid Q) \quad \longrightarrow \quad (P \mid Q)\{y/x\}$$

where the reduction of the term $x = y$ results in updating x in all parallel subterms in the scope of x by substituting y for x.

With the π-calculus viewpoint that *names* should be sufficient as primitive values, in [VP96] we encoded logical variables by separate processes, variable handlers, which can be read, updated, etc. by sending requests and receiving responses. Although this work allows us to lift π-calculus techniques to the γ-calculus, as we will see in Chapter 6 this approach is quite involved, making the system as a whole rather complex. In that chapter we will also see how the ρ-calculus, which is a generalization of the γ-calculus, can be more elegantly encoded in the fusion calculus.

Finally, some concluding observations:

The π-calculus is riddled by a few curious asymmetries. In π there are two binding operators, input binding $a(\tilde{x}) \,.\, P$ meaning "receive something for $\tilde{x}$ before continuing with P" and restriction $(\boldsymbol{\nu} x)P$ meaning "x in P will always be distinct from other names". Since input actions must be accompanied by a binding there is a fundamental asymmetry between input and output. The locality of communication effects in the π-calculus is due to this double role of the input prefix: binding the names being changed by the effect and temporally guarding the effect.

The πI-calculus attains symmetry by *restricting* the π-calculus; in the work described in this thesis we reach this goal by *removing* restrictions already in the π-calculus. We started this line of work with the *update calculus* [PV97], where many of the ideas of the fusion calculus appeared. The symmetry was not complete in the update calculus, and for this reason the calculus did not generalize to polyadicity.

2.4 Further Topics in Mobile Processes

The topics discussed in the previous section do not constitute a complete list of areas covered in previous work on calculi for mobile processes. We can also mention type systems, modal logics, non-interleaving semantics, physical distribution, programming languages, cryptographic protocols, connections to linear logic and category theory,...

For further reading on these or other topics in calculi for mobile processes, we refer to the extensive bibliography at the end of the thesis. This bibliography is collected by Uwe Nestmann and the present author for use with BibTeX, and can be found at the URLs `http://www.cs.auc.dk/mobility/pi.bib` (in BibTeX format) and `http://www.cs.auc.dk/mobility/pi-bib.dvi` (as a formatted DVI file). Furthermore, the bibliography is regularly mirrored at `http://liinwww.ira.uka.de/bibliography/Theory/pi.html` where it provides the user with a very convenient service for online searching of the bibliographic information. Nevertheless, the web page on mobile processes, `http://www.cs.auc.dk/mobility/`, will usually yield a slightly more up-to-date bibliography.

The bibliography should ideally cover every publication in the context of the π-calculus, its variants, descendants, simplifications, and extensions, including higher-order forms, and implementations. In general, the listed items should address mobility phenomena via the investigation of mathematically tractable calculi. So the bibliography does not include just anything about process algebra (unless mobility is explicitly tackled) or mobile systems (unless a foundational calculus is involved).

Any further items for inclusion and any corrections or updates of existing information should be reported to both `uwe@cs.auc.dk` and `Bjorn.Victor@DoCS.UU.SE` and will be included in future versions if they comply with the scope of the bibliography.

The bibliography was initially created by the present author in November 1994, and after a call for updates on the mailing list `concurrency@cwi.nl`, it was merged with a similar bibliography collected by Uwe Nestmann. Ever since then we have worked with joint forces to keep the bibliography up-to-date. In March 1995 the bibliography was added to the *Collection of Computer Science Bibliographies* at `http://liinwww.ira.uka.de/bibliography/`, administered by Alf-Christian Achilles. In 1997 the home site of the bibliography was incorporated into the mobility web pages at `http://www.cs.auc.dk/mobility/`.

As of April 28, 1998, the bibliography contained 326 references including 54 references to online publications. There were 244 references to papers, the other being proceedings from conferences. Figure 2.3 shows that after its birth in 1989, the π-calculus community continuously grew into its current state, where over 40 research papers are published every year. The figure includes only papers (not proceedings). Note that articles published in journals appear only with that publication entry even if they were originally published earlier, as technical reports or in conference proceedings.

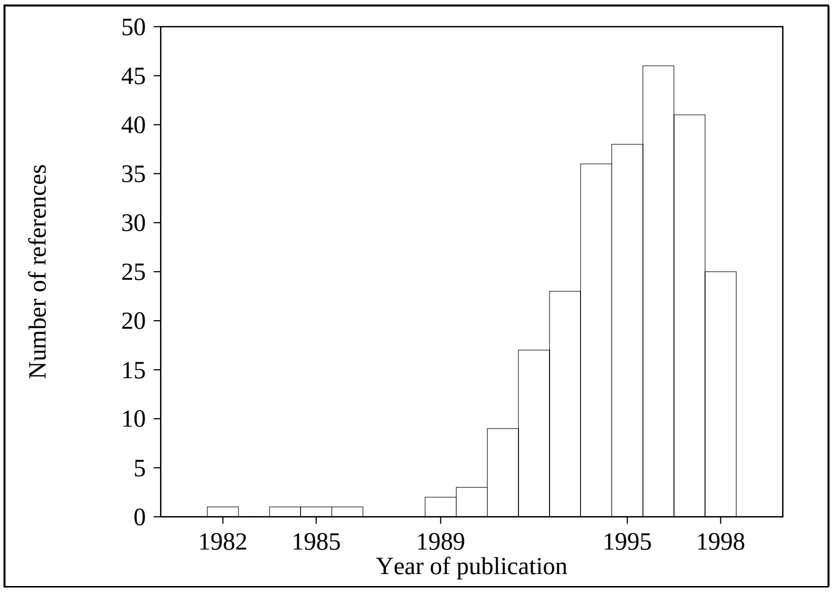

Figure 2.3: Number of papers per year (until April 28, 1998)

Chapter 3

Syntax and Semantics

In this chapter we present the syntax of the fusion calculus, followed by three different versions of operational semantics. First we present the *labelled transition* semantics, which intuitively says what parts of agents can do, and how these behaviours can be combined in a bottom-up fashion. This is the traditional type of semantics which is used, e.g., for CCS [Mil89] and it was the first type given for the π-calculus [MPW92]. In this section we also show that the π-calculus is a proper subcalculus of the fusion calculus. Secondly, we give a *reduction-based* semantics (or unlabelled transition semantics), which can perhaps be seen as giving a top-down intuition for what the system as a whole can do, based on its parts. Some readers may find this type of semantics easier to understand, since there is only one type of transition. It is also familiar from rewrite systems such as the λ-calculus. Finally, we present a *symbolic* labelled transition semantics, which is motivated by the implementation of tools for analysing systems described in the fusion calculus. Here transitions are decorated with a symbolic condition which must be satisfied for the transition to be enabled.

To calm readers worried by this abundance of semantics and transition systems, in the next chapter we will show that they all match, and describe the same phenomena from different viewpoints.

Like the π-calculus, the fusion calculus postulates a set of *names*, ranged over by $u, v, \ldots, z$, used to represent communication ports, values, variables etc. An interaction occurs between two processes, a sender performing an output and a receiver performing an input. In the fusion calculus the input primitive does *not* bind names, in contrast with other value-passing process calculi such as the π-calculus. The effect of an interaction is that the output objects and input objects become identified in what we call a *fusion*. It is important to note that the effect of a fusion is not necessarily local but regulated by the use of a *scope* operator, and that fusions are symmetric with respect to input-output polarities.

As we will see, the separation between input and binding not only leads to a simpler calculus but also gives increased expressiveness.

Formally a fusion is just an equivalence relation on names signifying that names related by it should be considered equal. This effect is applicable not only to the receiver but also to the sender and to any process running in parallel with them. Therefore it can be thought of as an update of a (not explicitly represented) shared state. An example interaction:

$$\overline{u}\,vw\,.\,P \mid u\,xy\,.\,Q \quad \xrightarrow{\{v=x,w=y\}} \quad P \mid Q$$

The output prefix $\overline{u}\,vw$ sends two objects v and w along u, and the input prefix $u\,xy$, receives two objects into x and y. The interaction results in a fusion of v and x, and of w and y. This equivalence relation is $\{(v,x),(x,v),(w,y),(y,w)\} \cup \mathbf{1}$, where $\mathbf{1}$ is the identity relation on names. We write this equivalence more compactly as $\{v = x, w = y\}$. The fusion is placed above the arrow to signify its unbounded scope. When further transitions from the derivative $P \mid Q$ are considered, they will be treated in an environment which identifies v with x and w with y. Our definitions of equivalence between processes in Chapter 4 will ensure this effect. The fact that the fusion is global can be seen by including further parallel processes, which will also be similarly affected by the environment, for example in

$$\overline{u}\,vw\,.\,P \mid u\,xy\,.\,Q \mid R \mid S \quad \xrightarrow{\{v=x,w=y\}} \quad P \mid Q \mid R \mid S$$

This ability to express global effects is the reason why the fusion calculus gains in expressiveness over the π-calculus.

The fact that the interaction is symmetric can be seen by inverting the input-output polarities of the actions:

$$u\,vw\,.\,P \mid \overline{u}\,xy\,.\,Q \quad \xrightarrow{\{v=x,w=y\}} \quad P \mid Q$$

The fusion here is the same as before. Therefore the designations "input" and "output" are arbitrary and a more appropriate terminology might be "action" and "co-action", though in this thesis we shall retain the more familiar terms from traditional value-passing calculi.

The single binding operator in the fusion calculus is called *scope* and is written $(x)P$, meaning that the x in P is local. In a sense it is the common denominator of input binding and restriction in the π-calculus: an instantiation of the bound name is neither mandatory (like in input) nor impossible (like in restriction). The main idea is that scopes can be used to delimit the extent of fusions. As a first example we show how the π-calculus interaction can be emulated by scoping the input names:

$$\overline{u}\,vw\,.\,P \mid \Big((x)(y)u\,xy\,.\,Q\Big) \mid R \mid S \quad \xrightarrow{\mathbf{1}} \quad P \mid \Big(Q\{v/x, w/y\}\Big) \mid R \mid S$$

Here $\mathbf{1}$ is an inert action (formally it is just the identity fusion) and the effect of the interaction does not extend beyond the scopes. Since x and y are bound they are substituted by some non-bound names fused to them.

The increased expressiveness over the π-calculus is that scopes may be placed in arbitrary places and not only directly over the input actions. For example

$$(x)\Big(\overline{u}\,vw\,.\,P \mid (y)\Big(u\,xy\,.\,Q \mid R\Big) \mid S\Big) \;\xrightarrow{\mathbf{1}}\; \Big(P \mid \Big(Q \mid R\Big)\{w/y\} \mid S\Big)\{v/x\}$$

where the scopes determine the extent and direction of the resulting substitutions.

We now formally define the syntax of the fusion calculus, followed by three formulations of its operational semantics: one based on labelled transition systems, one based on unlabelled transition systems, and one symbolic semantics where the conditions for transitions are included in the label.

3.1 Syntax

We begin by presupposing an infinite set $\mathcal{N}$ of *names* ranged over by $u, v, \ldots, z$. Like in the π-calculus, names represent communication channels, which are also the values transmitted. We keep the notation $\tilde{x}$ for a (possibly empty) finite sequence $x_1 \cdots x_n$ of names. φ, ψ range over total equivalence relations over $\mathcal{N}$ (i.e. equivalence relations with $\mathrm{dom}(\varphi) = \mathcal{N}$) with only finitely many non-singular equivalence classes. If x and y are relate by φ we write $x \;\varphi\; y$. We write $\{\tilde{x} = \tilde{y}\}$ to mean the smallest such equivalence relation relating each x_i with y_i, and write $\mathbf{1}$ for the identity relation. As a consequence, a fusion written $\{x = x\}$ is the same as $\{y = y\}$, namely $\mathbf{1}$, and $\{x = y, x = z\}$ is the same as $\{x = y, y = z\}$.

Definition 14 *The* free actions*, ranged over by α, and the* agents*, ranged over by $P, Q, \ldots$, are defined by*

$\alpha ::=$	$u\tilde{x}$	(Input)	$P ::=$	$\mathbf{0}$	(Inaction)
	$\overline{u}\tilde{x}$	(Output)		$\alpha\,.\,Q$	(Prefix)
	φ	(Fusion)		$Q + R$	(Summation)
				$Q \mid R$	(Composition)
				$(x)Q$	(Scope)
				$[x = y]Q$	(Match)
				$[x \neq y]Q$	(Mismatch)
				$A(\tilde{x})$	(Identifier)

In the fusion calculus prefixes correspond exactly to the free actions defined above. An input action $u\tilde{x}$ means "input objects along the port u and replace $\tilde{x}$ with these objects." Contrary to the situation in the π-calculus this action does not bind x. In other words, the scope of $\tilde{x}$ is not bounded by the action. The output action $\overline{u}\tilde{x}$ is familiar from the π-calculus and means "output the objects $\tilde{x}$ along the port u.

The input and output actions above are collectively called *free communication* actions. In these, the names $\tilde{x}$ are the *objects* of the action, and the name u

is the *subject*. We write a to stand for either u or $\overline{u}$, thus $a\tilde{x}$ is the general form of a communication action. Fusion actions have neither subject nor objects.

Prefixing an agent Q means that the agent must perform the action α before acting like Q. Performing a fusion here represents an obligation to treat all related names as identical. Like in the π-calculus, Summation is alternative choice and Composition lets agents act in parallel, and Match, Mismatch and Identifiers work exactly as in the π-calculus. We write $\sum_{i \in I} P_i$ for finite general summation, $P_1 + \cdots + P_n$.

The Scope $(x)Q$ limits the scope of x to Q; no communication action of $(x)Q$ can have x as its subject, and fusion effects with respect to x are limited to Q. Restriction and input binding of the π-calculus can be recovered as special cases of Scope, as we will see in Section 3.2.1.

The name x is said to be *bound* in $(x)P$. We write $(\tilde{x})P$ for $(x_1)\cdots(x_n)P$. The *free names* in P, denoted $\mathrm{fn}(P)$, are the names in P with a non-bound occurrence; here the names occurring in the fusion φ is defined to be the names in the non-singular equivalence classes, i.e., in the relation $\varphi - \mathbf{1}$. Notice that since the syntax $\{x = x\}$ means $\mathbf{1}$, the free names of that fusion do not include x, but is the empty set. As usual we will not distinguish between alpha-variants of agents, i.e., agents differing only in the choice of bound names.

The action of a transition may be free or bound:

Definition 15 *The* actions*, ranged over by γ, consist of the fusion actions and of communication actions of the form $(z_1)\cdots(z_n)a\tilde{x}$ (written $(\tilde{z})a\tilde{x}$), where $n \geq 0$ and all names in $\tilde{z}$ are also in $\tilde{x}$. If $n > 0$ we say it is a* bound *action.*

Note that there are no bound fusion actions. In the communication actions above, $\tilde{z}$ are the *bound objects* and the names in $\tilde{x}$ that are not in $\tilde{z}$ are the *free objects*. Free actions have no bound objects. We further write $\mathrm{n}(\gamma)$ to mean all names occurring in γ (i.e., also including the subject of communication actions and the names in non-singular equivalence classes in fusion actions).

For convenience we define $\varphi \backslash z$ to mean $\varphi \cap (\mathcal{N} - \{z\})^2 \cup \{(z,z)\}$, i.e., the equivalence relation φ with all references to z removed (except for the identity). For example, $\{x = z, z = y\} \backslash z = \{x = y\}$, and $\{x = y\} \backslash y = \mathbf{1}$.

Following the presentation of the π-calculus, we now define a structural congruence which equates all agents we will never want to distinguish for any semantic reason, and then use this when giving the different versions of semantics. The structural congruence for the fusion calculus is just like the one for the π-calculus (Definition 2), with Scope in the place of Restriction.

Definition 16 *The* structural congruence*, $\equiv$, between agents is the least congruence satisfying the abelian monoid laws for Summation and Composition (associativity, commutativity and $\mathbf{0}$ as identity), and the scope laws*

$$(x)\mathbf{0} \equiv \mathbf{0}, \quad (x)(y)P \equiv (y)(x)P, \quad (x)(P+Q) \equiv (x)P + (x)Q,$$
$$(x)MP \equiv M(x)P \textit{ if } x \notin \mathrm{n}(M)$$

and also the scope extension *law $P \mid (z)Q \equiv (z)(P \mid Q)$ where $z \notin \mathrm{fn}(P)$, and the law for process identifiers: $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ if $A(\tilde{x}) \stackrel{def}{=} P$.*

$$\text{PREF}\ \frac{-}{\alpha\,.\,P \xrightarrow{\alpha} P} \qquad \text{SUM}\ \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\text{PAR}\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \text{COM}\ \frac{P \xrightarrow{u\tilde{x}} P',\ \ Q \xrightarrow{\overline{u}\tilde{y}} Q',\ \ |\tilde{x}| = |\tilde{y}|}{P \mid Q \xrightarrow{\{\tilde{x}=\tilde{y}\}} P' \mid Q'}$$

$$\text{SCOPE}\ \frac{P \xrightarrow{\varphi} P',\ z\,\varphi\,x,\ z \neq x}{(z)P \xrightarrow{\varphi\backslash z} P'\{x/z\}} \qquad \text{PASS}\ \frac{P \xrightarrow{\alpha} P',\ z \notin \mathrm{n}(\alpha)}{(z)P \xrightarrow{\alpha} (z)P'}$$

$$\text{OPEN}\ \frac{P \xrightarrow{(\tilde{y})\,a\,\tilde{x}} P',\ z \in \tilde{x} - \tilde{y},\ a \notin \{z, \overline{z}\}}{(z)P \xrightarrow{(z\tilde{y})\,a\,\tilde{x}} P'}$$

$$\text{MATCH}\ \frac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'} \qquad \text{MISMATCH}\ \frac{P \xrightarrow{\alpha} P',\ x \neq y}{[x \neq y]P \xrightarrow{\alpha} P'}$$

Table 3.1: Laws of action for the Fusion Calculus.

3.2 Labelled Transition Semantics

Definition 17 *The family of* transitions $P \xrightarrow{\gamma} Q$ *is the least family satisfying the laws in Table 3.1. In this definition structurally equivalent agents are considered the same, i.e., if* $P \equiv P'$ *and* $Q \equiv Q'$ *and* $P \xrightarrow{\gamma} Q$ *then also* $P' \xrightarrow{\gamma} Q'$.

Use of the SCOPE rule entails a substitution of the scoped name z for a nondeterministically chosen name x related to it by φ. For the purpose of the equivalences defined in Chapter 4 it will not matter which such x replaces z. The only rule dealing with bound actions is OPEN. This simplifies proofs by induction on the depth of inference of transitions. Using structural congruence, pulling the relevant scope to the top level, we can still infer e.g. $P \mid (x)ayx\,.\,Q \xrightarrow{(x)ayx} P \mid Q$ using PREF, PAR and OPEN (provided $x \notin \mathrm{fn}(P)$, otherwise an alpha-conversion is necessary). It is clear that for the purpose of the semantics, fusion prefixes could be regarded as derived forms since $\{\tilde{y} = \tilde{z}\}\,.\,P$ has exactly the same transitions as $(u)(\overline{u}\tilde{y}\,.\,\mathbf{0} \mid u\tilde{z}\,.\,P)$ when $u \notin \mathrm{fn}(P)$. (In the same way the τ prefix could be regarded as derived in CCS and in the π-calculus.)

It is illuminating to consider the COM and SCOPE rules in particular. COM results in a fusion action, signifying a requirement to fuse names made equal by the equivalence relation. The SCOPE rule effectuates the fusion by performing an appropriate substitution, and changes the action by removing the scoped name from the equivalence relation.

The following examples should be noted on how scoping delimits the effects of fusion:

$$\begin{array}{rcl}
P \mid \overline{u}y\,.\,Q \mid (z)uz\,.\,R & \xrightarrow{\mathbf{1}} & P \mid Q \mid R\{y/z\} \\
P \mid (z)(\overline{u}y\,.\,Q \mid uz\,.\,R) & \xrightarrow{\mathbf{1}} & P \mid (Q \mid R)\{y/z\} \\
(z)(P \mid \overline{u}y\,.\,Q \mid uz\,.\,R) & \xrightarrow{\mathbf{1}} & (P \mid Q \mid R)\{y/z\} \\
(z)(P \mid \overline{u}xy\,.\,Q \mid uzw\,.\,R) & \xrightarrow{\{y=w\}} & (P \mid Q \mid R)\{x/z\}
\end{array}$$

Let us see in detail how the last transition is derived:

$\overline{u}xy\,.\,Q$	$\xrightarrow{\overline{u}xy}$	Q	by PREF
$u\,zw\,.\,R$	$\xrightarrow{u\,zw}$	R	by PREF
$\overline{u}xy\,.\,Q \mid uzw\,.\,R$	$\xrightarrow{\{x=z,y=w\}}$	$Q \mid R$	by COM and the above two lines
$P \mid \overline{u}xy\,.\,Q \mid uzw\,.\,R$	$\xrightarrow{\{x=z,y=w\}}$	$P \mid Q \mid R$	by PAR and previous line
$(z)(P \mid \overline{u}xy\,.\,Q \mid uzw\,.\,R)$	$\xrightarrow{\{y=w\}}$	$(P \mid Q \mid R)\{x/z\}$	by SCOPE and previous line

Notice that we can derive $(x)(P \mid \overline{u}xy\,.\,Q \mid uzw\,.\,R) \xrightarrow{\{y=w\}} (P \mid Q \mid R)\{z/x\}$ exactly the same way.

The gain in expressiveness over the π-calculus lies in this possibility to decouple the input action from the binding of the input object. As an example of the expressiveness it is now easy to define a *delayed input* construct $u(x){:}\,P$, which is input without the temporal precedence. Here P may execute and can also at any time receive something for x along u. The binding of x extends into P, which is the reason this is different from the π-calculus agent $u(x)\,.\,\mathbf{0} \mid P$. This operator would have complicated effects in the π-calculus: an output of a name bound by a delayed input would make the delayed input be included in the action in the same way as for restriction:

$$u(x){:}\,(\overline{v}x\,.\,P \mid Q) \xrightarrow{u(x):\overline{v}x} (P \mid Q)$$

And if u is restricted, the action gets even more complex:

$$(\boldsymbol{\nu})u(x){:}\,(\overline{v}x\,.\,P \mid Q) \xrightarrow{(\boldsymbol{\nu}u)u(x):\overline{v}x} (P \mid Q)$$

Generalizing these sketches to a polyadic setting gives a very cumbersome semantics. In the fusion calculus, however, the (polyadic) delayed input can be simply defined as $(\tilde{x})(u\tilde{x} \mid P)$; here it is possible for P to execute before $u\tilde{x}$ interacts with an action of kind $\overline{u}\tilde{v}$, fusing $\tilde{x}$ in P with $\tilde{v}$. There are no additional complications by bound outputs of names in $\tilde{x}$.

Another example of expressiveness is the polyadic input of the same name twice: $(z)u\,zz \mid P$. Here any two names emitted by P along u will be fused, for example

$$(z)u\,zz \mid \overline{u}\,vw\,.\,P \xrightarrow{\{v=w\}} P$$

In Section 4.1 we will see that $(z)u\,zz$ can not be encoded compositionally in the π-calculus.

3.2.1 The π-Calculus as a Subcalculus

The polyadic π-calculus is isomorphic to a subcalculus of the fusion calculus, formed by omitting fusion prefixes and by requiring all input prefixes $u\tilde{x} \,.\, P$ to occur immediately under a scope $(\tilde{x})$ of the objects in the input prefix; the objects must be pairwise distinct. This way, no fusion actions except $\mathbf{1}$ are possible, since the objects are always sufficiently scoped. Call this subcalculus f_π. The isomorphism $\leftrightarrow$ between f_π and π is simply the homomorphic extension of

$$\begin{array}{rcll} (\tilde{x})u\tilde{x} \,.\, P & \leftrightarrow & u(\tilde{x}) \,.\, P & \\ (z)P & \leftrightarrow & (\boldsymbol{\nu} z)P & \text{if } P \text{ is not a free input prefix} \end{array}$$

The rationale behind this is as follows. Consider a scope $(x)Q$ in f_π. If it is immediately over an input prefix, $Q = ux \,.\, P$, then before anything happens in P something must be received along u to replace x in P, precisely as the effect of a π-calculus input prefix $u(x) \,.\, P$. On the other hand, if it is not immediately over an input prefix, $Q \neq ux \,.\, P$, then there cannot be *any* input prefix inside Q for that particular x. Because of the syntactic restrictions in f_π, a free input ux not at the top level of Q must be enclosed by another scope (x), making the object different from the x in $(x)Q$. So this x cannot occur in an input and there are no fusion prefixes, therefore x will remain distinct from other names throughout execution of the agent. This is precisely the effect of $(\boldsymbol{\nu} x)Q$ in the π-calculus.

We identify actions of the π-calculus and of f_π as follows: $u(\tilde{x}) = (\tilde{x})u\tilde{x}$, $(\boldsymbol{\nu}\tilde{z})\overline{u}\,\tilde{y} = (\tilde{z})\overline{u}\,\tilde{y}$ and $\tau = \mathbf{1}$. Decorating the transition arrows with their respective calculus, we can then formalize the result as

Theorem 3.1 *For any π-calculus agent P and an f_π-calculus agent Q such that $P \leftrightarrow Q$,*

- *$P \xrightarrow{\gamma}_\pi P'$ implies $Q \xrightarrow{\gamma}_f Q'$ with $P' \leftrightarrow Q'$, and*
- *$Q \xrightarrow{\gamma}_f Q'$ implies $P \xrightarrow{\gamma}_\pi P'$ with $P' \leftrightarrow Q'$.*

PROOF: By induction on the depth of inference of the transition of P and Q, respectively.

We show only the case for πCOM, where $P \equiv P_1 \mid P_2$, $P_1 \xrightarrow{u(\tilde{x})}_\pi P_1'$, $P_2 \xrightarrow{(\boldsymbol{\nu}\tilde{z})\overline{u}\tilde{y}}_\pi P_2'$, and $P \xrightarrow{\tau}_\pi (\tilde{z})(P_1'\{\tilde{y}/\tilde{x}\} \mid P_2')$. Then $Q \equiv Q_1 \mid Q_2$ and by induction $Q_1 \xrightarrow{(\tilde{x})u\tilde{x}}_f Q_1'$, $Q_2 \xrightarrow{(\tilde{z})\overline{u}\tilde{y}}_f Q_2'$ and we want to show that $Q \xrightarrow{\mathbf{1}}_f (\tilde{z})(Q_1'\{\tilde{y}/\tilde{x}\} \mid Q_2')$. We observe that if a fusion agent $R \xrightarrow{(\tilde{x})a\tilde{y}}_f R'$, then by scope extension $R \equiv (\tilde{x})R_0$ and $R_0 \xrightarrow{a\tilde{y}}_f R'$, and we use this to write Q as $(\tilde{z})(\tilde{x})(Q_1'' \mid Q_2'')$, where $Q_1'' \xrightarrow{u\tilde{x}}_f Q_1'$ and $Q_2'' \xrightarrow{\overline{u}\tilde{y}}_f Q_2'$. Then $Q'' \equiv Q_1'' \mid Q_2'' \xrightarrow{\{\tilde{x}=\tilde{y}\}}_f Q_1' \mid Q_2'$ by COM, and $(\tilde{x})Q'' \xrightarrow{\mathbf{1}}_f (Q_1' \mid Q_2')\{\tilde{y}/\tilde{x}\}$ by SCOPE, but by alpha conversion $\tilde{x} \cap \mathrm{fn}(Q_2) = \emptyset$, so the substitution has no effect on Q_2'. Finally $(\tilde{z})(\tilde{x})Q'' \equiv Q \xrightarrow{\mathbf{1}}_f (Q_1'\{\tilde{y}/\tilde{x}\} \mid Q_2')$ as we wanted. $\square$

This existence of this isomorphism means that results for the π-calculus carry over to f_π and vice versa. This fact will be used, e.g., in Chapter 7.

The πI-calculus by Sangiorgi [San96b], where all output prefixes $\overline{u}\tilde{x}$ occur immediately below a restriction $(\boldsymbol{\nu}\tilde{x})$ of the objects, is isomorphic to the subcalculus of f_π requiring that *all* prefixes — not merely input prefixes — occur under a scope of the object. As Sangiorgi has remarked the πI-calculus has a pleasing symmetry between input and output actions. The fusion calculus has the same kind of symmetry, achieved by extending rather than restricting the π-calculus, as implied by the following theorem where $\overline{P}$ $(\overline{\gamma})$ stands for P (γ) with all input/output polarities inverted:

Theorem 3.2 $P \xrightarrow{\gamma} Q$ *iff* $\overline{P} \xrightarrow{\overline{\gamma}} \overline{Q}$.
PROOF: By induction on the depth of deduction, using the fact that no rule refers to the polarities except the COM rule, which is symmetric. □

This result shortens many proofs, since we only have to give the proof for one polarity — the other follows by Theorem 3.2.

3.3 Reduction Semantics

We shall now provide a *reduction* semantics for the fusion calculus. The reduction semantics can in some respects be easier to comprehend, since it avoids the new concept of fusion, and it facilitates the comparison with rewrite systems such as the λ-calculus, ρ-calculus and γ-calculus in Chapter 6. Like the π-calculus, the reduction semantics for the fusion calculus is only defined for the subcalculus given by the BNF equation

$$P ::= \mathbf{0} \;\Big|\; \sum \tilde{M}_i\alpha_i \,.\, P_i \;\Big|\; P_1 \mid P_2 \;\Big|\; (x)P \;\Big|\; A(\tilde{y})$$

We call this subcalculus the *guarded fusion calculus.*

Definition 18 *A name u is* fresh *wrt P is $u \notin \mathrm{fn}(P)$. We say that u* is fresh *when the agent(s) it is fresh with respect to is clear from context.*

Definition 19 *A substitution σ* agrees with *the fusion φ if $\forall x, y : x \,\varphi\, y \Leftrightarrow \sigma(x) = \sigma(y)$.*

The reduction relation of the fusion calculus is the least relation satisfying the rules in Table 3.2. Here and in the following we use $\mathrm{dom}(\sigma) = \{u : \sigma(u) \neq u\}$ and $\mathrm{ran}(\sigma) = \{\sigma(u) : \sigma(u) \neq u\}$. The reductions of an appropriately scoped fusion prefix are obtained by viewing a fusion prefix $\{\tilde{x} = \tilde{y}\} \,.\, P$ as defined by $(u)(\overline{u}\,\tilde{x} \mid u\,\tilde{y} \,.\, P)$ where u is fresh. We sometimes write $P \not\rightarrow$ for $\neg(\exists P' : P \rightarrow P')$.

For the purpose of the following lemma, we define $\varphi \restriction \tilde{z} = (\varphi \cap (\tilde{z} \times \tilde{z})) \cup \mathbf{1}$, i.e., the fusion φ with only references to names in $\tilde{z}$ (and identity). For example, $\{x = y, y = z\} \restriction \{x, z\} = \{x = z\}$.

$$(\tilde{z})\big(R \mid (\cdots + \tilde{M} u\, \tilde{x}\,.\,P) \mid (\tilde{N} \overline{u}\, \tilde{y}\,.\,Q + \cdots)\big) \rightarrow (R \mid P \mid Q)\sigma$$
$$\text{if} \quad |\tilde{x}| = |\tilde{y}|, \tilde{M} \Leftrightarrow \tilde{N} \Leftrightarrow \mathsf{true},$$
$$\sigma \text{ agrees with } \{\tilde{x} = \tilde{y}\}, \mathrm{ran}(\sigma) \cap \tilde{z} = \emptyset, \text{ and } \mathrm{dom}(\sigma) = \tilde{z}$$

$$\frac{P \rightarrow P'}{[x = x]P \rightarrow P'} \qquad \frac{P \rightarrow P',\ x \neq y}{[x \neq y]P \rightarrow P'} \qquad \frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q}$$

$$\frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad \frac{P \rightarrow P'}{P + Q \rightarrow P'} \qquad \frac{P \rightarrow P'}{(x)P \rightarrow (x)P'}$$

Table 3.2: Reduction rules for the Fusion Calculus.

Lemma 3.3 *If $P \xrightarrow{\varphi} Q$, then $P \equiv (\tilde{z})\tilde{M}_1(R \mid (\cdots + \tilde{M}_2 u\tilde{x}\,.\,P_1) \mid (\tilde{M}_3\overline{u}\,\tilde{y}\,.\,P_2 + \cdots))$ and $Q \equiv (\tilde{w})(R \mid P_1 \mid P_2)\sigma$, where $\tilde{M}_1 \Leftrightarrow \tilde{M}_2 \Leftrightarrow \tilde{M}_3 \Leftrightarrow \mathsf{true}$, $|\tilde{x}| = |\tilde{y}|$, $\varphi = \{\tilde{x} = \tilde{y}\}\backslash\tilde{z}$, σ agrees with $\{\tilde{x} = \tilde{y}\} \restriction \tilde{z}$, $\mathrm{dom}(\sigma) \cap \tilde{z} = \emptyset$, and $\tilde{w} = \tilde{z} - \mathrm{ran}(\sigma)$.*

PROOF: By induction on the depth of inference, using the fact that if $P \xrightarrow{a\tilde{x}} Q$ then $P \equiv (\tilde{w})\tilde{M}(R \mid (\cdots + \tilde{N}a\tilde{x}\,.\,P'))$ and $Q \equiv (\tilde{w})(R \mid P')$. □

Proposition 3.4 *$P \rightarrow Q$ iff $P \xrightarrow{1} Q$.*

PROOF: The implication to the right is by routine induction over inference depth. The implication to the left is by Lemma 3.3. □

3.4 Symbolic Semantics

In this section we present a symbolic semantics for the fusion calculus, along the lines of symbolic semantics for the π-calculus [San96c, Lin94, Lin95a]. Symbolic semantics are used to give efficient characterizations of bisimulation equivalences for value-passing calculi: Hennessy and Lin [HL95] did this for value-passing CCS, Sangiorgi [San96c, published as a technical report in 1993] gave symbolic semantics for the π-calculus and a symbolic characterization of open bisimulation equivalence, which was later used by the present author and Faron Moller [VM94] in the Mobility Workbench, an automated verification tool for mobile processes (see Chapter 7). Lin [Lin94, Lin95a] used symbolic transition systems to get efficient characterizations of late and early bisimulation equivalences in the π-calculus. In Table 3.3 we present the symbolic transition system.

Definition 20 *We say that a match sequence is* consistent *if $\tilde{M}\sigma \Leftrightarrow \mathsf{true}$ for some σ. $\tilde{M}$ is* consistent with *$\tilde{N}$ if $\tilde{M}\tilde{N}$ is consistent, and $\tilde{M}$ is consistent with φ if it is consistent with a sequence of matches corresponding to the non-singular equivalence classes of φ. We write $\tilde{M}^+$ for the match part of $\tilde{M}$, and $\tilde{M}^-$ for the mismatch part, and $\tilde{M}\backslash z$ for a match sequence such that $\tilde{M}\backslash z \Rightarrow [x \Box y]$ iff $\tilde{M} \Rightarrow [x \Box y]$ and $z \notin \{x, y\}$, for $\Box \in \{=, \neq\}$, i.e., an equivalent match sequence but without mention of z. We write $\emptyset$ for the empty match sequence.*

$$\text{S-PREF}\quad \frac{-}{\alpha\,.\,P \xmapsto{\emptyset,\alpha} P} \qquad\qquad \text{S-SUM}\quad \frac{P \xmapsto{\tilde{M},\alpha} P'}{P + Q \xmapsto{\tilde{M},\alpha} P'}$$

$$\text{S-PAR}\quad \frac{P \xmapsto{\tilde{M},\alpha} P'}{P \mid Q \xmapsto{\tilde{M},\alpha} P' \mid Q}$$

$$\text{S-COM}\quad \frac{P \xmapsto{\tilde{M},u\tilde{x}} P',\quad Q \xmapsto{\tilde{N},\overline{v}\tilde{y}} Q',\quad |\tilde{x}| = |\tilde{y}|,\ \tilde{M}\tilde{N}[u=v] \text{ consistent}}{P \mid Q \xmapsto{\tilde{M}\tilde{N}[u=v],\{\tilde{x}=\tilde{y}\}} P' \mid Q'}$$

$$\text{S-SCOPE}\quad \frac{P \xmapsto{\tilde{M},\varphi} P',\ z\,\varphi\,x,\ z \neq x,\ \forall y \neq z : \neg(\tilde{M} \Rightarrow [z=y])}{(z)P \xmapsto{\tilde{M}\backslash z,\varphi\backslash z} P'\{x/z\}}$$

$$\text{S-PASS}\quad \frac{P \xmapsto{\tilde{M},\alpha} P',\ z \notin \mathrm{n}(\tilde{M}^{+},\alpha)}{(z)P \xmapsto{\tilde{M}\backslash z,\alpha} (z)P'}$$

$$\text{S-OPEN}\quad \frac{P \xmapsto{\tilde{M},(\tilde{y})a\,\tilde{x}} P',\ z \in \tilde{x} - \tilde{y},\ a \notin \{z,\overline{z}\},\ z \notin \mathrm{n}(\tilde{M}^{+})}{(z)P \xmapsto{\tilde{M}\backslash z,(z\tilde{y})a\,\tilde{x}} P'}$$

$$\text{S-MATCH}\quad \frac{P \xmapsto{\tilde{M},\alpha} P',\ \tilde{M}[x=y] \text{ consistent}}{[x=y]P \xmapsto{\tilde{M}[x=y],\alpha} P'}$$

$$\text{S-MISMATCH}\quad \frac{P \xmapsto{\tilde{M},\alpha} P',\ \tilde{M}[x\neq y] \text{ consistent}}{[x\neq y]P \xmapsto{\tilde{M}[x\neq y],\alpha} P'}$$

Table 3.3: Symbolic transition system for the Fusion Calculus.

For example, $[x = y][y = z]\backslash y \Leftrightarrow [x = z]$.

A transition in the symbolic transition system is on the form $P \xmapsto{\tilde{M},\gamma} Q$, where $\tilde{M}$ is the enabling condition of the action γ in the sense that $\tilde{M}$ represents the equalities and inequalities a minimal substitution $\sigma_{\tilde{M}}$ must make true in order for $P\sigma_{\tilde{M}}$ to perform the corresponding action in the original labelled transition system. For example,

$$[x \neq y]u\,.\,P \xmapsto{[x\neq y],u} P \qquad \text{and} \qquad [x \neq y]u\,.\,P \mid \overline{v}\,.\,Q \xmapsto{[x\neq y,u=v],\mathbf{1}} P \mid Q$$

We will see the use of this formulation of the semantics in Section 4.4, where we also show how the symbolic transitions correspond to the non-symbolic labelled transitions.

3.5 Conclusion

The new concepts of fusion and scope were defined and three variants of the operational semantics of the fusion calculus were presented. It is perhaps of interest to see how our precursor of the fusion calculus, the *update calculus* [PV97], differs. The update calculus is a monadic calculus; only one object can be used in input and output actions. Interactions give rise not to a symmetric fusion action, but to a directed *update action* $[y/x]$, signifying that x should everywhere be replaced by y.

$$\text{UP}\textsc{com}\ \frac{P \xrightarrow{ux} P',\ \ Q \xrightarrow{\overline{u}y} Q'}{P \mid Q \xrightarrow{[y/x]} P' \mid Q'}$$

The scope operator effectuates the update operation just like it does for fusion actions in the fusion calculus.

$$\text{UP}\textsc{scope}\ \frac{P \xrightarrow{[y/x]} P',\ x \neq y}{(x)P \xrightarrow{\mathbf{1}} P'\{y/x\}}$$

The update calculus is complicated by the existence of bound update actions $(y)[y/x]$, and because of the explicit direction of the update actions this calculus is difficult to generalize to polyadic interactions. Consider

$$u\,zz \mid \overline{u}\,xy\,.\,P \xrightarrow{[x/z,y/z]} P$$

What would the effect of this update action be? As we have seen, the corresponding situation in the fusion calculus is unproblematic.

The idea to separate scope from input has occurred independently to Fu in encodings of the λ-calculus and of cut elimination in linear logic [Fu97]. Fu's semantics is significantly more complex; it has no actions corresponding to fusions but three types of input and output actions.

Similarities to fusion actions can be seen in an early version of the calculus underlying the Oz programming language [HSW93], where an equational constraint could arise from a special type of communication:

$$x!y \wedge z?y \rightarrow x = z$$

These "constraint communication actions" are completely asynchronous, and separate from the other forms of interaction such as input-output interactions between agents.

Independently of our work, Honda [Hon97] has in the action calculus framework described a calculus with concepts similar to our fusion actions; here a composition can result in what Honda calls an "open substitution":

$$ab \mid \overline{a}c \;\rightarrow\; [b = c]$$

The motivation for this calculus is to study the core part of name passing; there is no synchronization primitives, and no operators for summation or matching, which are necessary, e.g., for algebraic theories of processes.

Chapter 4

Bisimulation Equivalences

In the previous chapter we defined three different operational semantics of the fusion calculus, so now we certainly know what behaviour fusion agents have — or do we? What should be considered equivalent behaviour? When can we replace one agent for another in a system of agents without changing the behaviour of the whole system? Until we know the answers to such questions, we can not really tell what our system does.

In this chapter we will define behavioural equivalences based on bisimulation for each of the three types of operational semantics given in the previous chapter. We will show that they coincide, i.e., that regardless of which operational semantics we choose, the respective equivalences find exactly the same agents equal.

As we saw in Section 2.2, in the π-calculus there are three major varieties: *early*, *late* [MPW92], and *open* [San96c] bisimulation, which differ in the use of universal quantification over names, each leading to a different congruence.

In the fusion calculus this diversity is mercifully absent; there is only one sensible bisimulation congruence. The reason is that the fusion actions allow us to formulate more powerful contexts than what is possible in the π-calculus. Fusion actions can enforce a global substitution of *any* name, whereas in the π-calculus no action can enforce a substitution on a name bound by a restriction. Therefore a congruence for the fusion calculus must be closed under arbitrary substitutions in a way which will be made precise below.

We also consider weak, or *observation*, equivalences, where internal behaviour of agents is ignored. While these are well understood for process calculi such as CCS and the π-calculus, it is not trivial to see how fusion actions should be treated. We consider two alternatives, but favour one, based on its correspondence to the weak *barbed* congruence for the fusion calculus. The barbed congruence is based on the simple concept of *barbed bisimulation*, which is reduction bisimulation with an added capability of observing interaction possibilities.

Finally, we investigate *symbolic* bisimulation based on the symbolic semantics of Chapter 3. Symbolic bisimulation equivalences do not use universal quantification over substitutions, and can therefore be implemented efficiently in tools. We will see how this is done in Chapter 7. As a spin-off effect of this work, we can define a variant of open bisimulation equivalence for the π-calculus, which behaves well in the presence of the mismatch operator.

We begin by giving a set of definitions used throughout this thesis.

Definition 21 *A substitution σ is* idempotent *if $\sigma\sigma = \sigma$.*

The intuition behind this definition is that all elements in $\mathrm{ran}(\sigma)$ are mapped to themselves by σ. It follows that the composition of two idempotent substitutions is idempotent.

Definition 22 *A* substitutive effect *of a fusion φ is an idempotent substitution σ agreeing with φ (i.e., σ sends all members of each equivalence class of φ to one representative in the class). The only substitutive effect of a communication action is the identity substitution.*

For example, the substitutive effects of $\{x = y\}$ are $\{x/y\}$ and $\{y/x\}$. Note that not all substitutions which agree with φ are substitutive effects of φ. For example any injective substitution agrees with $\mathbf{1}$, but the only substitutive effect of $\mathbf{1}$ is the identity substitution.

Definition 23 *A substitution σ* agrees with *a match sequence $\tilde{M}$, and $\tilde{M}$ agrees with σ, if for all x, y which appear in $\tilde{M}$ it holds that $\sigma(x) = \sigma(y)$ iff $\tilde{M} \Rightarrow [x = y]$.*

Definition 24 *A substitution σ* satisfies *a match sequence $\tilde{M}$, written $\sigma \models \tilde{M}$, if for all x, y, $\tilde{M} \Rightarrow [x = y]$ implies $\sigma(x) = \sigma(y)$ and $\tilde{M} \Rightarrow [x \neq y]$ implies $\sigma(x) \neq \sigma(y)$.*

Definition 25 *A substitution σ* respects *a match sequence $\tilde{M}$ if $\tilde{M} \Rightarrow [x \neq y]$ implies $\sigma(x) \neq \sigma(y)$, for all x, y.*

It may be helpful to relate the concepts of *agree with*, *satisfies* and *respects*: "σ agrees with $\tilde{M}$" means σ makes exactly those names equal which $\tilde{M}$ tests for equality; "σ satisfies $\tilde{M}$" means σ may make more names equal as long as it does not make $\tilde{M}$ inconsistent; "σ respects $\tilde{M}$" only means that σ does not make $\tilde{M}$ inconsistent.

4.1 Hyperequivalence

Definition 26 *A* bisimulation *is a binary symmetric relation $\mathcal{S}$ between agents such that $P\ \mathcal{S}\ Q$ implies:*
If $P \xrightarrow{\gamma} P'$ with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$
then $Q \xrightarrow{\gamma} Q'$ and $P'\sigma\ \mathcal{S}\ Q'\sigma$ for some substitutive effect σ of γ.
P is bisimilar to Q, written $P \dot{\sim} Q$, if $P\ \mathcal{S}\ Q$ for some bisimulation $\mathcal{S}$.

This definition differs from basic bisimulation (Definition 5) only by considering the substitutive effects of actions, i.e., in the treatment of fusion actions. A fusion $\{x = y\}$ represents an obligation to make x and y equal everywhere. Therefore, if γ above is such a fusion, it only makes sense to relate P' and Q' when a substitution $\{y/x\}$ or $\{x/y\}$ has been performed. Note that it does not matter which substitution we choose, since $P\{x/y\} \stackrel{\cdot}{\sim} Q\{x/y\}$ implies $P\{y/x\} \stackrel{\cdot}{\sim} Q\{y/x\}$, by the simple fact that $(P\{x/y\})\{y/x\} \equiv P\{y/x\}$ and that bisimilarity is closed under injective substitutions.

Consider the following examples:

$$\begin{array}{rcl} y \mid \overline{z} & \stackrel{\cdot}{\sim} & y \,.\, \overline{z} + \overline{z} \,.\, y \\ y \mid \overline{y} & \stackrel{\cdot}{\not\sim} & y \,.\, \overline{y} + \overline{y} \,.\, y \\ y \mid \overline{y} & \stackrel{\cdot}{\sim} & y \,.\, \overline{y} + \overline{y} \,.\, y + \mathbf{1} \\ \{y = z\} \,.\, (y \mid \overline{z}) & \stackrel{\cdot}{\sim} & \{y = z\} \,.\, (y \,.\, \overline{z} + \overline{z} \,.\, y + \mathbf{1}) \end{array}$$

The third example implies the fourth even though $y \mid \overline{z}$ is not bisimilar to $y \,.\, \overline{z} + \overline{z} \,.\, y + \mathbf{1}$. In fact, we also have

$$\{y = z\} \,.\, y \,.\, \mathbf{0} \stackrel{\cdot}{\sim} \{y = z\} \,.\, z \,.\, \mathbf{0}$$

Thus we argue that any sensible bisimulation-like equivalence must contain bisimilarity. But bisimilarity is not a congruence. An easy example is that $x \mid \overline{y}$ is bisimilar to $x \,.\, \overline{y} + \overline{y} \,.\, x$, but prefixing both with $\{x = y\}$ will result in non bisimilar agents. We therefore look for the largest congruence included in bisimilarity. This is achieved by closing the definition of bisimulation under arbitrary substitutions.

Definition 27 *A* hyperbisimulation *is a substitution closed bisimulation, i.e., a bisimulation $\mathcal{S}$ with the property that $P \; \mathcal{S} \; Q$ implies $P\sigma \; \mathcal{S} \; Q\sigma$ for any substitution σ. Two agents P and Q are* hyperequivalent*, written $P \sim Q$, if they are related by a hyperbisimulation.*

Bisimilarity is closed under injective substitutions, and any substitution can be defined as a composition of an idempotent substitution and an injective substitution. Therefore if only idempotent substitutions are considered in Definition 27, the resulting hyperequivalence will be the same. We will often use this fact implicitly in proofs to come.

Theorem 4.1 *Hyperequivalence is a congruence.*

PROOF: Through a case analysis over the operators in the calculus. Recursive definitions requires some care (similar to Proposition 4.14 in [Mil89], page 103), but all other cases are simple. □

Note that the special treatment of fusion actions in the definition of bisimulation is significant even though hyperbisimulation closes under all substitutions! The point is that $P' \; \mathcal{S} \; Q'$ in Definition 26 would be a too *strong* requirement if $\gamma = \{y = z\}$, since it would require bisimulation also under the substitutions which do not equate z and y.

It is interesting to relate hyperequivalence to open equivalence, which is a congruence in the π-calculus and hence also in f_π. For a definition of open we refer to Section 2.2, but even without the details of the definition a reader might appreciate the following arguments. Broadly speaking, for the purpose of hyperbisimulation, *all* names are considered universally quantified *after every transition*. That is, if we for a moment disregard the fusion actions (which are not part of the π-calculus), hyperbisimulation requires, if $P \mathcal{S} Q$ and $P \xrightarrow{\gamma} P'$, that $Q \xrightarrow{\gamma} Q'$ and $P'\sigma \mathcal{S} Q'\sigma$ *for all substitutions* σ. For late equivalence the corresponding requirement would be *for all substitutions* σ *involving a bound input name in* γ. For open equivalence the requirement would be *for all substitutions* σ *except those involving a bound output name in* γ.

In the π-calculus without the restriction operator the definitions of open and hyper would coincide. But with it hyper is strictly stronger, as evidenced by the f_π-calculus agents

$$(x\,y)\overline{u}\,xy\,.\,(x \mid \overline{y}) \qquad \text{and} \qquad (x\,y)\overline{u}\,xy\,.\,(x\,.\,\overline{y} + \overline{y}\,.\,x)$$

These are open equivalent but not hyperequivalent. To see this, after a bound output transition the agents become

$$x \mid \overline{y} \qquad \text{and} \qquad x\,.\,\overline{y} + \overline{y}\,.\,x$$

and these are not bisimilar under the substitution $\{x/y\}$. This substitution, which affects names previously emitted as bound objects, is not relevant for open equivalence since x and y have been objects of a bound output action. To see that this substitution *is* relevant to hyperequivalence, now put the two agents in parallel with $(z)u\,zz$ (note that this agent is not in f_π because the two input objects are not distinct). Then there is one more $\mathbf{1}$ transition from

$$(z)u\,zz \mid (x\,y)\overline{u}\,xy\,.\,(x \mid \overline{y})$$

than from

$$(z)u\,zz \mid (x\,y)\overline{u}\,xy\,.\,(x\,.\,\overline{y} + \overline{y}\,.\,x)$$

In essence, the parallel context $(z)u\,zz$ will enforce a fusion of any two names emitted along u. This demonstrates that open equivalence is not a congruence in the fusion calculus, and that this context cannot be encoded compositionally in the π-calculus. A similar argument holds for the monadic $(z)uz\,.\,uz$ in place of $(x)u\,zz$.

Even though open is not a congruence there might conceivably exist some other congruences larger than hyper, but our next theorem demonstrates that this is impossible. This result confirms that hyperequivalence is the only "interesting" bisimulation congruence.

Theorem 4.2 *Hyperequivalence is the largest congruence in bisimilarity.*

The rest of this section is devoted to the proof.

We construct a context which given a set of names N can perform any substitutions of names in N, and capture all bound names emitted on a name in N.

Let N be any finite set of names. Let a be a natural number and let $\mathcal{F}_N^a = \{s, r_x, e, d, \tilde{u} : x \in N, |\tilde{u}| \leq a\}$ consist of fresh names.

Define

$$\begin{array}{rcl} S_a(N) & \stackrel{def}{=} & \sum_{x,y\in N} s\,.\,\{x=y\}\,.\,d\,.\,S_a(N) \\ & + & \sum_{x\in N} r_x\,.\sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset}(\tilde{u})x\tilde{u}\,.\,\overline{e}\,\tilde{u}\,.\,S_a(N\tilde{u}) \\ & + & \sum_{x\in N} \overline{r_x}\,.\sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset}(\tilde{u})\overline{x}\,\tilde{u}\,.\,e\tilde{u}\,.\,S_a(N\tilde{u}) \end{array}$$

Formally s, r_x, e, d (for $x \in N$) are parameters of S_a, and since identifiers have a fixed arity, S_a above represents a set of definitions S_a^k where k is the arity of the argument N. In all contexts below these details are clear, so we choose to use the simple definition above. Note also that the sums prefixed by r_x and $\overline{r_x}$ are finite up to alpha-conversion.

We now give a few lemmas about this family of agents, which we use in the proof of Lemma 4.8, which is crucial for Theorem 4.2. The first lemma states that bisimulation in an $S_a(N)$ context is closed under substitution.

Lemma 4.3 *For a fixed N and a the maximal arity of communication actions in P and Q, assume $S_a(N) \mid P \stackrel{\centerdot}{\sim} S_a(N) \mid Q$. Then $(S_a(N) \mid P)\{x/y\} \stackrel{\centerdot}{\sim} (S_a(N) \mid Q)\{x/y\}$ for all $x, y \in N$.*

PROOF: Reducing the first term of $S_a(N)$,

$$S_a(N) \mid P \xrightarrow{s}\xrightarrow{\{x=y\}}\xrightarrow{d} S_a(N) \mid P$$

which can only be imitated by $S_a(N) \mid Q$ by reducing $S_a(N)$ the same way. So after the first action, $\{x=y\}\,.\,d\,.\,S_a(N) \mid P \stackrel{\centerdot}{\sim} \{x=y\}\,.\,d\,.\,S_a(N) \mid Q$, so $(d\,.\,S_a(N) \mid P)\{x/y\} \stackrel{\centerdot}{\sim} (d\,.\,S_a(N) \mid Q)\{x/y\}$ and finally $(S_a(N) \mid P)\{x/y\} \stackrel{\centerdot}{\sim} (S_a(N) \mid Q)\{x/y\}$. □

The next lemmas state that an $S_a(N)$ context can capture any bound names in actions, while not affecting free actions:

Lemma 4.4 *For any N and a the maximal arity of communication actions in P, and α free,* $\mathrm{subj}(\alpha) \notin \mathcal{F}_N^a$,

1. *if $P \xrightarrow{\alpha} P'$ then $S_a(N) \mid P \xrightarrow{\alpha} S_a(N) \mid P'$*
2. *if $S_a(N) \mid P \xrightarrow{\alpha} R$ then $P \xrightarrow{\alpha} P'$ and $R \equiv S_a(N) \mid P'$*

PROOF: 1. Trivial; 2. All subjects of initial actions of $S_a(N)$ are in $\mathcal{F}_N^a$. □

Lemma 4.5 *For any N, a the maximal arity of communication actions in P, $x \in N$ and $\tilde{w} \subseteq \tilde{y}$, it holds that $S_a(N) \mid P \xrightarrow{r_x}\xrightarrow{\mathbf{1}}\xrightarrow{(\tilde{w})\overline{e}\,\tilde{y}} R$ iff $R \equiv S_a(N\tilde{y}) \mid P'$ and $P \xrightarrow{(\tilde{w})\overline{x}\,\tilde{y}} P'$.*

Proof: There is only one way for $S_a(N) \mid P$ to perform an r_x action;

$$S_a(N) \mid P \xrightarrow{r_x} \sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset} (\tilde{u})x\,\tilde{u}\,.\,\overline{e}\,\tilde{u}\,.\,S_a(N\tilde{u}) \mid P \equiv R_0$$

and the only way for R_0 to complete the transition sequence is if $P \xrightarrow{(\tilde{w})\overline{x}\,\tilde{y}} P'$; then

$$R_0 \xrightarrow{\mathbf{1}} (\tilde{w})(\overline{e}\,\tilde{y}\,.\,S_a(N\tilde{y}) \mid P') \xrightarrow{(\tilde{w})\overline{e}\,\tilde{y}} S_a(N\tilde{y}) \mid P'$$

as required. □

In the Lemma below, we write $\mathrm{init}(P)$ for the set of names in subjects of initial actions of P, $\{\mathrm{n}(\mathrm{subj}(\gamma)) : P \xrightarrow{\gamma} P'\}$.

Lemma 4.6
For agents P, Q, R, R' such that $\mathrm{init}(R) \cap \mathrm{init}(R') = \emptyset$ and $\mathrm{init}(R, R') \cap \mathrm{fn}(P, Q) = \emptyset$, it holds that if $P \mid (R + R') \stackrel{\cdot}{\sim} Q \mid (R + R')$ then $P \mid R \stackrel{\cdot}{\sim} Q \mid R$.

Proof: By showing that $\mathcal{S} = \{(P \mid R, Q \mid R) : P \mid (R + R') \stackrel{\cdot}{\sim} Q \mid (R + R'), \mathrm{init}(R) \cap \mathrm{init}(R') = \emptyset, \mathrm{n}(\mathrm{init}(R, R')) \cap \mathrm{fn}(P, Q) = \emptyset\}$ is a bisimulation. □

Lemma 4.7 *If $y \notin N$ and $x \in N$ and $N = N' \cup \{y\}$, then $(S_a(N) \mid P)\{x/y\} \stackrel{\cdot}{\sim} (S_a(N) \mid Q)\{x/y\}$ implies $S_a(N') \mid P\{x/y\} \stackrel{\cdot}{\sim} S_a(N') \mid Q\{x/y\}$,*

Proof: It is easy to see that if $y \notin N$ and $x \in N$,

$$\begin{array}{rcll} S_a(N) & \equiv & & S_a(N') + \sum_{z \in N'} s\,.\,\{z = y\}\,.\,d\,.\,S_a(N) \\ & & + & r_y\,.\, \sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset} (\tilde{u})y\tilde{u}\,.\,\overline{e}\,\tilde{u}\,.\,S_a(N\tilde{u}) \\ & & + & \overline{r_y}\,.\, \sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset} (\tilde{u})\overline{y}\tilde{u}\,.\,e\,\tilde{u}\,.\,S_a(N\tilde{u}) \end{array}$$

Applying the substitution $\{x/y\}$ on this equation, we get

$$\begin{array}{rcll} S_a(N)\{x/y\} & \equiv & & S_a(N') + \sum_{z \in N'} s\,.\,\{z = x\}\,.\,d\,.\,S_a(N)\{x/y\} \\ & & + & r_y\,.\, \sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset} (\tilde{u})x\tilde{u}\,.\,\overline{e}\,\tilde{u}\,.\,S_a(N\tilde{u})\{x/y\} \\ & & + & \overline{r_y}\,.\, \sum_{|\tilde{u}|<a,\tilde{u}\cap N=\emptyset} (\tilde{u})\overline{x}\tilde{u}\,.\,e\,\tilde{u}\,.\,S_a(N\tilde{u})\{x/y\} \end{array}$$

where the summands prefixed by s are already in $S_a(N')$, and can be removed up to structural congruence. Since $\{r_y, \overline{r_y}\} \cap \mathrm{init}(S_a(N')) = \emptyset$ and $r_y \notin \mathrm{fn}(P, Q)$, we use Lemma 4.6 to prove that indeed the implication of this lemma holds. □

The following Lemma states that if two agents are bisimilar in an $S_a(N)$ context, then they must also be hyperequivalent:

Lemma 4.8
$\mathcal{S} \equiv \{(P,Q) : a = \mathrm{am}(P,Q), \forall N : \mathrm{fn}(P,Q) \subseteq N : S_a(N) \mid P \dot{\sim} S_a(N) \mid Q\}$ *is a hyperbisimulation, where* $\mathrm{am}(P,Q)$ *is the maximal arity of communication actions in* P *or* Q*.*
PROOF:

1. $\mathcal{S}$ is closed under substitution.
   If $(P,Q) \in \mathcal{S}$ then $(S_a(N) \mid P) \dot{\sim} (S_a(N) \mid Q)$, and by Lemma 4.3 we have $(S_a(N) \mid P)\{x/y\} \dot{\sim} (S_a(N) \mid Q)\{x/y\}$, and by Lemma 4.7 we have $S_a(N') \mid P\{x/y\} \dot{\sim} S_a(N') \mid Q\{x/y\}$ where $N = N' \cup \{y\}$, and since $y \notin \mathrm{fn}(P\{x/y\}, Q\{x/y\})$, it holds that $(P\{x/y\}, Q\{x/y\}) \in \mathcal{S}$.

2. $\mathcal{S}$ is a bisimulation. If $(P,Q) \in \mathcal{S}$, then $S_a(N) \mid P \dot{\sim} S_a(N) \mid Q$ for N s.t. $\mathrm{fn}(P,Q) \subseteq N$. Now, suppose $P \xrightarrow{\gamma} P'$.

   (a) If γ is free, then by Lemma 4.4 $S_a(N) \mid P \xrightarrow{\gamma} S_a(N) \mid P'$, and then $S_a(N) \mid Q \xrightarrow{\gamma} Q''$ and by Lemma 4.4 $Q'' \equiv S_a(N) \mid Q'$ and $Q \xrightarrow{\gamma} Q'$, and $S_a(N) \mid P' \dot{\sim} S_a(N) \mid Q'$, so $(P', Q') \in \mathcal{S}$.

   (b) If $\gamma = (\tilde{w})x\tilde{u}$ where all names in $\tilde{w}$ are in $\tilde{u}$, then by Lemma 4.5, $S_a(N) \mid P \xrightarrow{\overline{r_x}} \xrightarrow{\mathbf{1}} \xrightarrow{(\tilde{w})\overline{e}\tilde{u}} S_a(N\tilde{u}) \mid P'$ and then $S_a(N) \mid Q \xrightarrow{\overline{r_x}} \xrightarrow{\mathbf{1}} \xrightarrow{(\tilde{w})\overline{e}\tilde{u}} Q''$ and by Lemma 4.5 $Q'' \equiv S_a(N\tilde{u}) \mid Q'$ and $Q \xrightarrow{\gamma} Q'$, and since $\mathrm{fn}(P', Q') \subseteq (N\tilde{u})$, $(P', Q') \in \mathcal{S}$.

   (c) If $\gamma = (\tilde{w})\overline{x}\tilde{u}$, then by symmetric reasoning $(P', Q') \in \mathcal{S}$. $\square$

Now Theorem 4.2 is a consequence of Lemma 4.8, using $S_a(N)$ contexts: If $\mathcal{R}$ is a bisimulation congruence, $(P,Q) \in \mathcal{R}$ implies that for all contexts $\mathcal{C}$, $(\mathcal{C}[P], \mathcal{C}[Q]) \in \mathcal{R}$, so consider the family of contexts $\mathcal{C}[\cdot] = S_a(N) \mid \cdot$, for $N = \mathrm{fn}(P,Q)$ and a the maximal arity of communication actions in P, Q. Then by Lemma 4.8, $\mathcal{R}$ is a hyperbisimulation. Thus there cannot exist a larger (or "weaker") bisimulation congruence than hyperequivalence for the fusion calculus.

4.2 Weak Hyperequivalence

We now turn to weak equivalence, or *observation equivalence*, where internal behaviour of agents is abstracted; we can only observe their external behaviour. This is useful when reasoning at different abstraction levels, e.g., when relating an abstract specification with a more detailed implementation.

We abstract from internal behaviour by letting a transition of one agent be simulated by zero or more transitions of the other, with the same "observable content". Like CCS or the π-calculus regards τ as unobservable, we regard $\mathbf{1}$, so a communication action γ can be simulated by a sequence of zero or more $\mathbf{1}$ actions followed by γ, followed by a possibly empty sequence of $\mathbf{1}$ actions again. We write $\overset{\gamma}{\Longrightarrow}$ for such a transition sequence, and call it a *weak* transition.

For the case that γ is a fusion the definition of $\stackrel{\gamma}{\Longrightarrow}$ requires some care. The intuition is that $\stackrel{\varphi}{\Longrightarrow}$ represents a sequence of actions with "observable content" φ. But fusions are "observable" only through their substitutive effects. They have no subject and the environment cannot synchronize with them and keep track of, e.g., how many have been performed. Therefore we allow $\stackrel{\varphi}{\Longrightarrow}$ to be a sequence of fusions whose aggregated substitutive effect is the same as that of φ. This view is strengthened further when we consider barbed bisimulation in Section 4.3.

Definition 28 *Define the* composition *of two transitions,* $\circ$, *by* $P(\xrightarrow{\gamma} \circ \xrightarrow{\gamma'})Q$ *iff there exists an agent* P' *such that* $P \xrightarrow{\gamma} P'$ *and* $P'\sigma_\gamma \xrightarrow{\gamma'} Q$, *where* σ_γ *is a substitutive effect of* γ. *Define the* conjunction *of two fusions* φ *and* ψ, *written* $\varphi \wedge \psi$, *to be the least equivalence relation containing* φ *and* ψ, *i.e.,* $(\varphi \cup \psi)^*$. *Define the* weak transition $\stackrel{\gamma}{\Longrightarrow}$ *by the following:* $P \stackrel{\gamma}{\Longrightarrow} Q$ *means that for some* $n \geq 0$, $P \xrightarrow{\gamma_1} \circ \cdots \circ \xrightarrow{\gamma_n} Q$ *and either of*

1. γ *is a communication and* $\gamma = \gamma_i$ *for some* i *and* $\gamma_j = \mathbf{1}$ *for all* $j \neq i$, *or*
2. γ *and all* γ_i *are fusions and* $\gamma = \gamma_1 \wedge \cdots \wedge \gamma_n$. *Here we allow* $n = 0$ *where the empty conjunction is* $\mathbf{1}$, *in other words* $P \stackrel{\mathbf{1}}{\Longrightarrow} P$ *holds for all* P.

The weak bisimulation and bisimilarity $\dot{\approx}$ are as usual defined by replacing $Q \xrightarrow{\gamma} Q'$ by $Q \stackrel{\gamma}{\Longrightarrow} Q'$ in the definition of strong bisimulation (Definition 26).

Definition 29 *A* weak bisimulation *is a binary symmetric relation* $\mathcal{S}$ *between agents such that* $P \,\mathcal{S}\, Q$ *implies:*

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

P *is weakly bisimilar to* Q, *written* $P \dot{\approx} Q$, *if* $P \,\mathcal{S}\, Q$ *for some weak bisimulation* $\mathcal{S}$.

An illuminating exercise is to verify that

$$\{x=y\}\,.\,\{v=w\}\,.\,P \;\dot{\approx}\; \{x=y\}\,.\,\{v=w\}\,.\,P + \{x=y, v=w\}\,.\,P \tag{4.1}$$

but not

$$\{x=y\}\,.\,\{v=w\}\,.\,P \quad \dot{\approx} \quad \{x=y, v=w\}\,.\,P$$

since the RHS cannot simulate the transition $\xrightarrow{\{x=y\}}$. Another example is

$$\varphi\,.\,\varphi\,.\,P \,\dot{\approx}\, \varphi\,.\,P$$

Weak hyperequivalence is defined by closing under substitution, just like in Definition 27. We will sometimes refer to hyperequivalence (Definition 27) as "strong hyperequivalence", to distinguish it from weak hyperequivalence.

Definition 30 *A* weak hyperbisimulation *is a substitution closed weak bisimulation, i.e., a weak bisimulation $\mathcal{S}$ with the property that $P \; \mathcal{S} \; Q$ implies $P\sigma \; \mathcal{S} \; Q\sigma$ for any substitution σ. Two agents P and Q are* weakly hyperequivalent, *written $P \approx Q$, if they are related by a weak hyperbisimulation.*

As for the strong hyperequivalence it suffices to consider idempotent substitutions in Definition 30.

Weak hyperequivalence is not a congruence, but it is a congruence if we replace the summation operator by *guarded* summation $\sum \alpha_i \,.\, P_i$. The standard example of why the congruence property does not hold for general summation is that $b \,.\, \mathbf{0} \approx \mathbf{1} \,.\, b \,.\, \mathbf{0}$, but $a \,.\, \mathbf{0} + b \,.\, \mathbf{0} \not\approx a \,.\, \mathbf{0} + \mathbf{1} \,.\, b \,.\, \mathbf{0}$. (This is as expected: also in CCS and in the π-calculus weak bisimulation equivalence fails to be a congruence for the same reason.)

Theorem 4.9 *With guarded summation, weak hyperequivalence is the largest congruence in weak bisimilarity.*

This is proved the same way as the strong case, Theorem 4.2, using weak variants of its supporting lemmas. They are only stated here; their proofs are very similar to their strong counterparts.

Lemma 4.10 *For a fixed N and a the maximal arity of communication actions in P and Q, assume $S_a(N) \mid P \stackrel{\cdot}{\approx} S_a(N) \mid Q$. Then $(S_a(N) \mid P)\{x/y\} \stackrel{\cdot}{\approx} (S_a(N) \mid Q)\{x/y\}$ for all $x, y \in N$.*

Lemma 4.11 *For any N and a the maximal arity of communication actions in P, and α free, $\mathrm{subj}(\alpha) \notin \mathcal{F}^a_N$,*

1. *if $P \stackrel{\alpha}{\Longrightarrow} P'$ then $S_a(N) \mid P \stackrel{\alpha}{\Longrightarrow} S_a(N) \mid P'$*

2. *if $S_a(N) \mid P \stackrel{\alpha}{\Longrightarrow} R$ then $P \stackrel{\alpha}{\Longrightarrow} P'$ and $R \equiv S_a(N) \mid P'$*

Lemma 4.12 *For any N, a the maximal arity of communication actions in P, $x \in N$ and $\tilde{w} \subseteq \tilde{y}$, it holds that*

$$S_a(N) \mid P \stackrel{r_x}{\Longrightarrow}\stackrel{(\tilde{w})\overline{e}\,\tilde{y}}{\Longrightarrow} R \quad \textit{iff} \quad R \equiv S_a(N\tilde{y}) \mid P' \textit{ and } P \stackrel{(\tilde{w})\overline{x}\,\tilde{y}}{\Longrightarrow} P' \,.$$

Lemma 4.13 *If $P \mid (R{+}R') \stackrel{\cdot}{\approx} Q \mid (R{+}R')$ where $R{+}R'$ is a guarded sum, then $P \mid R \stackrel{\cdot}{\approx} Q \mid R$, where $\mathrm{init}(R) \cap \mathrm{init}(R') = \emptyset$, and $\mathrm{n}(\mathrm{init}(R, R')) \cap \mathrm{fn}(P, Q) = \emptyset$.*

Lemma 4.14 *$(S_a(Ny) \mid P)\{x/y\} \stackrel{\cdot}{\approx} (S_a(Ny) \mid Q)\{x/y\}$ implies $S_a(N) \mid P\{x/y\} \stackrel{\cdot}{\approx} S_a(N) \mid Q\{x/y\}$, if $y \notin N$ and $x \in N$.*

Lemma 4.15
$\mathcal{S} \equiv \{(P, Q) : a = \mathrm{am}(P, Q), \forall N : \mathrm{fn}(P, Q) \subseteq N, \forall k : S_a(N) \mid P \stackrel{\cdot}{\approx} S_a(N) \mid Q\}$ is a weak hyperbisimulation, where $\mathrm{am}(P, Q)$ is the maximal arity of communication actions in P or Q.

Now Theorem 4.9 follows from Lemma 4.15 just like Theorem 4.2 follows from Lemma 4.8.

4.2.1 Weak Hypercongruence

We noted that weak hyperequivalence was only a congruence under *guarded* summation. To get a congruence in the full fusion calculus, we use a standard trick and require that the first transition of either agent must be matched by *at least one* transition of the other.

Definition 31 *Define $P \overset{\gamma}{\Longrightarrow}_+ Q$ to mean $P \overset{\gamma}{\Longrightarrow} Q$ if $\gamma \neq \mathbf{1}$, and $P \overset{\mathbf{1}}{\longrightarrow}\overset{\gamma}{\Longrightarrow} Q$ if $\gamma = \mathbf{1}$. Two agents P and Q are* weakly hypercongruent*, written $P \approx_+ Q$, iff for any substitution σ*

$$P\sigma \xrightarrow{\gamma} P' \textit{ and } \mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset \textit{ implies } Q\sigma \overset{\gamma}{\Longrightarrow}_+ Q' \textit{ and } P'\sigma_\gamma \approx Q'\sigma_\gamma$$

where σ_γ is a substitutive effect of γ, and vice versa.

Again, idempotent substitutions σ will suffice.

Notice that this relation is not, in a sense, purely observational, since we require that the simulating agent perform at least one *unobservable* transition in the case where $\gamma = \mathbf{1}$.

Proposition 4.16 *Weak hypercongruence is the largest congruence in weak hyperequivalence.*
PROOF: Very much as for the corresponding result in [Mil89], pages 153–154. □

4.3 Barbed Congruence

In this section we return to the *guarded* fusion calculus which we defined in Section 3.3. We use the idea of *barbed bisimulation*, developed by Milner and Sangiorgi [MS92a] in the setting of CCS and further investigated in a π-calculus setting by Sangiorgi [San93]. Two agents are considered equivalent if their reductions match and they are indistinguishable under global observations. The question is, what is the minimal observation power needed for the congruence to coincide with the labelled bisimulation congruence?

If there are no observables, the resulting *reduction congruence* is very weak, but turns out to be sufficient for divergence-free CCS [MS92a]. For possibly divergent CCS processes, it is sufficient to be able to observe the possibility of communication [MS92a], while in the π-calculus we must also observe the subject of the communication [San93]. With the addition of fusion actions, you may suspect that these need to be observable. This is not the case: we can already observe fusions indirectly using the match and mismatch operators. The observation predicate below is thus the same as for the π-calculus (Definition 11).

Definition 32 *The observation relation is the least relation satisfying the rules below.*

$x\tilde{y} \,.\, P \downarrow x$	
$\overline{x}\,\tilde{y} \,.\, P \downarrow x$	
$[x = x]P \downarrow y$	*if* $P \downarrow y$
$[x \neq z]P \downarrow y$	*if* $P \downarrow y$ *and* $x \neq z$
$A(\tilde{y}) \downarrow z$	*if* $A(\tilde{x}) \stackrel{def}{=} P$, $\lvert\tilde{y}\rvert = \lvert\tilde{x}\rvert$ *and* $P\{\tilde{y}/\tilde{x}\} \downarrow z$
$(P \mid Q) \downarrow x$	*if* $P \downarrow x$ *or* $Q \downarrow x$
$(P + Q) \downarrow x$	*if* $P \downarrow x$ *or* $Q \downarrow x$
$(x)P \downarrow y$	*if* $P \downarrow y$ *and* $x \neq y$

We repeat the standard definition of barbed bisimulation from Section 2.2.1:

Definition 33 *A* barbed bisimulation *is a symmetric binary relation $\mathcal{S}$ between agents such that $P \,\mathcal{S}\, Q$ implies:*

1. *If $P \to P'$ then $Q \to Q'$ and $P' \,\mathcal{S}\, Q'$.*
2. *If $P \downarrow x$ for some x, then $Q \downarrow x$.*

P is barbed bisimilar to Q, written $P \stackrel{\cdot}{\sim}_b Q$, if $P \,\mathcal{S}\, Q$ for some barbed bisimulation $\mathcal{S}$.

We note that barbed bisimulation is a very weak equivalence: not only is $u \,.\, v \,.\, \mathbf{0} \stackrel{\cdot}{\sim}_b u \,.\, \mathbf{0}$, but in the fusion calculus also $\varphi \,.\, P \stackrel{\cdot}{\sim}_b \mathbf{0}$ for any P and for φ different from $\mathbf{1}$, since a non-inert fusion prefix has neither reductions nor observations. By closing the bisimulation under contexts we obtain a more interesting relation:

Definition 34 *Two agents P and Q are* barbed congruent*, written $P \sim_b Q$, if for all contexts $C[\cdot]$, it holds that $C[P] \stackrel{\cdot}{\sim}_b C[Q]$.*

For CCS, the barbed congruence coincides with strong bisimulation [MS92a]. For the π-calculus, it coincides with the early bisimulation congruence [San93], presenting a strong argument for early bisimulation as the most natural for π-calculus. For the fusion calculus we can show that it coincides with hyper-equivalence.

Theorem 4.17 *$P \sim Q$ iff $P \sim_b Q$.*
PROOF: Every hyperbisimulation is clearly a barbed bisimulation, so $\sim \subseteq \stackrel{\cdot}{\sim}_b$, and since $\sim$ is a congruence the also $\sim \subseteq \sim_b$. We know from Theorem 4.2 that $\sim$ is the largest congruence contained in bisimulation, so we need only show that barbed congruence is also contained in bisimulation to complete the proof. This is done in Lemma 4.18 below. $\square$

Lemma 4.18 *Barbed congruence is contained in bisimulation.*

The proof goes by constructing a context $C[\cdot]$ such that $\{(P, Q) : C[P] \sim_b C[Q]\}$ is a bisimulation. The context used is similar to (but simpler than) the one used for the weak case in Lemma 4.20 below; we omit the details and turn to weak barbed bisimulation.

Definition 35 *A* weak barbed bisimulation *is a symmetric binary relation $\mathcal{S}$ between agents such that $P \;\mathcal{S}\; Q$ implies:*

1. *If $P \rightarrow P'$ then $Q \rightarrow^* Q'$ and $P' \;\mathcal{S}\; Q'$.*
2. *If $P \downarrow x$ for some x, then $Q \rightarrow^* \downarrow x$.*

P is weak barbed bisimilar to Q, written $P \stackrel{\bullet}{\approx}_b Q$, if $P \;\mathcal{S}\; Q$ for some weak barbed bisimulation $\mathcal{S}$. Two agents P and Q are weak barbed congruent, *written $P \approx_b Q$, if for all contexts $C[\cdot]$, it holds that $C[P] \stackrel{\bullet}{\approx}_b C[Q]$, where $C[\cdot]$ only uses guarded summation.*

Theorem 4.19 *$P \approx_b Q$ iff $P \approx Q$.*

The proof is just like that for Theorem 4.17, and needs Theorem 4.9 of Section 4.2 and Lemma 4.20 below.

Lemma 4.20 *Weak barbed congruence is contained in weak bisimulation.*
PROOF: The proof is reminiscent of the corresponding proof for the π-calculus [San93], but simpler. We construct a family of contexts $C^{\tilde{z}}_{n,k}[\cdot]$ such that

$$\mathcal{R} = \{(P,Q) : C^{\tilde{z}}_{n,k}[P] \approx_b C^{\tilde{z}}_{n,k}[Q], \mathrm{fn}(P,Q) \subseteq \tilde{z}, n = |\tilde{z}|, k = \mathrm{am}(P,Q)\}$$

is a weak bisimulation, where $\mathrm{am}(P,Q)$ is the maximal arity of communication actions in P and Q.

In the definition below we postulate for each name u a fresh name u'; for each finite sequence of names $\tilde{y}$ a fresh name y''; for each finite equivalence relation e on names a fresh name m_e; an for each equivalence e and inequivalence d on pairs of names fresh names m^i_e and m^i_d for $i \geq 0$. Furthermore, $\{c, s, in, out, d_n : n \geq 0\}$ contains fresh names. Freshness guarantees that fresh names can never be confused with names used in P or Q. M_e is a sequence of match and mismatch operators agreeing with the equivalence relation e, and $\mathrm{eq}(\tilde{z})$ stands for the set of equivalence relations on names in $\tilde{z}$.

$$\begin{array}{rcl}
C^{\tilde{z}}_{n,k}[\cdot] & \equiv & (\tilde{z})(\cdot \mid V^k_n(\tilde{z}) \mid \mathit{Count}_n) \\
\mathit{Count}_n & \stackrel{def}{=} & d_n + c \,.\, \mathit{Count}_{n+1} \\
V^k_n(\tilde{z}) & \stackrel{def}{=} & \displaystyle\sum_{u\in\tilde{z}, |\tilde{y}|\leq k} (\tilde{y}) u\, \tilde{y} \,.\, \overline{c} \,.\, (\mathbf{1} \,.\, (u' + \mathit{out} + y'' + X(\tilde{y},\tilde{z})) + \mathbf{1} \,.\, V^k_{n+1}(\tilde{z}\tilde{y})) \\
& + & \displaystyle\sum_{u\in\tilde{z}, |\tilde{y}|\leq k} (\tilde{y}) \overline{u}\, \tilde{y} \,.\, \overline{c} \,.\, (\mathbf{1} \,.\, (u' + \mathit{in} + y'' + X(\tilde{y},\tilde{z})) + \mathbf{1} \,.\, V^k_{n+1}(\tilde{z}\tilde{y})) \\
& + & \mathbf{1} \,.\, s \\
& + & \mathbf{1} \,.\, \displaystyle\sum_{e\in\mathrm{eq}(\tilde{z})} M_e m_e
\end{array}$$

where $X(\tilde{y},\tilde{z})$ stands for $\sum_{i=1}^{|\tilde{y}|} \sum_{z\in\tilde{z}} ([y_i = z] m^i_{y_i=z} + [y_i \neq z] m^i_{y_i\neq z})$.

Count is used to control the number of interactions between V and the agent put in the context: for each interaction, the observation d_n is replaced by d_{n+1}.

Note that the summations on the first two lines are finite up to alpha-conversion.

The observables m_e^i, m_d^i in $X(\tilde{y}, \tilde{z})$ tells what elements of $\tilde{y}$ are equal to what elements of $\tilde{z}$, by making a name observable corresponding to the resulting (in)equivalences for each element of $\tilde{y}$. Note that the position of each y_i in $\tilde{y}$ must be encoded in the observable name since in $X(\tilde{y}, \tilde{z})$, $\tilde{y}$ needs not be pairwise distinct.

Now, for $(P, Q) \in \mathcal{R}$,

1. If $P \xrightarrow{(\tilde{x})u\tilde{w}} P'$, we must find a Q' such that $Q \overset{(\tilde{x})u\tilde{w}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$. $C_{n,k}^{\tilde{z}}[P] \rightarrow\rightarrow (\tilde{z})(\tilde{x})(P' \mid W \mid \mathit{Count}_{n+1}) \equiv S$ where $W \equiv (\tilde{x})(\mathbf{1} \,.\, (u' + in + y'' + X(\tilde{w}, \tilde{z})) + \mathbf{1} \,.\, V_{n+1}^k(\tilde{z}\tilde{w}))$. Now, $S \downarrow d_{n+1}$, so for $C[Q]$ to simulate the reduction, it must reduce involving Q and V. It cannot let Q communicate more than once, since $S \rightarrow S'$ s.t. $S' \downarrow d_{n+1}$ and $S' \downarrow u_i'$, but not $S' \rightarrow^* \downarrow d_{n+2}$. Furthermore, $S' \downarrow u_i'$ and $S \downarrow in$, so $Q \overset{\alpha}{\Longrightarrow} Q'$ with $\mathrm{subj}(\alpha) = u_i$, and $S' \downarrow y''$, so $|\tilde{y}| = |\mathrm{obj}(\alpha)|$. Also, the set of observables from the $X(\tilde{y}, \tilde{z})$ term must be preserved. This gives $\mathrm{obj}(\alpha) = \tilde{w}$, where $\tilde{x} \subseteq \tilde{w}$ is fresh. Then $Q \overset{(\tilde{x})u\tilde{w}}{\Longrightarrow} Q'$ and $C[Q] \rightarrow^* (\tilde{z})(\tilde{x})(Q' \mid W \mid \mathit{Count}_{n+1}) \equiv T$. Finally the reduction $S \rightarrow C_{n+1,k}^{\tilde{z}\tilde{x}}[P']$ must be simulated by $T \rightarrow C_{n+1,k}^{\tilde{z}\tilde{x}}[Q']$ since $C_{n+1,k}^{\tilde{z}\tilde{x}}[P'] \rightarrow S''$ s.t. $S'' \downarrow s$ and $S'' \downarrow d_{n+1}$.
2. If $P \xrightarrow{(\tilde{x})\overline{u}\,\tilde{w}} P'$ the argument is symmetric.
3. If $P \xrightarrow{\varphi} P'$, then $C_{n,k}^{\tilde{z}}[P] \rightarrow C_{n,k}^{\tilde{z}}[P']\sigma \rightarrow S'''$ s.t. $S''' \downarrow m_e$ and $S''' \downarrow d_n$ for σ agreeing with φ and m_e encoding the equivalence relation of names in $\tilde{z}$ agreeing with φ. To preserve this observation, it must be that $Q \overset{\varphi}{\Longrightarrow} Q'$, i.e., Q can perform a sequence of fusion actions which "sum up" to the same effect as φ. There can be no interaction between Q and V since not $S''' \rightarrow^* \downarrow d_{n+1}$. Then $C_{n,k}^{\tilde{z}}[Q] \overset{\mathbf{1}}{\Longrightarrow} C_{n,k}^{\tilde{z}}[Q']\sigma \downarrow m_e$; if $Q \overset{\psi}{\Longrightarrow} Q'$ for some ψ different from φ, $m_f \neq m_e$ will be observable.

Thus, the relation $\mathcal{R}$ is a weak bisimulation. □

The definition of Count_n implicitly uses an infinite number of parameters (remember that all free names must be in the parameter list of the identifier). For the sake of simplifying this proof we allow identifiers with infinite parameter lists as long as there are still infinitely many fresh names available. For image-finite agents (agents P s.t. $\{P' : P \overset{\gamma}{\Longrightarrow} P'\}$ is finite up to $\equiv$), we can avoid infinite parameter lists by parameterizing the context on the maximal length r of the argument list and letting $\mathcal{R}$ include contexts for all r. For each increment in the maximal lengths we get a finer approximation of the equivalence, and when we include all possible (finite) lengths the approximation coincides with the equivalence. This argument only applies to image-finite agents and for image-infinite agents with finite parameter lists we only conjecture that the lemma holds.

Recovering strong and weak hyperequivalence by means of a reduction relation and a simple observation predicate, these results gives us confidence that our definitions of strong and weak hyperequivalence are ***truly observational*** in the sense that two agents are equivalent only when no difference can be observed by interacting with them from the "outside world".

4.4 Symbolic Hyperequivalences

Hyperequivalence as defined in Section 4.1 makes heavy use of universal quantification over substitutions, and barbed congruence does the same for contexts. To automatically decide bisimulation equivalence using these definitions is very inefficient, if at all possible. But using the symbolic transition system of Section 3.4, hyperequivalence can be characterized in a computationally more efficient way, without universal quantification. In Chapter 7 this characterization is used to obtain a decision procedure for a fragment of the fusion calculus, which is implemented in an automatic tool.

A transition of one process may need to be simulated by a *set* of transitions of the other process. Consider the following example:

$$a\,.\,P \sim [x=y]a\,.\,P + [x \neq y]a\,.\,P$$

For each substitution, the $\xrightarrow{a}$ transition of the LHS is simulated by *different* summands of the RHS depending on whether the substitution makes x and y the same or not. In symbolic hyperbisimulation, the action $\xmapsto{\emptyset,a}$ of the LHS must therefore be matched by the *set* containing the two actions $\xmapsto{[x=y],a}$ and $\xmapsto{[x\neq y],a}$ of the RHS. The conditions of the LHS (here $\emptyset$ or "true") imply the disjunction of the conditions of the RHS actions (here $x = y \vee x \neq y$). The important point when simulating with a set of symbolic transitions is that any substitution which satisfies the LHS condition satisfies one of the RHS conditions. Symbolic bisimulation thus uses a set of process relations, indexed by match sequences representing assertions of equalities between names.

Recalling from Section 3.4 that we write $\tilde{M}^+$ for the match part of $\tilde{M}$, we now define the symbolic hyperequivalence.

Definition 36 *The set $\mathcal{S} = \{S^C\}_{\mathcal{C}}$ (for $\mathcal{C}$ a set of sequences of match operators) of binary symmetric process relations is a* symbolic hyperbisimulation *if for each $S^C \in \mathcal{S}$, $(P,Q) \in S^C$ implies:*

If $P \xmapsto{\tilde{M},\gamma} P'$ with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$ and M consistent with C, then
for some $T \subseteq \{Q \xmapsto{\tilde{N}_i,\delta_i} Q'_i : \gamma\sigma = \delta_i\sigma, \text{for all } \sigma \text{ which agree with } C\tilde{M}\tilde{N}_i\}$
$C\tilde{M} \Rightarrow \bigvee \tilde{N}_i$, and for all i, $(P', Q'_i) \in S^{(L_i^+)} \in \mathcal{S}$, where
$L_i = C\tilde{M}\tilde{N}_i\tilde{M}_\gamma$ and $\tilde{M}_\gamma$ a sequence of matches such that
any substitutive effect of γ agrees with $\tilde{M}_\gamma$.

P is symbolically C-hyperequivalent *to Q, written $P \simeq_C Q$, if $(P,Q) \in S^C \in \mathcal{S}$ for some symbolic hyperbisimulation $\mathcal{S}$. We write $P \simeq Q$ if $P \simeq_\emptyset Q$.*

We define the weak symbolic transitions $\models\!\Longrightarrow$ as follows:

Definition 37

$$P \overset{\tilde{M},\gamma}{\models\!\!\Longrightarrow} Q \text{ if } \begin{cases} P \overset{\tilde{M}_1,\mathbf{1}}{\longmapsto} \cdots \overset{\tilde{M}_n,\mathbf{1}}{\longmapsto}\overset{\tilde{L},\gamma}{\longmapsto}\overset{\tilde{N}_1,\mathbf{1}}{\longmapsto} \cdots \overset{\tilde{N}_m,\mathbf{1}}{\longmapsto} Q \\ \quad \text{if } \gamma \text{ a communication action, } m,n \geq 0, \\ \quad \tilde{M} \Leftrightarrow (\bigwedge_{i=1}^n \tilde{M}_i) \wedge \tilde{L} \wedge (\bigwedge_{i=1}^m \tilde{N}_i) \\ P \overset{\tilde{M}_1,\psi_1}{\longmapsto} \cdots \overset{\tilde{M}_n,\psi_n}{\longmapsto} Q \\ \quad \text{if } \gamma \text{ a fusion action, } n \geq 0,\ \tilde{M}_{k+1} \text{ is consistent with} \\ \quad \psi_1 \cdots \psi_k,\ \tilde{M} \Leftrightarrow \bigwedge_{i=1}^n (\tilde{M}_i \sigma_{\psi_1 \cdots \psi_{i-1}}) \text{ is consistent, where} \\ \quad \sigma_{\psi_1 \cdots \psi_{i-1}} \text{ is a substitutive effect of } \psi_1 \wedge \cdots \wedge \psi_{i-1} \text{ and} \\ \quad \gamma = \psi_1 \wedge \cdots \wedge \psi_n. \end{cases}$$

Note that in Definition 28 of the standard weak transitions a substitution is performed on each intermediary agent in the case of a fusion action. In Definition 37 above the substitution needs only be done on the conditions of the intermediary (strong) transitions. This is much more efficient from the point of view of an implementation, but makes the proof of Lemma 4.29 below a little more complicated, requiring Lemmas 4.25 and 4.26.

In the fusion calculus with mismatch, the monotonicity property of [MPW92] does not hold: $P \xrightarrow{\alpha} P'$ does not imply $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$. However, in the symbolic semantics we can formulate a limited monotonicity property:

Lemma 4.21 *If $P \overset{\tilde{M},\gamma}{\longmapsto} P'$ then $P\sigma \overset{\tilde{M}\sigma,\gamma\sigma}{\longmapsto} P'\sigma$ for any σ which respects $\tilde{M}$.*
PROOF: Easy induction on the depth of inference. □

Next we establish the correspondence between symbolic and standard transitions.

Lemma 4.22

1. *If $P \overset{\tilde{M},\gamma}{\longmapsto} P'$ then $P\sigma \xrightarrow{\gamma\sigma} P'\sigma$ for any $\sigma \models \tilde{M}$.*
2. *If $P\sigma \xrightarrow{\gamma'} P''$ then $\exists \tilde{M},\gamma,P' : P \overset{\tilde{M},\gamma}{\longmapsto} P'$ such that $\sigma \models \tilde{M}$, $\gamma' = \gamma\sigma$, and $P'' \equiv P'\sigma$.*

PROOF:

1. By induction on the depth of the inference of the transition. The only non-trivial case is S-COM:

   Suppose $P \equiv P_1 \mid P_2 \overset{\tilde{M},\{\tilde{x}=\tilde{y}\}}{\longmapsto} P_1' \mid P_2'$ because $P_1 \overset{M_1,u\tilde{x}}{\longmapsto} P_1'$ and $P_2 \overset{\tilde{M}_2,\overline{v}\,\tilde{y}}{\longmapsto} P_2'$, with $\tilde{M} \Leftrightarrow \tilde{M}_1\tilde{M}_2[u=v]$. For any σ s.t. $\sigma \models \tilde{M}_1\tilde{M}_2[u=v]$, σ respects both $\tilde{M}_1$ and $\tilde{M}_2$, so by Lemma 4.21, $P_1\sigma \overset{\tilde{M}_1\sigma,(u\tilde{x})\sigma}{\longmapsto} P_1'\sigma$ and

$P_2\sigma \xmapsto{\tilde{M}_2\sigma,(\overline{v}\,\tilde{y})\sigma} P_2'\sigma$. And since $\sigma \models \tilde{M}_1$ and $\sigma \models \tilde{M}_2$, $P_1\sigma \xrightarrow{(u\tilde{x})\sigma} P_1'\sigma$ and $P_2\sigma \xrightarrow{(\overline{v}\,\tilde{y})\sigma} P_2'\sigma$ and $\sigma \models [u = v]$, it follows that $(P_1 \mid P_2)\sigma \xrightarrow{\{\tilde{x}=\tilde{y}\}\sigma} (P_1' \mid P_2')\sigma$.

2. Simple induction on the depth of inference. □

In the rest of this subsection, we write σ_φ for a substitutive effect of φ. Note that all substitutive effects are idempotent substitutions.

Lemma 4.23 *For any idempotent substitution σ and any x and y, $x\sigma\sigma_{\varphi\sigma} = y\sigma\sigma_{\varphi\sigma}$ iff $x\sigma\sigma_\varphi = y\sigma\sigma_\varphi$.*
PROOF: $x\sigma\sigma_{\varphi\sigma} = y\sigma\sigma_{\varphi\sigma}$ iff $x\sigma\ (\varphi\sigma)\ y\sigma$ iff $x\sigma\sigma\ \varphi\ y\sigma\sigma$ iff $x\sigma\ \varphi\ y\sigma$ iff $x\sigma\sigma_\varphi = y\sigma\sigma_\varphi$. □

The adequacy of the symbolic equivalence is given by showing that it coincides with its non-symbolic counterpart, using the correspondence between symbolic and non-symbolic transitions.

Theorem 4.24 *$P \sim Q$ iff $P \simeq Q$.*
PROOF:

⇒: We show that $\mathcal{S} = \{S^C = \{(P,Q) : P\sigma \sim Q\sigma \text{ for any } \sigma \models C\}\}$ is a symbolic hyperbisimulation.

For $(P,Q) \in S^C$, if $P \xmapsto{\tilde{M},\gamma} P'$, then for any $\sigma \models C\tilde{M}$, by Lemma 4.22 $P\sigma \xrightarrow{\gamma\sigma} P'\sigma$, and $P\sigma \sim Q\sigma$, so $Q\sigma \xrightarrow{\gamma\sigma} Q''$ such that $P'\sigma\sigma_{\gamma\sigma} \sim Q''\sigma_{\gamma\sigma}$. By Lemma 4.22 there exist $Q', \gamma', \tilde{N}$ such that $Q \xmapsto{\tilde{N},\gamma'} Q'$ and $\sigma \models \tilde{N}$, $\gamma\sigma = \gamma'\sigma$ and $Q'' \equiv Q'\sigma$, so, $P'\sigma\sigma_{\gamma\sigma} \sim Q'\sigma\sigma_{\gamma\sigma}$.

Let $\tilde{M}_\gamma$ be a sequence of matches such that any substitutive effect of γ agrees with $\tilde{M}_\gamma$. We know $\sigma \models C\tilde{M}\tilde{N}$ and $\sigma\gamma \models \tilde{M}_\gamma$, so $\sigma\sigma_\gamma \models (C\tilde{M}\tilde{N}\tilde{M}_\gamma)^+$, and by Lemma 4.23 $\sigma\sigma_{\gamma\sigma} \models (C\tilde{M}\tilde{N}\tilde{M}_\gamma)^+$, so $(P',Q') \in S^{(\tilde{L}^+)}$ where $\tilde{L} = C\tilde{M}\tilde{N}\tilde{M}_\gamma$.

Collecting these transitions of Q, we can form a set $T = \{Q \xmapsto{N_i,\delta_i} Q_i' : \gamma\sigma_i = \delta_i\sigma_i, \sigma_i \text{ agrees with } C\tilde{M}\tilde{N}_i\}$. Since for any σ such that $\sigma \models C\tilde{M}$ there exists an element i of T such that $\sigma \models N_i$, we know $C\tilde{M} \Rightarrow \bigvee \tilde{N}_i$. Also for all i, $(P', Q_i') \in S^{(\tilde{L}^+)}$ for $\tilde{L}$ as above.

Thus $\mathcal{S}$ is a symbolic hyperbisimulation.

⇐: We show that $\mathcal{R} = \{(P\sigma, Q\sigma) : (P,Q) \in S^C \in \mathcal{S} \text{ for all } \sigma \models C\}$ is a hyperbisimulation, where $\mathcal{S}$ is a symbolic hyperbisimulation.

For $(P\sigma, Q\sigma) \in \mathcal{R}$, if $P\sigma \xrightarrow{\gamma} P'$, then by Lemma 4.22 $P \xmapsto{\tilde{M},\gamma'} P''$ s.t. $\sigma \models \tilde{M}$, $\gamma = \gamma'\sigma$ and $P' \equiv P''\sigma$. Then for some $T \subseteq \{Q \xmapsto{\tilde{N}_i,\delta_i} Q_i' : \gamma'\sigma_i = \delta_i\sigma_i, \sigma_i \text{ agrees with } C\tilde{M}\tilde{N}_i\}$ such that $\forall i \in T : (P'', Q_i') \in S^{(\tilde{L}^+)}$

for $\tilde{L} = C\tilde{M}\tilde{N}_i\tilde{M}_{\gamma'}$ and $C\tilde{M} \Rightarrow \bigvee \tilde{N}_i$. So for any $\sigma \models C\tilde{M}$ there is an element i in T s.t. $\sigma \models C\tilde{M}\tilde{N}_i$, and thus by Lemma 4.22 $Q\sigma \xrightarrow{\delta_i\sigma} Q'_i\sigma$, $\delta_i\sigma = \gamma'\sigma = \gamma$ and $(P'\sigma_\gamma, Q'_i\sigma\sigma_\gamma) \in \mathcal{R}$, where σ_γ a substitutive effect of γ. □

Lemma 4.25 *For all φ and ψ there is an injective substitution ρ such that $\sigma_{\varphi\sigma_\psi}\rho = \sigma_\varphi\sigma_\psi$.*
PROOF: We show that $\sigma_{\varphi\sigma_\psi}$ and $\sigma_\varphi\sigma_\psi$ equate the same names, so then trivially the one substitution is equal to the other modulo an injective substitution.
$\sigma_{\varphi\sigma_\psi}(x) = \sigma_{\varphi\sigma_\psi}(y)$ iff $x\ (\varphi\sigma_\psi)\ y$ iff $\sigma_\psi(x)\ \varphi\ \sigma_\psi(y)$ iff $\sigma_\varphi\sigma_\psi(x) = \sigma_\varphi\sigma_\psi(y)$. □

Lemma 4.26 *For all φ and ψ there is an injective substitution ρ such that $\sigma_\varphi\sigma_\psi\rho = \sigma_{(\varphi\wedge\psi)}$.*
PROOF: By the same reasoning as in the previous Lemma, it suffices to show that $\sigma_\varphi\sigma_\psi$ and $\sigma_{(\varphi\wedge\psi)}$ equate the same names.
$\sigma_\varphi\sigma_\psi = (\sigma_\varphi\sigma_\psi)^*$ (because both substitutions are idempotent), and $\sigma_\varphi\sigma_\psi(x) = \sigma_\varphi\sigma_\psi(y)$ iff $x\ (\psi\varphi\psi)\ y$, so $x\ (\psi\varphi\psi)\ y$ iff $x\ (\psi\varphi\psi)^*\ y$ iff $x\ (\varphi\psi)^*\ y$ iff $x\ (\varphi\cup\psi)^*\ y$ iff $x\ (\varphi\wedge\psi)\ y$ iff $\sigma_{\varphi\wedge\psi}(x) = \sigma_{\varphi\wedge\psi}(y)$. □

Lemma 4.27 $\sigma_\varphi\sigma_{\varphi\wedge\psi} = \sigma_{\varphi\wedge\psi}$.
PROOF: $\varphi \subseteq \varphi\wedge\psi$. □

Lemma 4.28 *$P \equiv Q$ iff $P\rho \equiv Q\rho$ for any injective substitution ρ.*
PROOF: To the right: by straight-forward induction over the inference of structural congruence. To the left: by observing that any injective substitution has an inverse. □

We can now relate the symbolic and non-symbolic weak transitions:

Lemma 4.29

1. *If $\sigma \models \tilde{M}$ and $P \overset{\tilde{M},\gamma}{\Longmapsto} P'$, then $P\sigma \overset{\gamma\sigma}{\Longrightarrow} P''$ s.t. $P''\sigma_{\gamma\sigma} \equiv P'\sigma\sigma_{\gamma\sigma}$.*
2. *If $P\sigma \overset{\gamma}{\Longrightarrow} P''$ then there are $\tilde{M}, \gamma', P'$ such that $P \overset{\tilde{M},\gamma'}{\Longmapsto} P'$ and $\sigma \models \tilde{M}$, $\gamma = \gamma'\sigma$, and $P''\sigma_\gamma \equiv P'\sigma\sigma_{\gamma'\sigma}$.*

PROOF: By induction on the number of strong transitions in the weak transitions.

1. The base cases are simple, using Lemma 4.21, as are the induction cases where γ is a communication action. The interesting case is where γ is a fusion action:

   Then $\gamma = \varphi_1 \wedge \varphi_2$ and $\tilde{M} \Leftrightarrow \tilde{M}_1(\tilde{M}'\sigma_{\varphi_1})$ where σ_{φ_1} is a substitutive effect of φ_1, and $P \overset{\tilde{M}_1,\varphi_1}{\longmapsto} P'_1$, $P'_1 \overset{\tilde{M}',\varphi'}{\Longmapsto} P'$.

   By Lemma 4.21 $P\sigma \xrightarrow{\varphi_1\sigma} P'_1\sigma$, and by induction $P'_1\sigma' \overset{\varphi'\sigma'}{\Longrightarrow} P''$ for any $\sigma' \models \tilde{M}'$, s.t. $P''\sigma_{\varphi'\sigma'} \equiv P'\sigma'\sigma_{\varphi'\sigma'}$.

So if $\sigma\sigma_{\varphi_1\sigma} \models \tilde{M}'$, we know that $P_1'\sigma\sigma_{\varphi_1\sigma} \xRightarrow{\varphi'\sigma\sigma_{\varphi_1\sigma}} P''$ such that $P''\sigma_{\varphi'\sigma\sigma_{\varphi_1\sigma}} \equiv P'\sigma(\sigma_{\varphi_1\sigma})(\sigma_{\varphi'\sigma(\sigma_{\varphi_1\sigma})})$. By Lemma 4.23 $\sigma\sigma_{\varphi_1\sigma} \models \tilde{M}'$ iff $\sigma\sigma_{\varphi_1} \models \tilde{M}'$ iff $\sigma \models \tilde{M}'\sigma_{\varphi_1}$, and since $\sigma \models \tilde{M}$ and $\tilde{M} \Leftrightarrow \tilde{M}_1(\tilde{M}'\sigma_{\varphi_1})$ this holds.

We need to show that $P''\sigma_{\varphi\sigma} \equiv P'\sigma\sigma_{\varphi\sigma}$, i.e., $P''\sigma_{(\varphi_1\wedge\varphi')\sigma} \equiv P'\sigma\sigma_{(\varphi_1\wedge\varphi')\sigma}$.

We know $P''\sigma_{\varphi'\sigma(\sigma_{\varphi_1\sigma})} \equiv P'\sigma(\sigma_{\varphi_1\sigma})(\sigma_{\varphi'\sigma(\sigma_{\varphi_1\sigma})})$, and by Lemmas 4.25 and 4.26, $P''\sigma_{\varphi'\sigma(\sigma_{\varphi_1\sigma})} \equiv P''\sigma_{(\varphi_1\wedge\varphi')\sigma}\rho$ for an injective substitution ρ. Again by Lemmas 4.25 and 4.26, $P'\sigma(\sigma_{\varphi_1\sigma})(\sigma_{\varphi'\sigma(\sigma_{\varphi_1\sigma})}) \equiv P'\sigma(\sigma_{\varphi_1\sigma})(\sigma_{(\varphi_1\wedge\varphi')\sigma})\rho$ for the *same* injective substitution ρ, and by Lemma 4.27 $P'\sigma(\sigma_{\varphi_1\sigma})(\sigma_{(\varphi_1\wedge\varphi')\sigma})\rho \equiv P'\sigma(\sigma_{(\varphi_1\wedge\varphi')\sigma})\rho$. We now have $P''\sigma_{(\varphi_1\wedge\varphi')\sigma}\rho \equiv P'\sigma(\sigma_{(\varphi_1\wedge\varphi')\sigma})\rho$, and by Lemma 4.28, $P''\sigma_{(\varphi_1\wedge\varphi')\sigma} \equiv P'\sigma(\sigma_{(\varphi_1\wedge\varphi')\sigma})$ as required.

2. Similar. □

The *weak* symbolic hyperbisimulation and hyperequivalence $\approxeq$ are defined by replacing $Q \xmapsto{\tilde{N}_i,\delta_i} Q_i'$ by $Q \models\!\!\xRightarrow{\tilde{N}_i,\delta_i} Q_i'$ in Definition 36.

Definition 38 *The set $\mathcal{S} = \{S^C\}_\mathcal{C}$ (for $\mathcal{C}$ a set of sequences of match operators) of binary symmetric process relations is a* weak symbolic hyperbisimulation *if for each $S^C \in \mathcal{S}$, $(P,Q) \in S^C$ implies:*

If $P \xmapsto{\tilde{M},\gamma} P'$ with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$ and M consistent with C, then
for some $T \subseteq \{Q \models\!\!\xRightarrow{\tilde{N}_i,\delta_i} Q_i' : \gamma\sigma = \delta_i\sigma, \text{for all } \sigma \text{ s.t. } \sigma \text{ agrees with } C\tilde{M}\tilde{N}_i\}$
$C\tilde{M} \Rightarrow \bigvee \tilde{N}_i$, and for all i, $(P',Q_i') \in S^{(L_i^+)} \in \mathcal{S}$, where
$L_i = C\tilde{M}\tilde{N}_i\tilde{M}_\gamma$ and $\tilde{M}_\gamma$ agrees with the substitutive effect of γ.

P is weak symbolically C-hyperequivalent *to Q, written $P \approxeq_C Q$, if $(P,Q) \in S^C \in \mathcal{S}$ for some symbolic hyperbisimulation $\mathcal{S}$. We write $P \approxeq Q$ if $P \approxeq_\emptyset Q$.*

Theorem 4.30 *$P \approx Q$ iff $P \approxeq Q$.*
PROOF: Like the proof of Theorem 4.24, using Lemma 4.29 in place of Lemma 4.22 at appropriate places. □

4.4.1 Symbolic Open Bisimulation in the π-Calculus

With only minor modifications the framework developed above can be adapted to the open bisimulation equivalence of the π-calculus. In Sangiorgi's original work [San96c] it was defined for the π-calculus without the mismatch operator, and a symbolic transition system and characterization of the equivalence were given. In this section we extend the symbolic open bisimulation to handle mismatch.

In any reasonable equivalence in the π-calculus, we would expect the following equation to hold.

$$[x \neq y]\alpha \,.\, P \sim [x \neq y]\alpha \,.\, [x \neq y]P \tag{4.2}$$

(where α does not bind x, y). Contrary to the case of the fusion calculus, no context can make x and y equal once they have been found to be distinct.

However, in open bisimulation as defined in Section 2.2, (4.2) does not hold. Sangiorgi says in [San96c] that the addition of mismatch conflicts with the requirement of closure under all substitutions; to handle closure under substitutions we need to update the distinction when P passes a mismatch operator. It is hard to see how to discover this when only using the standard labelled transition semantics. However, in the symbolic semantics we can easily discover the mismatch requirement of a transition so that the distinction can be updated, and we can formulate a symbolic bisimulation equivalence with the desired properties.

For simplicity we choose to work in the subcalculus f_π instead of the π-calculus proper; this allows us to use the symbolic semantics defined in Section 3.4.

Definition 39 *A distinction D* agrees with *a match sequence $\tilde{M}$ if $(x, y) \in D \Leftrightarrow (\tilde{M} \Rightarrow [x \neq y])$, and $\tilde{M}$* respects *D if $\tilde{M} \Rightarrow [x = y]$ implies $(x, y) \notin D$.*

Definition 40 *The set $\mathcal{S} = \{S^C\}_\mathcal{C}$ (for $\mathcal{C}$ a set of match sequences) of binary symmetric process relations is a* symbolic open bisimulation *if for each S^C, $(P, Q) \in S^C$ implies:*

If $P \xrightarrow{\tilde{M},\gamma} P'$ with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$ and $\tilde{M}$ consistent with C, then
for some $T \subseteq \{Q \xrightarrow{\tilde{N}_i,\delta_i} Q'_i : \gamma\sigma = \delta_i\sigma$, for all σ which agree with $C\tilde{M}\tilde{N}_i\}$
$C\tilde{M} \Rightarrow \bigvee \tilde{N}_i$, and for all i, $(P', Q'_i) \in S^{L_i} \in \mathcal{S}$, where

$$\begin{cases} L_i = C\tilde{M}\tilde{N}_i D & \text{if } \gamma \text{ is a bound output action } (\tilde{x})\overline{u}\,\tilde{y}, \text{ with} \\ & D = \bigwedge\{[x \neq y] : (x, y) \in \tilde{x} \times \mathrm{fn}(P, Q)\} \\ L_i = C\tilde{M}\tilde{N}_i & \text{otherwise} \end{cases}$$

P is symbolically open C-bisimilar *to Q, written $P \simeq^o_C Q$, if $(P, Q) \in S^C \in \mathcal{S}$ for some symbolic open bisimulation $\mathcal{S}$. P is* symbolically open D-bisimilar *to Q, written $P \simeq^o_D Q$, if $(P, Q) \in S^C \in \mathcal{S}$ for some symbolic open bisimulation $\mathcal{S}$, where C is a mismatch sequence corresponding to the distinction D.*

In the non-symbolic definition of open bisimilarity (Definition 10), the freshness of new names is recorded in the distinction relation, while in the symbolic bisimulation in Definition 40 they are recorded in the condition C. In Definition 10, equalities are implicitly recorded by the use of substitutions, while in Definition 40 equalities are explicitly added to the condition. In addition, when a transition is only possible if some names are different, we can add this to the condition C. It then turns out that equation (4.2) holds for open equivalence.

The differences between Definition 40 above and Definition 36 (for symbolic hyperequivalence) are that mismatches are removed from L_i in Definition 36 but not in Definition 40, where names bound in output actions are recorded by mismatches. (The case for fusion actions is omitted, since there are no such actions in the subcalculus we are considering.)

Weak symbolic open bisimulation and D-bisimilarity, $\approxeq^o_D$, is obtained by replacing $Q \xmapsto{N_i,\delta_i} Q'_i$ with $Q \stackrel{N_i,\delta_i}{\Longmapsto} Q'_i$ in Definition 40.

The equivalences of Definitions 10 and 40 coincide for the calculus without the mismatch operator, as is stated by the following Theorem.

Theorem 4.31 *For P, Q without mismatch operators, $P \simeq^o Q$ iff $P \sim^o Q$.*
PROOF: Along the lines of the proof of Theorem 4.24. The set T is always a singleton. $\square$

The corresponding result for the weak case can be proved in a similar fashion.

Chapter 5

Algebraic Laws

In Chapter 4 equality between agents was defined based on their *behaviour*. Understanding the implications of such definitions can be difficult, but might be simplified by providing an equivalent definition based on the *structure* of the agents. This chapter gives sets of algebraic laws, or axiom systems, satisfied by hyperequivalence and weak hypercongruence. The axioms are *sound*, meaning that if they equate two agents then they are behaviourally equivalent, and for finite processes they are proved to be *complete*, meaning that if two agents are behaviourally equivalent then this can be proved using the axiom systems.

First we study (strong) hyperequivalence, then weak hypercongruence, and finally we explore the symbolic open bisimulation equivalence from Section 4.4.1.

5.1 Strong Hyperequivalence

For the axiomatization of hyperequivalence we restrict the language to the finite fragment, i.e., we do not consider recursively defined agents. Below we give complete axiomatizations of hyperequivalence, both with and without mismatch. The mismatch operator adds expressiveness, and can be motivated from practical modelling experience, such as in Chapter 6, where an "if-then-else" construction (built from a match/mismatch pair) is often very useful. The naturalness of the mismatch operator can also be seen in axiomatizations of late and early bisimulation congruences in the π-calculus [PS95]. Nevertheless, we will see that the mismatch operator is not essential for axiomatizing hyperequivalence.

For the axiomatization of hyperequivalence we subsume the fact that the equivalence is a congruence. We also use some of the laws for structural congruence (see Table 5.1). The axioms for hyperequivalence in the fusion calculus are given in Table 5.2, and in Table 5.3 we present some derived rules (whose names start with **D**).

Summation		
S1	$P + \mathbf{0} = P$	
S2	$P + Q = Q + P$	
S3	$P + (Q + R) = (P + Q) + R$	
Scope		
R0	$(x)\mathbf{0} = \mathbf{0}$	
R1	$(x)(y)P = (y)(x)P$	
R2	$(x)(P + Q) = (x)P + (x)Q$	
Match and Scope		
RM1	$(x)[y = z]P = [y = z](x)P$	if $x \neq y, x \neq z$

Table 5.1: Axioms from structural congruence.

Summation		
S4	$P + P = P$	
Match		
M1	$\tilde{M}P = \tilde{N}P$	if $\tilde{M} \Leftrightarrow \tilde{N}$
M2	$[x = y]P = [x = y](P\{x/y\})$	
M3	$MP + MQ = M(P + Q)$	
M4	$[x \neq x]P = \mathbf{0}$	
M5	$P = [x = y]P + [x \neq y]P$	
Scope		
R3	$(x)\alpha \,.\, P = \alpha \,.\, (x)P$	if $x \notin \mathrm{n}(\alpha)$
R4	$(x)\alpha \,.\, P = \mathbf{0}$	if x is the subject of α
Match and Scope		
RM2	$(x)[x = y]P = \mathbf{0}$	if $x \neq y$
Fusion		
F1	$\varphi \,.\, P = \varphi \,.\, [x = y]P$	if $x \,\varphi\, y$
F2	$(z)\varphi \,.\, P = \varphi\backslash z \,.\, P$	if $z \notin \mathrm{fn}(P)$
Expansion		
E	for $P \equiv \Sigma_i M_i(\tilde{x}_i)\alpha_i.P_i$, $Q \equiv \Sigma_j N_j(\tilde{y}_j)\beta_j.Q_j$, $P \mid Q = \sum_i M_i(\tilde{x}_i)\alpha_i \,.\, (P_i \mid Q) + \sum_j N_j(\tilde{y}_j)\beta_j \,.\, (P \mid Q_j) + \sum_{\alpha_i \mathrm{opp} \beta_j} M_i N_j(\tilde{x}_i\,\tilde{y}_j)[u_i = v_j]\{\tilde{z}_i = \tilde{w}_j\} \,.\, (P_i \mid Q_j)$ where $\alpha_i \mathrm{opp} \beta_j$ means $\alpha_i \equiv \overline{u_i}\tilde{z}_i$ and $\beta_j \equiv v_j\tilde{w}_j$ or vice versa, and $\lvert\tilde{z}_i\rvert = \lvert\tilde{w}_j\rvert$.	

Table 5.2: Axioms for hyperequivalence.

Match		
DM1	$[x=x]P = P$	
DM2	$[x=y]\alpha\,.\,P = [x=y]\alpha\,.\,[x=y]P$	
DM3	$\tilde{M}P = \tilde{M}(P\sigma)$	for σ agreeing with $\tilde{M}$
DM4	$M\mathbf{0} = \mathbf{0}$	
DM5	$MP + P = P$	
Match and Scope		
DRM1	$(x)[y \neq z]P = [y \neq z](x)P$	if $x \neq y, x \neq z$
DRM2	$(x)[x \neq y]P = (x)P$	if $x \neq y$
Fusion		
DF1	$\varphi\,.\,P = \varphi\,.\,(P\sigma)$	where σ agrees with φ
DF2	$(z)\varphi\,.\,P = \varphi\backslash z\,.\,(P\{w/z\})$	if $z\varphi w$ and $z \neq w$

Table 5.3: Derived rules.

Of the derived rules, **DM1** follows from $P \overset{\mathbf{M5}}{=} [x=x]P + [x \neq x]P \overset{\mathbf{M4}}{=} [x=x]P + \mathbf{0} \overset{\mathbf{S1}}{=} [x=x]P$; **DM2** follows from $[x=y]\alpha\,.\,P \overset{\mathbf{M2}}{=} [x=y](\alpha\,.\,P)\{x/y\} \overset{\mathbf{DM1}}{=} [x=y](\alpha\,.\,[x=y]P)\{x/y\} \overset{\mathbf{M2}}{=} [x=y]\alpha\,.\,[x=y]P$; **DM3** is derivable because we can write $\tilde{M}$ with the matches closest to P, and then apply **M2** repeatedly. The only worry is if the accumulated substitution becomes ambiguous, e.g. if $\{x/y\}\{z/y\}$ occurs. Since σ agrees with $\tilde{M}$, this means that $\tilde{M} \Rightarrow [x=y]$ and $\tilde{M} \Rightarrow [z=y]$, but then $\tilde{M} \Rightarrow [x=z]$ as well, and a corresponding substitution must occur in σ. **DM4** follows from $M\mathbf{0} \overset{\mathbf{M4}}{=} M[x \neq x]\mathbf{0} \overset{\mathbf{M1}}{=} [x \neq x]\mathbf{0} \overset{\mathbf{M4}}{=} \mathbf{0}$. **DM5** can be derived from $P \overset{\mathbf{M5}}{=} MP + \neg MP \overset{\mathbf{S4}}{=} (MP+MP) + \neg MP \overset{\mathbf{S3}}{=} MP + (MP + \neg MP) \overset{\mathbf{M5}}{=} MP + P$, where $\neg[x=y] \overset{def}{=} [x \neq y]$ and $\neg[x \neq y] \overset{def}{=} [x=y]$.

The derived rules **DRM** for match and scope correspond to the mismatch cases of the axioms **RM**, which apply to matches. **DRM1** is derived along the lines of [PS95], using **R2,M3,M4,M5,RM1** and the axioms for sum and congruence. **DRM2** follows from $(x)P \overset{\mathbf{M5,R1}}{=} (x)[x=y]P + (x)[x \neq y]P \overset{\mathbf{RM2,S1}}{=} (x)[x \neq y]P$. Finally, **DF1** follows from **F1** and **M2**, while **DF2** follows from **F2** and **DF1**.

It is interesting to note that all axioms except the two for fusion are well known from previous axiomatizations of the π-calculus. For example, all the axioms from Sangiorgi's axiomatization of open bisimulation equivalence for the π-calculus [San96c] hold for hyperequivalence in the fusion calculus, except the axiom treating restriction and the weaker form of congruence wrt input prefix. Please note also that our expansion law **E** is syntactically simpler than the corresponding laws for π-calculus equivalences.

Distinction relations between names have been used in axiomatizing π-calculus equivalences in [MPW92, San96c], but are not needed in our axiomatization of fusion calculus. In both calculi **RM2**, **DRM2** and **DM2** are valid, saying that a new name (by Scope or Restriction) is not equal to another name in the immediately following action (**RM2,DRM2**), and that if two names are equal they stay equal (**DM2**). However, consider again the π-calculus law

$$[x \neq y]\alpha \,.\, P \quad = \quad [x \neq y]\alpha \,.\, [x \neq y]P \tag{5.1}$$

This law holds in late and early π-calculus equivalences [PS95]: If $x \neq y$ then these names will continue to be distinct within the agent, so inserting an extra test for inequality is harmless. In the fusion calculus this law is invalid since the agent may be a component in a parallel composition where another agent may fuse x and y; so even if $x \neq y$ holds initially it may not hold after α.

In the rest of this section we first present the completeness proof for the axiom system of tables 5.1 and 5.2, and then give an axiom system for fusion calculus without mismatch, along with a completeness proof for that system.

We begin with a basic definition and one Lemma from [PS95].

Definition 41 *A match sequence $\tilde{M}$ is* complete on *a set of names V if for some equivalence relation $\mathcal{R}$ on V, called the equivalence relation* corresponding *to $\tilde{M}$, it holds that $\tilde{M} \Rightarrow [x = y]$ iff $x \,\mathcal{R}\, y$, and $\tilde{M} \Rightarrow [x \neq y]$ iff $\neg(x \,\mathcal{R}\, y)$*

A complete match sequence $\tilde{M}$ is maximally consistent *in the sense that adding something not already implied by $\tilde{M}$ makes $\tilde{M}$ inconsistent.*

Lemma 5.1 [PS95] *Let V be a set of names and let $\tilde{M}$ be complete on V.*

1. *If $\tilde{N}$ is another match sequence with names in V, then either $\tilde{M}\tilde{N}$ is unsatisfiable or $\tilde{M}\tilde{N} \Leftrightarrow \tilde{M}$.*

2. *If $\tilde{N}$ is another match sequence complete on V such that $\tilde{M}$ and $\tilde{N}$ both agree with the same substitution σ, then $\tilde{M} \Leftrightarrow \tilde{N}$.*

Lemma 5.2 $\sum_{i \in I} \tilde{M}_i P = P$, *if each $\tilde{M}_i$ complete on* $\mathrm{fn}(P)$ *and all equivalence relations of* $\mathrm{fn}(P)$ *are in $\{\tilde{M}_i : i \in I\}$.*

Proof: Use **M1** to rearrange match sequences, **M5** to remove match/mismatch pairs and **S** to rearrange sums. $\square$

For completeness we only consider the finite fragment of the fusion calculus, i.e., we do not consider recursively defined agents. The completeness of our axiom system says that if two agents are hyperequivalent, we can prove this using our axioms. We show that this is true in a standard way: first we define a head normal form for agents, and show that any agent can be written on this form. We can then find exactly which part of one agent that was used to simulate the transition of the other. Let $\mathcal{H}$ be the set of axioms in tables 5.1 and 5.2. We write $\vdash_{\mathcal{H}} P = Q$ if P and Q can be proven equal from $\mathcal{H}$.

Definition 42 *An agent P is in* head normal form (HNF) on V *(a finite set of names) if P is on the form*

$$\sum_{i\in I} \tilde{M}_i(\tilde{x}_i)\alpha_i \,.\, P_i$$

where for all i, $\tilde{x}_i \cap V = \emptyset$, $\tilde{x}_i \subseteq \mathrm{obj}(\alpha_i)$ and $\tilde{M}_i$ is complete on V.

Definition 43 *The* depth *of an agent P, $\mathrm{d}(P)$, is defined inductively as follows:* $\mathrm{d}(\mathbf{0}) = 0$, $\mathrm{d}(\alpha\,.\,P) = 1 + \mathrm{d}(P)$, $\mathrm{d}((\tilde{x})P) = \mathrm{d}(MP) = \mathrm{d}(P)$, $\mathrm{d}(P \mid Q) = \mathrm{d}(P) + \mathrm{d}(Q)$, $\mathrm{d}(P + Q) = \max(\mathrm{d}(P), \mathrm{d}(Q))$.

Lemma 5.3 *For all agents P and finite V such that $\mathrm{fn}(P) \subseteq V$, there is an agent H such that $\mathrm{d}(H) \leq \mathrm{d}(P)$, H is in HNF on V, and $\vdash_{\mathcal{H}} P = H$.*
PROOF: By induction on the structure of P.

1. $P \equiv \mathbf{0}$ is in HNF.

2. $P \equiv Q + R$ just requires induction.

3. $P \equiv \alpha\,.\,P'$ (where α is free):

   For each $i \in I$, let $\tilde{M}_i$ be complete on V, together corresponding to all possible equivalence classes of V, so intuitively $\vee_{i\in I}\tilde{M}_i \Leftrightarrow True$. Then by Lemma 5.2
   $$P = \sum_{i\in I} \tilde{M}_i\alpha\,.\,P'$$
   which is in HNF, and the depth is equal to $\mathrm{d}(P)$.

4. $P \equiv MP'$

   By induction $P' = \sum_{j\in J} \tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j$, with $\mathrm{d}(P') \leq \mathrm{d}(P)$, so
   $$P = M(\sum_{j\in J} \tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j) \stackrel{\mathbf{M3}}{=} \sum_{j\in J} M\tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j = \sum_{i\in I} \tilde{M}_i(\tilde{y}_i)\alpha_i\,.\,P'_i$$
   for $I \subseteq J$, using Lemma 5.1(1), with depth $\leq \mathrm{d}(P)$. If J is empty, $P = \mathbf{0}$ by **DM4**.

5. $P \equiv (x)P'$

   Again, $P' = \sum_{j\in J} \tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j$, with $\mathrm{d}(P') \leq \mathrm{d}(P)$, and
   $$P = (x)\sum_{j\in J} \tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j \stackrel{\mathbf{R2}}{=} \sum_{j\in J}(x)\tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j$$
   If J is empty, $P = \mathbf{0}$ by **R0**. Otherwise, each summand $(x)\tilde{M}_j(\tilde{y}_j)\alpha_j\,.\,P'_j$, call it R, can be reduced and rewritten to HNF, without increasing the depth, by **RM** and **R** axioms.

(a) $R \overset{\mathbf{RM2}}{=} \mathbf{0}$, if $[x = y]$ occurs in $\tilde{M}_j$ for any y.

(b) $R \overset{\mathbf{RM1,DRM1,DRM2}}{=} \tilde{M}'_j(x)(\tilde{y}_j)\alpha_j \,.\, P'_j$, where $\tilde{M}'_j$ is $\tilde{M}_j$ without any mismatches $[x \neq y]$ for any y. If $x \in \mathrm{n}(\alpha_j)$ but $x \neq \mathrm{subj}(\alpha_j)$, we have a HNF already. If $x \notin \mathrm{n}(\alpha_j)$, we use **R1** and **R3** to push the scoping over the prefix, arriving at $\tilde{M}'_j(x)(\tilde{y}_j)\alpha_j \,.\, (x)P'_j$, which is in HNF. Otherwise, if $x = \mathrm{subj}(\alpha_j)$, we have $\tilde{M}'_j(\tilde{y}_j)\mathbf{0}$ by **R4**, and using **DM4** and **R0** repeatedly, we get $R = \mathbf{0}$.

6. $P \equiv Q \mid R$. Apply **E**, which does not increase the depth, and induction.
$\square$

Theorem 5.4 *$P \sim Q$ iff $\vdash_{\mathcal{H}} P = Q$.*
PROOF: That $\vdash_{\mathcal{H}} P = Q$ implies $P \sim Q$ is easily verified by exhibiting the appropriate bisimulations. The other implication is more interesting:

By Lemma 5.3 we can assume P and Q to be in HNF on $\mathrm{fn}(P, Q)$. The proof is by induction on the depth of $P + Q$, showing that each summand of P is provably equal to a summand of Q.

If the depth is 0, $P \equiv Q \equiv \mathbf{0}$ and we are done. Otherwise, let $\tilde{M}(\tilde{x})\alpha \,.\, P'$ be a summand of P, σ a substitution agreeing with $\tilde{M}$, and $\sigma(w) = w$ for all $w \notin \mathrm{fn}(P, Q)$. Then $P\sigma \xrightarrow{((\tilde{x})\alpha)\sigma} P'\sigma$. By alpha-conversion $\tilde{x} \cap \mathrm{fn}(Q) = \emptyset$, and by $P \sim Q$, $Q\sigma \xrightarrow{((\tilde{x})\alpha)\sigma} Q''$, and

$$P'\sigma\rho = Q''\rho \qquad \text{by induction} \tag{5.2}$$

where ρ is a substitutive effect of $\alpha\sigma$.

By alpha-conversion, we can write the summand of Q used to simulate the action of $P\sigma$ as $\tilde{N}(\tilde{x})\beta \,.\, Q'$. Then $\tilde{N}$ agrees with σ and $(\tilde{N}(\tilde{x})\beta \,.\, Q')\sigma \xrightarrow{((\tilde{x})\beta)\sigma} Q'\sigma$, so $Q'' \equiv Q'\sigma$ and $((\tilde{x})\alpha)\sigma = ((\tilde{x})\beta)\sigma$, which implies

$$\alpha\sigma = \beta\sigma \tag{5.3}$$

since σ is identity for names not free in P, Q.

$\tilde{M}$ and $\tilde{N}$ are complete, so by Lemma 5.1(2) we know $\tilde{M} \Leftrightarrow \tilde{N}$. Since σ agrees with $\tilde{M}$

$$\tilde{M}(\tilde{x})\alpha \,.\, P' \overset{\mathbf{DM3}}{=} \tilde{M}((\tilde{x})\alpha \,.\, P')\sigma$$

$$\begin{aligned}
\tilde{M}((\tilde{x})\alpha \,.\, P')\sigma \quad &\overset{(\mathbf{DF1})}{=} \quad \tilde{M}(\tilde{x})(\alpha\sigma) \,.\, (P'\sigma\rho) \\
&\overset{(5.2)}{=} \quad \tilde{M}(\tilde{x})(\alpha\sigma) \,.\, (Q'\sigma\rho) \\
&\overset{(\mathbf{DF1})}{=} \quad \tilde{M}(\tilde{x})(\alpha\sigma) \,.\, (Q'\sigma) \\
&\overset{(5.3)}{=} \quad \tilde{M}(\tilde{x})(\beta\sigma) \,.\, (Q'\sigma) \\
&\overset{\mathbf{M1}}{=} \quad \tilde{N}(\tilde{x})(\beta\sigma) \,.\, (Q'\sigma)
\end{aligned}$$

$$
\begin{aligned}
&= \tilde{N}((\tilde{x})\beta \,.\, Q')\sigma \\
&\overset{\mathbf{DM3}}{=} \tilde{N}(\tilde{x})\beta \,.\, Q'
\end{aligned}
$$

where the steps marked (**DF1**) are valid either by the axiom **DF1** or simply because ρ is the identity substitution.

Thus we have proven that each summand of P is provably equal to a summand of Q, and then it suffices to use the **S** rules to infer that P and Q are provably equal. □

5.1.1 The Subcalculus without Mismatch

By dropping axioms **M4** and **M5**, and promoting **DM5** to an axiom, we get an axiomatization of hyperequivalence for the fusion calculus without mismatch, which can be proved complete along the lines of [San96c].

Call the new set of axioms $\mathcal{M}$. We proceed to prove the completeness of this axiom system. Because we no longer have mismatch, we cannot use the HNF defined in Definition 42. Instead we use a "head-only" variant of Sangiorgi's *strong normal form*.

Definition 44 *An agent P is in* mismatch-free head normal form (mHNF) *if P is on the form $\sum_{i\in I} \tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i$, where*

1. $\forall i : \tilde{x_i} \cap \mathrm{fn}(P) = \emptyset$, *and* $\tilde{x_i} \subseteq \mathrm{obj}(\alpha_i)$
2. *if* $i \neq j$ *then* $\tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i \not\sim \tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i + \tilde{M}_j(\tilde{x_j})\alpha_j \,.\, P_j$

In the following, we write $\sigma_{\tilde{M}}$ for a substitution which agrees with $\tilde{M}$ and which is identity for all names not in $\tilde{M}$.

We know that

$$(\tilde{N}(\tilde{x})\alpha \,.\, P)\sigma_{\tilde{M}} \xrightarrow{(\tilde{x})\alpha\sigma_{\tilde{M}}} P\sigma_{\tilde{M}} \quad \text{iff } \tilde{M} \Rightarrow \tilde{N} \tag{5.4}$$

since $\tilde{M} \Rightarrow \tilde{N}$ iff $\sigma_{\tilde{M}}$ agrees with $\tilde{N}$, and we bear in mind that $\tilde{M} \Rightarrow \tilde{N}$ iff $\tilde{M}\tilde{N} \Leftrightarrow \tilde{M}$.

Lemma 5.5 *For all mismatch-free agents P there is an agent H such that* $\mathrm{d}(H) \leq \mathrm{d}(P)$*, H is in mHNF, and* $\vdash_{\mathcal{M}} P = H$.

Theorem 5.6 *If P and Q contain no mismatch operators, then $P \sim Q$ iff* $\vdash_{\mathcal{M}} P = Q$.

We prove Lemma 5.5 and Theorem 5.6 by simultaneous induction over the depth of agents. The proofs are given separately, one using the other for strictly smaller depths.

Proof of Lemma 5.5: By induction on the depth of P. The base case, $P \equiv \mathbf{0}$, is trivial.

For the induction case, we write P as $P' \equiv \sum_{i\in I} \tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i$ (using **E** in case P is a parallel composition, distributing match and scope by **R2** and **M3**, pushing scope inward by **R3**). The first requirement on a mHNF is handled by alpha-conversion. We need to show that if Q and R are two summands of P' such that $Q \sim Q + R$, then $\vdash_{\mathcal{M}} Q = Q + R$, and thus we can remove R from P'.

Assume $Q \sim Q + R$. Let $R \equiv \tilde{M}(\tilde{x})\alpha \,.\, R_1$, and write Q as $\tilde{N}(\tilde{x})\beta \,.\, Q_1$. Since hyperequivalence is closed under substitution, $Q\sigma_{\tilde{M}} \sim Q\sigma_{\tilde{M}} + R\sigma_{\tilde{M}}$, and then

$$
\begin{array}{rcll}
(Q+R)\sigma_{\tilde{M}} & \xrightarrow{(\tilde{x})\alpha\sigma_{\tilde{M}}} & R_1\sigma_{\tilde{M}} & \\
Q\sigma_{\tilde{M}} & \xrightarrow{(\tilde{x})\beta\sigma_{\tilde{M}}} & Q_1\sigma_{\tilde{M}} & \\
\alpha\sigma_{\tilde{M}} & = & \beta\sigma_{\tilde{M}} & (5.5) \\
R_1\sigma_{\tilde{M}}\rho & \sim & Q_1\sigma_{\tilde{M}}\rho & (5.6) \\
\tilde{M}\tilde{N} & \stackrel{(5.4)}{\Leftrightarrow} & \tilde{M} & (5.7)
\end{array}
$$

where the substitution ρ is a substitutive effect of $\alpha\sigma_{\tilde{M}}$, and since $R_1\sigma_{\tilde{M}}\rho$ and $Q_1\sigma_{\tilde{M}}\rho$ can be written in mHNF by induction, we can use Theorem 5.6, in conjunction with (5.6) to get

$$R_1\sigma_{\tilde{M}}\rho \quad = \quad Q_1\sigma_{\tilde{M}}\rho \tag{5.8}$$

Now,

$$
\begin{array}{rcl}
R & \equiv & \tilde{M}(\tilde{x})\alpha \,.\, R_1 \\
& \stackrel{\mathbf{DM3}}{=} & \tilde{M}(\tilde{x})\alpha\sigma_{\tilde{M}} \,.\, R_1\sigma_{\tilde{M}} \\
& \stackrel{(\mathbf{DF1})}{=} & \tilde{M}(\tilde{x})\alpha\sigma_{\tilde{M}} \,.\, R_1\sigma_{\tilde{M}}\rho \\
& \stackrel{(5.8)}{=} & \tilde{M}(\tilde{x})\alpha\sigma_{\tilde{M}} \,.\, Q_1\sigma_{\tilde{M}}\rho \\
& \stackrel{(\mathbf{DF1})}{=} & \tilde{M}(\tilde{x})\alpha\sigma_{\tilde{M}} \,.\, Q_1\sigma_{\tilde{M}} \\
& \stackrel{(5.5)}{=} & \tilde{M}(\tilde{x})\beta\sigma_{\tilde{M}} \,.\, Q_1\sigma_{\tilde{M}} \\
& \stackrel{\mathbf{DM3}}{=} & \tilde{M}(\tilde{x})\beta \,.\, Q_1 \\
& \stackrel{(5.7)}{=} & \tilde{M}\tilde{N}(\tilde{x})\beta \,.\, Q_1 \\
& \equiv & \tilde{M}Q
\end{array}
$$

where again the steps marked (**DF1**) are valid either by **DF1** or because ρ is the identity substitution.

So, $Q + R = Q + \tilde{M}Q \stackrel{\mathbf{DM5}}{=} Q$, and the term R can be dropped from P'. □

Given this result, we can concentrate the proof of completeness on agents in mHNF.

Proof of Theorem 5.6: That $\vdash_{\mathcal{M}} P = Q$ implies $P \sim Q$ is easily verified by exhibiting the appropriate bisimulations. We study the other implication in more detail.

The proof goes by induction on $\mathrm{d}(P + Q)$. The base case is trivial. In the induction case, we show that each summand of P (which can be written in mHNF) can be transformed into a summand of Q (also written in mHNF).

Let $P_1 \equiv \tilde{M}(\tilde{x})\alpha\,.\,P_1'$ be a summand of P such that $P\sigma_{\tilde{M}} \xrightarrow{(\tilde{x})\alpha\sigma_{\tilde{M}}} P_1'\sigma_{\tilde{M}}$. Let $Q_1 \equiv \tilde{N}(\tilde{x})\beta\,.\,Q_1'$ be the summand of Q simulating this transition. Then $\tilde{M} \Rightarrow \tilde{N}$, and $\sigma_{\tilde{M}}$ agrees with $\tilde{N}$, so

$$\begin{array}{rcll} \sigma_{\tilde{M}} & = & \sigma_{\tilde{N}}\rho \quad \text{for some } \rho & (5.9) \\ Q\sigma_{\tilde{M}} & \xrightarrow{(\tilde{x})\beta\sigma_{\tilde{M}}} & Q_1'\sigma_{\tilde{M}} & \\ \alpha\sigma_{\tilde{M}} & = & \beta\sigma_{\tilde{M}} & (5.10) \\ P_1'\sigma_{\tilde{M}}\rho & \sim & Q_1'\sigma_{\tilde{M}}\rho & (5.11) \end{array}$$

where ρ is a substitutive effect of $\alpha\sigma_{\tilde{M}}$, and using Lemma 5.5, by induction

$$P_1'\sigma_{\tilde{M}}\rho \quad = \quad Q_1'\sigma_{\tilde{M}}\rho \tag{5.12}$$

If $\tilde{M} \Leftrightarrow \tilde{N}$, we can now proceed just like the end of the proof of Theorem 5.4. We already know that $\tilde{M} \Rightarrow \tilde{N}$, and show that $\neg(\tilde{N} \Rightarrow \tilde{M})$ leads to a contradiction.

We know that $P \sim Q$; consider the transition $Q\sigma_{\tilde{N}} \xrightarrow{(\tilde{x})\beta\sigma_{\tilde{N}}} Q_1'\sigma_{\tilde{N}}$. P cannot simulate this transition using the term P_1, since by assumption $\neg(\tilde{N} \Rightarrow \tilde{M})$. Instead, let $P_2 \equiv \tilde{L}(\tilde{x})\gamma\,.\,P_2'$ be the term used. Then

$$\begin{array}{rcll} Q\sigma_{\tilde{N}} & \xrightarrow{(\tilde{x})\beta\sigma_{\tilde{N}}} & Q_1'\sigma_{\tilde{N}} & \\ P\sigma_{\tilde{N}} & \xrightarrow{(\tilde{x})\gamma\sigma_{\tilde{N}}} & P_2'\sigma_{\tilde{N}} & \\ \beta\sigma_{\tilde{N}} & = & \gamma\sigma_{\tilde{N}} & (5.13) \\ Q_1'\sigma_{\tilde{N}}\rho' & \sim & P_2'\sigma_{\tilde{N}}\rho' & (5.14) \end{array}$$

where ρ' is a substitutive effect of $\beta\sigma_{\tilde{N}}$, and $\tilde{N} \Rightarrow \tilde{L}$.

We now show that in this case, $P_1 + P_2 \sim P_2$, which is in contradiction to the fact that P is in mHNF. That means that indeed $\tilde{N} \Rightarrow \tilde{M}$, which is what we need to finish the proof of the Theorem.

To prove $P_1 + P_2 \sim P_2$, we look at the case where P_2 makes a transition which is to be matched by P_1:

$$\begin{array}{rcl} (P_1 + P_2)\sigma_{\tilde{M}} & \xrightarrow{(\tilde{x})\alpha\sigma_{\tilde{M}}} & P_1'\sigma_{\tilde{M}} \\ P_2\sigma_{\tilde{M}} & \xrightarrow{(\tilde{x})\gamma\sigma_{\tilde{M}}} & P_2'\sigma_{\tilde{M}} \end{array}$$

The transition of P_2 is possible since $\tilde{M} \Rightarrow \tilde{N} \Rightarrow \tilde{L}$. From (5.10), (5.9) and (5.13), we see that

$$\alpha\sigma_{\tilde{M}} = \gamma\sigma_{\tilde{M}} \tag{5.15}$$

and we need to show that

$$P_1'\sigma_{\tilde{M}}\rho \sim P_2'\sigma_{\tilde{M}}\rho \tag{5.16}$$

By the soundness of the axiom system we can use rule **DF1** (if α is a fusion) to rewrite P_1 as $\tilde{M}(\tilde{x})\alpha\,.\,(P_1'\varrho)$ where ϱ is a substitutive effect of α, and similarly P_2 as $\tilde{L}(\tilde{x})\gamma\,.\,(P_2'\varrho')$ and Q_1 as $\tilde{N}(\tilde{x})\beta\,.\,(Q_1'\varrho'')$, where ϱ' a substitutive effect of γ and ϱ'' one of β. Then, using (5.10), (5.13) and (5.15), the trailing substitutions in (5.11), (5.14) and (5.16) are ineffectuous, and we know

$$\begin{array}{rcll}
P_2'\varrho'\sigma_{\tilde{N}} & \sim & Q_1'\varrho''\sigma_{\tilde{N}} & \text{by (5.14)} \qquad (5.17)\\
P_2'\varrho'\sigma_{\tilde{N}}\rho & \sim & Q_1'\varrho''\sigma_{\tilde{N}}\rho & \text{by (5.17) and Definition 27}\\
 & \equiv & Q_1'\varrho''\sigma_{\tilde{M}} & \text{by (5.9)}\\
 & \sim & P_1'\varrho\sigma_{\tilde{M}} & \text{by (5.11)}
\end{array}$$

which suffices for (5.16) to be true.

We have now proved that $\tilde{M} \Leftrightarrow \tilde{N}$, and can proceed just like the end of the proof of Theorem 5.4, from where we found the same fact from Lemma 5.1(2).

This concludes the proof of Theorem 5.6. □

5.2 Weak Hypercongruence

Algebraic laws for observation equivalence were first presented in [Mil80], and the first completeness proof for weak bisimulation equivalence is by Hennessy and Milner [HM85]. For the π-calculus weak early and late bisimulation have been axiomatized by Lin [Lin95a]. Axiomatizations are usually formulated in a set of so called tau-laws, τ being the name of the unobservable action in CCS. We will in this thesis keep the by now well established epithet "tau-law" even though the unobservable action in the fusion calculus is denoted **1**. Milner's original tau-laws can thus be written

$$\begin{array}{lrcl}
\textbf{T1} & \alpha\,.\,\mathbf{1}\,.\,P & = & \alpha\,.\,P\\
\textbf{T2} & P + \mathbf{1}\,.\,P & = & \mathbf{1}\,.\,P\\
\textbf{T3} & \alpha\,.\,(P + \mathbf{1}\,.\,Q) & = & \alpha\,.\,(P + \mathbf{1}\,.\,Q) + \alpha\,.\,Q
\end{array}$$

The main result in this section is to give a complete axiomatization of weak hypercongruence in the fusion calculus. It might be expected that adding the three laws **T1**-**T3** to an axiomatization of hyperequivalence would be enough, just as in CCS and in the π-calculus. However this turns out not to be the case, for two independent reasons.

The first reason has to do with the mismatch operator $[x \neq y]P$. In the fusion calculus we have fewer laws for it since (5.1) in Section 5.1 does not hold.

So it turns out that we actually need a stronger version of **T3**, involving an arbitrary sequence $\tilde{M}$ of mismatches

$$\textbf{T3a} \qquad \alpha\,.\,(P+\tilde{M}\mathbf{1}\,.\,Q) \quad = \quad \alpha\,.\,(P+\tilde{M}\mathbf{1}\,.\,Q)+\tilde{M}\alpha\,.\,Q$$

In the π-calculus all instances of **T3a** are derivable from the other axioms including **T3**; this is not the case in the fusion calculus.

The second reason is that weak hyperequivalence allows a fusion to be simulated by several smaller fusions with the same combined effect, as in (4.1) in Section 4.2. None of **T1**-**T3** caters for this. So we need an additional law. As in **T3a** it needs a sequence of mismatches.

$$\textbf{T3b} \qquad \varphi\,.\,(P+\tilde{M}\psi\,.\,Q) \quad = \quad \varphi\,.\,(P+\tilde{M}\psi\,.\,Q)+\tilde{M}(\varphi\wedge\psi)\,.\,Q$$

Here φ and ψ are fusion actions, $\varphi\wedge\psi$ is a fusion with the same effect as φ and ψ combined, and a side condition says that if $\tilde{M}\Rightarrow x\neq y$ then x and y may not be fused by φ (otherwise the law would be unsound).

T1	$\alpha\,.\,\mathbf{1}\,.\,P=\alpha\,.\,P$	
T2	$P+\mathbf{1}\,.\,P=\mathbf{1}\,.\,P$	
T3a	$\alpha\,.\,(P+\tilde{M}\mathbf{1}\,.\,Q)=$	$\alpha\,.\,(P+\tilde{M}\mathbf{1}\,.\,Q)+\tilde{M}\alpha\,.\,Q$
		if α is a communication
T3b	$\varphi\,.\,(P+\tilde{M}\psi\,.\,Q)=$	$\varphi\,.\,(P+\tilde{M}\psi\,.\,Q)+\tilde{M}(\varphi\wedge\psi)\,.\,Q$
		if $\forall x,y:\tilde{M}\Rightarrow x\neq y$ implies $\neg x\varphi y$

Table 5.4: Axioms for weak hypercongruence

Table 5.4 contains the additional axioms for weak hypercongruence. Let $\mathcal{W}$ be the axioms in tables 5.1,5.2 and 5.4.

Axioms **T1** and **T2** are direct counterparts of the familiar two first "tau-laws" from Milner. **T3a** generalizes to bound prefixes. For any free or bound prefix γ with $\mathrm{bn}(\gamma)\cap\mathrm{fn}(\tilde{M})=\emptyset$ we can use **R2, RM1, DRM1** to derive the more general form of **T3a** where a bound communication prefix replaces α.

Proposition 5.7 *The axioms in Table 5.4 are sound for* $\approx_+$.
PROOF: Directly from the definition of $\approx_+$. For **T3b**, note that if $\tilde{M}\sigma$ is true then so is $\tilde{M}\sigma\sigma_\varphi$, for σ_φ a substitutive effect of φ, because of the side condition. □

The completeness proof stretches over the rest of this section. In the following, σ will range over idempotent substitutions.

Lemma 5.8 *If* $P\sigma\xrightarrow{\gamma}P'$ *then* $P'\equiv P'\sigma$.
PROOF: By alpha-conversion we can assume γ does not bind names in $\mathrm{dom}(\sigma)$. By induction over transitions it is easy to establish that $\mathrm{fn}(P')\subseteq\mathrm{fn}(P\sigma)\cup\mathrm{bn}(\gamma)$.

So $\mathrm{fn}(P') \subseteq \mathrm{ran}(\sigma) \cup \mathrm{bn}(\gamma)$ and therefore $\sigma(x) = x$ for all names in $\mathrm{fn}(P')$ and the result follows. $\square$

For the sake of brevity, in the rest of this section we will use the derived bound prefix $((\tilde{x})a\tilde{z}) \,.\, P$ to mean $(\tilde{x})a\tilde{z} \,.\, P$ when $\tilde{x} \subseteq \tilde{z}$ and $a \notin \tilde{x}$, and let γ range over any (free or bound) prefix.

Lemma 5.9 *Let P be in HNF on V, where $\mathrm{fn}(P) \subseteq V$. If $P\sigma \xrightarrow{\gamma} P'$ then $\vdash_{\mathcal{W}} P = P + \tilde{M}\gamma \,.\, P'$ where $\tilde{M}$ agrees with σ and is complete on V.*

Proof: Since P is in HNF it has a summand $\tilde{N}\gamma' \,.\, Q$ such that $(\tilde{N}\gamma' \,.\, Q)\sigma \xrightarrow{\gamma} P'$, for $\tilde{N}$ complete on V and agreeing with σ. So $\gamma = \gamma'\sigma$ and $P' = Q\sigma$. So by **DM3**, $\vdash_{\mathcal{W}} \tilde{N}\gamma' \,.\, Q = \tilde{N}(\gamma' \,.\, Q)\sigma \equiv \tilde{N}\gamma \,.\, P'$. Now $\tilde{M}$ and $\tilde{N}$ are complete on V and agree with σ, so $\tilde{M} \Leftrightarrow \tilde{N}$. So by **M1**, $\vdash_{\mathcal{W}} \tilde{N}\gamma \,.\, P' = \tilde{M}\gamma \,.\, P'$. So by **S4**, $\vdash_{\mathcal{W}} P = P + \tilde{M}\gamma \,.\, P$. $\square$

Lemma 5.10 (Saturation lemma) *Let P be in HNF on V, where $\mathrm{fn}(P) \subseteq V$. If $P\sigma \overset{\gamma}{\Longrightarrow}_+ P'$ then $\vdash_{\mathcal{W}} P = P + \tilde{M}\gamma \,.\, P'$ where $\tilde{M}$ agrees with σ and is complete on V.*

Proof: By induction on the depth of P. There are four cases for $P\sigma \overset{\gamma}{\Longrightarrow}_+ P'$, the first of which also covers the base of the induction.

Case 1 $P\sigma \xrightarrow{\gamma} P'$. The result is immediate from Lemma 5.9.

Case 2 $P\sigma \xrightarrow{\gamma} Q$ and $Q \overset{\mathbf{1}}{\Longrightarrow}_+ P'$, where γ is a communication (γ a fusion is handled by Case 4 below). Let $\tilde{M}$ agree with σ and be complete on V. By Lemma 5.9 we get $\vdash_{\mathcal{W}} P = P + \tilde{M}\gamma \,.\, Q$. By alpha-conversion we can assume $\mathrm{fn}(\tilde{M}) \cap \mathrm{bn}(\gamma) = \emptyset$. By Lemma 5.8 $Q \equiv Q\sigma$. So $Q\sigma \overset{\mathbf{1}}{\Longrightarrow}_+ P'$. By induction then $\vdash_{\mathcal{W}} Q = Q + \tilde{N}\mathbf{1} \,.\, P'$. Since $\tilde{M}$ and $\tilde{N}$ are complete and agree with σ we have $\tilde{M} \Leftrightarrow \tilde{N}$. So by **M1**, $\vdash_{\mathcal{W}} Q = Q + \tilde{M}\mathbf{1} \,.\, P'$. In summary,

$$
\begin{array}{rcll}
\vdash_{\mathcal{W}} P & = & P + \tilde{M}\gamma \,.\, (Q + \tilde{M}\mathbf{1} \,.\, P') & (*) \\
& \overset{\mathbf{T3a}}{=} & P + \tilde{M}(\gamma \,.\, (Q + \tilde{M}\mathbf{1} \,.\, P') + \tilde{M}\gamma \,.\, P') & \\
& \overset{\mathbf{M3}}{=} & P + \tilde{M}\gamma \,.\, (Q + \tilde{M}\mathbf{1} \,.\, P') + \tilde{M}\tilde{M}\gamma \,.\, P' & \\
& \overset{\mathbf{M1}}{=} & P + \tilde{M}\gamma \,.\, (Q + \tilde{M}\mathbf{1} \,.\, P') + \tilde{M}\gamma \,.\, P' & \\
& \overset{(*)}{=} & P + \tilde{M}\gamma \,.\, P' &
\end{array}
$$

as required.

Case 3 $P\sigma \xrightarrow{\mathbf{1}} Q$ and $Q \overset{\gamma}{\Longrightarrow}_+ P'$, where γ is a communication (γ a fusion is handled by Case 4 below). Let $\tilde{M}$ agree with σ and be complete on V. By Lemma 5.9 we get $\vdash_{\mathcal{W}} P = P + \tilde{M}\mathbf{1} \,.\, Q$. By Lemma 5.8 $Q \equiv Q\sigma$. So $Q\sigma \overset{\gamma}{\Longrightarrow}_+ P'$. By induction then $\vdash_{\mathcal{W}} Q = Q + \tilde{N}\gamma \,.\, P'$. Since $\tilde{M}$

and $\tilde{N}$ are complete and agree with σ we have $\tilde{M} \Leftrightarrow \tilde{N}$. So by **M1**, $\vdash_{\mathcal{W}} Q = Q + \tilde{M}\gamma\,.\,P'$. In summary,

$$\begin{array}{rcl} \vdash_{\mathcal{W}} P & = & P + \tilde{M}\mathbf{1}\,.\,(Q + \tilde{M}\gamma\,.\,P') \\ & \stackrel{\mathbf{T2}}{=} & P + \tilde{M}(\mathbf{1}\,.\,(Q + \tilde{M}\gamma\,.\,P') + Q + \tilde{M}\gamma\,.\,P') \\ & \stackrel{\mathbf{S4}}{=} & P + \tilde{M}(\mathbf{1}\,.\,(Q + \tilde{M}\gamma\,.\,P') + Q + \tilde{M}\gamma\,.\,P' + \tilde{M}\gamma\,.\,P') \\ & \stackrel{\mathbf{T2}}{=} & P + \tilde{M}(\mathbf{1}\,.\,(Q + \tilde{M}\gamma\,.\,P') + \tilde{M}\gamma\,.\,P') \\ & \stackrel{\mathbf{M3}}{=} & P + \tilde{M}\mathbf{1}\,.\,(Q + \tilde{M}\gamma\,.\,P') + \tilde{M}\tilde{M}\gamma\,.\,P' \\ & \stackrel{\mathbf{M1}}{=} & P + \tilde{M}\mathbf{1}\,.\,(Q + \tilde{M}\gamma\,.\,P') + \tilde{M}\gamma\,.\,P' \\ & = & P + \tilde{M}\gamma\,.\,P' \end{array}$$

as required.

Case 4 $P\sigma \xrightarrow{\varphi} Q$, and $Q\sigma_\varphi \stackrel{\psi}{\Longrightarrow}_+ P'$ where σ_φ agrees with φ, and γ is a fusion with $\gamma = \varphi \wedge \psi$. Let $\tilde{M}$ agree with σ and be complete on V. By Lemma 5.9 we get $\vdash_{\mathcal{W}} P = P + \tilde{M}\varphi\,.\,Q$. By Lemma 5.8 $Q \equiv Q\sigma$. So $Q\sigma\sigma_\varphi \stackrel{\psi}{\Longrightarrow}_+ P'$. By induction then $Q \vdash_{\mathcal{W}} Q + \tilde{N}\psi\,.\,P'$ where $\tilde{N}$ agrees with $\sigma\sigma_\varphi$ and is complete on V. So,

$$\begin{array}{rcl} \vdash_{\mathcal{W}} P & = & P + \tilde{M}\varphi\,.\,(Q + \tilde{N}\psi\,.\,P') \\ & \stackrel{(*)}{=} & P + \tilde{M}\varphi\,.\,(Q + \tilde{N}\psi\,.\,P') + \tilde{M}(\varphi \wedge \psi)\,.\,P' \\ & = & P + \tilde{M}(\varphi \wedge \psi)\,.\,P' \\ & = & P + \tilde{M}\gamma\,.\,P' \end{array}$$

as required, where there remains to prove $(*)$. Let F be a sequence of matches corresponding to the fusion φ, i.e. $F \Rightarrow [x = y]$ iff $x\,\varphi\,y$. Let $\tilde{M}'$ be formed from $\tilde{M}$ by removing every mismatch $[x \neq y]$ where $x\,\varphi\,y$. It follows that $\tilde{M}'F \Leftrightarrow \tilde{N}$ since both are complete on V and agree with $\sigma\sigma_\varphi$. Now,

$$\begin{array}{rcl} \vdash_{\mathcal{W}} \varphi\,.\,(Q + \tilde{N}\psi\,.\,P') & \stackrel{\mathbf{M1}}{=} & \varphi\,.\,(Q + \tilde{M}'F\psi\,.\,P') \\ & \stackrel{\mathbf{DF1,DM1}}{=} & \varphi\,.\,(Q + \tilde{M}'\psi\,.\,P') \\ & \stackrel{\mathbf{T3b}}{=} & \varphi\,.\,(Q + \tilde{M}'\psi\,.\,P') + \tilde{M}'(\varphi \wedge \psi)\,.\,P' \\ & = & \varphi\,.\,(Q + \tilde{N}\psi\,.\,P') + \tilde{M}'(\varphi \wedge \psi)\,.\,P' \end{array}$$

Note that the side condition in **T3b** is fulfilled by construction of $\tilde{M}'$. Therefore,

$$\begin{array}{rcl} \vdash_{\mathcal{W}} \tilde{M}\varphi\,.\,(Q + \tilde{N}\psi\,.\,P') & = & \tilde{M}(\varphi\,.\,(Q + \tilde{N}\psi\,.\,P') + \tilde{M}'(\varphi \wedge \psi)\,.\,P') \\ & \stackrel{\mathbf{M3}}{=} & \tilde{M}\varphi\,.\,(Q + \tilde{N}\psi\,.\,P') + \tilde{M}\tilde{M}'\varphi \wedge \psi)\,.\,P' \\ & \stackrel{\mathbf{M1}}{=} & \tilde{M}\varphi\,.\,(Q + \tilde{N}\psi\,.\,P') + \tilde{M}(\varphi \wedge \psi)\,.\,P' \end{array}$$

as required, where the last step follows since all matches/mismatches in $\tilde{M}'$ are also in $\tilde{M}$.

This completes Case 4 and the proof of the lemma. □

Definition 45 *A HNF P on V is called a* full *HNF on V if it has all summands implied by Lemma 5.10.*

Proposition 5.11 *For any HNF on V there is a provably equivalent full HNF on V of no greater depth.*

PROOF: Just apply Lemma 5.10 repeatedly. Eventually the HNF becomes a full HNF since there are only a finite number of derivatives and a finite number of substitutions on V to consider. □

Lemma 5.12 $P \approx Q$ *iff* ($P \approx_+ Q$ *or* $P \approx_+ \mathbf{1}\,.\,Q$ *or* $\mathbf{1}\,.\,P \approx_+ Q$).

PROOF: Precisely as in [Mil89], Proposition 11 on page 156–157. □

Theorem 5.13 $P \approx_+ Q$ *iff* $\vdash_{\mathcal{W}} P = Q$.

PROOF: That $\vdash_{\mathcal{W}} P = Q$ implies $P \approx_+ Q$ is easily verified, and we concentrate on the other implication. The proof is by induction on the sum of the depths of P and Q. We can assume that P and Q are full HNFs on V, for $\mathrm{fn}(P) \cup \mathrm{fn}(Q) \subseteq V$, by Proposition 5.11. The base case $P \equiv Q \equiv \mathbf{0}$ is trivial. For the inductive step assume $P \approx_+ Q$. Let $\tilde{M}\gamma\,.\,P'$ be a summand of P. By alpha-conversion we can assume that γ does not bind any name in $\tilde{M}$. We shall prove that Q has a provably equivalent summand.

Let σ agree with $\tilde{M}$ and be complete on V. Then

$$P\sigma \xrightarrow{\gamma\sigma} P'\sigma$$

From $P \approx_+ Q$ we get that

$$Q\sigma \overset{\gamma\sigma}{\Longrightarrow}_+ Q''$$

where $P\sigma\sigma_\gamma \approx Q''\sigma_\gamma$ for a substitutive effect σ_γ of γ. Since Q is a full HNF, Q must have a summand

$$\tilde{N}\gamma'\,.\,Q'$$

such that $Q'\sigma\sigma_\gamma \equiv Q''\sigma_\gamma$ and $\gamma'\sigma = \gamma\sigma$ and σ agrees with $\tilde{N}$ and $\tilde{N}$ is complete on V. So $\tilde{M} \Leftrightarrow \tilde{N}$ and $P'\sigma\sigma_\gamma \approx Q'\sigma\sigma_\gamma$. Therefore,

$$\begin{array}{rcl} \vdash_{\mathcal{W}} \tilde{M}\gamma\,.\,P' & \overset{\mathbf{DM3}}{=} & \tilde{M}(\gamma\,.\,P')\sigma \\ & = & \tilde{M}(\gamma\,.\,P'\sigma)\sigma \\ & \overset{\mathbf{DF1}}{=} & \tilde{M}(\gamma\,.\,P'\sigma\sigma_\gamma)\sigma \end{array}$$

(where the last step is void if γ is a communication). We cannot immediately apply induction to $P'\sigma\sigma_\gamma$ since we only know $P'\sigma\sigma_\gamma \approx Q'\sigma\sigma_\gamma$, and not that they are hypercongruent. So we use Lemma 5.12 to consider three cases.

Tm3a	$\alpha\,.\,(P + \mathbf{1}\,.\,Q) = \alpha\,.\,(P + \mathbf{1}\,.\,Q) + \alpha\,.\,Q$
Tm3b	$\varphi\,.\,(P + \psi\,.\,Q) = \varphi\,.\,(P + \psi\,.\,Q) + (\varphi \wedge \psi)\,.\,Q$

Table 5.5: **T3**-laws for the fusion calculus without mismatch.

Case 1 $P'\sigma\sigma_\gamma \approx_+ Q'\sigma\sigma_\gamma$. Then by induction they are provably equal, so

$$\begin{array}{rcl}\vdash_{\mathcal{W}} \tilde{M}(\gamma\,.\,P'\sigma\sigma_\gamma)\sigma & = & \tilde{M}(\gamma\,.\,Q'\sigma\sigma_\gamma)\sigma \\ & = & \tilde{M}(\gamma\sigma\,.\,Q'\sigma\sigma_\gamma)\sigma \\ & = & \tilde{M}(\gamma'\sigma\,.\,Q'\sigma\sigma_\gamma)\sigma \\ & \stackrel{\mathbf{M1}}{=} & \tilde{N}(\gamma'\sigma\,.\,Q'\sigma\sigma_\gamma)\sigma \\ & = & \tilde{N}\gamma'\,.\,Q'\end{array}$$

Case 2 $P'\sigma\sigma_\gamma \approx_+ \mathbf{1}\,.\,Q'\sigma\sigma_\gamma$. We can now apply induction because the sum of the depths is one less than that for P and Q. So

$$\begin{array}{rcl}\vdash_{\mathcal{W}} \tilde{M}(\gamma\,.\,P'\sigma\sigma_\gamma)\sigma & = & \tilde{M}(\gamma\,.\,\mathbf{1}\,.\,Q'\sigma\sigma_\gamma)\sigma \\ & \stackrel{\mathbf{T1}}{=} & \tilde{M}(\gamma\,.\,Q'\sigma\sigma_\gamma)\sigma \\ & = & \tilde{N}\gamma'\,.\,Q'\end{array}$$

where the last equality is similar to Case 1 above.

Case 3 $\mathbf{1}\,.\,P'\sigma\sigma_\gamma \approx_+ Q'\sigma\sigma_\gamma$. This is symmetric to Case 2.

We have proved that each summand in P has a provably equal summand in Q. The converse is symmetric. **S1**-**S4** thus completes the induction and proof of the theorem, giving us $\vdash_{\mathcal{W}} P = Q$. ☐

5.2.1 The Subcalculus without Mismatch

In this section we consider the calculus without Mismatch, and let $\tilde{M}$ etc. range over sequences of Match operators.

For the weak hypercongruence it turns out that simpler versions of **T3a** and **T3b** suffice: there is no longer a need for the match sequences to be part of the axioms. The simpler versions are called **Tm3a** and **Tm3b** and are given in Table 5.5.

Let $\mathcal{MW}$ be the axioms **T1**, **T2**, **Tm3a** and **Tm3b** plus the axioms in $\mathcal{M}$. Then $\mathcal{MW}$ is easily seen to be sound for $\approx_+$. Note that α in **Tm3a** can be a fusion (that would be equivalent to **Tm3b** with $\psi = \mathbf{1}$). Again the completeness proof stretches over several lemmas and covers the rest of this section.

Lemma 5.14 *If* $P\sigma \xrightarrow{\gamma} P'$ *then* $\vdash_{\mathcal{MW}} P = P + \tilde{M}\gamma\,.\,P'$ *where* $\tilde{M}$ *agrees with* σ.

PROOF: By Lemma 5.5 we can assume that P is in mHNF. Suppose $P\sigma \xrightarrow{\gamma} P'$. Let $\tilde{M}$ agree with σ. Then P has a summand $\tilde{N}\gamma' . Q'$ such that $\tilde{M} \Rightarrow \tilde{N}$, this means that $\tilde{M} \Leftrightarrow \tilde{L}\tilde{N}$ for some $\tilde{L}$, and further $\gamma'\sigma = \gamma$ and $Q'\sigma \equiv P'$. So we have:

$$\begin{array}{rcl} \vdash_{\mathcal{MW}} P & \stackrel{\mathbf{S4}}{=} & P + \tilde{N}\gamma' . Q' \\ & \stackrel{\mathbf{DM5}}{=} & P + \tilde{N}\gamma' . Q' + \tilde{L}\tilde{N}\gamma' . Q' \\ & \stackrel{\mathbf{M1}}{=} & P + \tilde{N}\gamma' . Q' + \tilde{M}\gamma' . Q' \\ & \stackrel{\mathbf{S4}}{=} & P + \tilde{M}\gamma' . Q' \\ & \stackrel{\mathbf{DM3}}{=} & P + \tilde{M}\gamma'\sigma . Q'\sigma \\ & = & P + \tilde{M}\gamma . P' \end{array}$$

□

The proof of the saturation lemma (Lemma 5.10) does not carry over immediately since it relies on complete match sequences. Without mismatches we instead prove it as follows.

Lemma 5.15 (Saturation lemma without mismatch) *If* $P\sigma \stackrel{\gamma}{\Longrightarrow}_+ P'$ *then* $\vdash_{\mathcal{MW}} P = P + \tilde{M}\gamma . P'$ *where* $\tilde{M}$ *agrees with* σ.

PROOF: By induction on the depth of P. There are four cases for $P\sigma \stackrel{\gamma}{\Longrightarrow}_+ P'$, the first of which also covers the base of the induction.

Case 1 $P\sigma \xrightarrow{\gamma} P'$. The result is immediate from Lemma 5.14.

Case 2 $P\sigma \xrightarrow{\gamma} Q$ and $Q \stackrel{\mathbf{1}}{\Longrightarrow}_+ P'$, where γ is a communication (γ a fusion is handled by Case 4 below). Let $\tilde{M}$ agree with σ. By Lemma 5.14 we get that $\vdash_{\mathcal{MW}} P = P + \tilde{M}\gamma . Q$. By induction and **DM1**, $\vdash_{\mathcal{MW}} Q = Q + \mathbf{1} . P'$. So,

$$\begin{array}{rcl} \vdash_{\mathcal{MW}} P & = & P + \tilde{M}\gamma . (Q + \mathbf{1} . P') \\ & \stackrel{\mathbf{Tm3a}}{=} & P + \tilde{M}(\gamma . (Q + \mathbf{1} . P') + \gamma . P') \\ & \stackrel{\mathbf{M3}}{=} & P + \tilde{M}\gamma . (Q + \mathbf{1} . P') + \tilde{M}\gamma . P' \\ & = & P + \tilde{M}\gamma . P' \end{array}$$

Case 3 $P\sigma \xrightarrow{\mathbf{1}} Q$ and $Q \stackrel{\gamma}{\Longrightarrow}_+ P'$, where γ is a communication (γ a fusion is handled by Case 4 below). Let $\tilde{M}$ agree with σ. By Lemma 5.14 we get that $\vdash_{\mathcal{MW}} P = P + \tilde{M}\mathbf{1} . Q$. By induction, $\vdash_{\mathcal{MW}} Q = Q + \gamma . P'$. So,

$$\begin{array}{rcl} \vdash_{\mathcal{MW}} P & = & P + \tilde{M}\mathbf{1} . (Q + \gamma . P') \\ & \stackrel{\mathbf{T2}}{=} & P + \tilde{M}(\mathbf{1} . (Q + \gamma . P') + Q + \gamma . P') \\ & \stackrel{\mathbf{S4,T2}}{=} & P + \tilde{M}(\mathbf{1} . (Q + \gamma . P') + \gamma . P') \\ & \stackrel{\mathbf{M3}}{=} & P + \tilde{M}\mathbf{1} . (Q + \gamma . P') + \tilde{M}\gamma . P') \\ & = & P + \tilde{M}\gamma . P' \end{array}$$

Case 4 $P\sigma \xrightarrow{\varphi} Q$, and $Q\sigma_\varphi \overset{\psi}{\Longrightarrow}_+ P'$ where σ_φ agrees with φ, and γ is a fusion with $\gamma = \varphi \wedge \psi$. Let $\tilde{M}$ agree with σ. By Lemma 5.14 we get

$$\vdash_{\mathcal{MW}} P = P + \tilde{M}\varphi \,.\, Q \overset{\mathbf{DF1}}{=} P + \tilde{M}\varphi \,.\, Q\sigma_\varphi$$

Induction gives $\vdash_{\mathcal{MW}} Q\sigma_\varphi = Q\sigma_\varphi + \psi \,.\, P'$. So,

$$\begin{array}{rcl} \vdash_{\mathcal{MW}} P & = & P + \tilde{M}\varphi \,.\, (Q\sigma_\varphi + \psi \,.\, P') \\ & \overset{\mathbf{Tm3b}}{=} & P + \tilde{M}(\varphi \,.\, (Q\sigma_\varphi + \psi \,.\, P') + (\varphi \wedge \psi) \,.\, P') \\ & \overset{\mathbf{M3}}{=} & P + \tilde{M}\varphi \,.\, (Q\sigma_\varphi + \psi \,.\, P') + \tilde{M}(\varphi \wedge \psi) \,.\, P' \\ & = & P + \tilde{M}(\varphi \wedge \psi) \,.\, P' \\ & = & P + \tilde{M}\gamma \,.\, P' \end{array}$$

This completes the proof of the lemma. □

For the completeness proof we need a variant of mHNF which uses weak hypercongruence:

Definition 46
An agent P is in mismatch-free weak head normal form (mwHNF) *if P is on the form $\sum_{i\in I} \tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i$, where*

1. $\forall i : \tilde{x_i} \cap \mathrm{fn}(P) = \emptyset$*, and* $\tilde{x_i} \subseteq \mathrm{obj}(\alpha_i)$
2. *if $i \neq j$ then* $\tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i \not\approx_+ \tilde{M}_i(\tilde{x_i})\alpha_i \,.\, P_i + \tilde{M}_j(\tilde{x_j})\alpha_j \,.\, P_j$

Again we use the derived bound prefixes, so the terms in a mwHNF are on the form $\tilde{M}\gamma \,.\, P$.

Definition 47 *A* full *mwHNF is a mwHNF with the property that if $P\sigma \overset{\gamma}{\Longrightarrow}_+ P'$ then P has a summand $\tilde{M}\gamma' \,.\, Q'$ such that $\sigma \models \tilde{M}$, $\gamma'\sigma = \gamma$ and $Q'\sigma\sigma_\gamma \approx_+ P'\sigma_\gamma$.*

The following lemma and theorem are proved by a simultaneous induction on depth:

Lemma 5.16 *For all agents P there is an agent H such that* $\mathrm{d}(H) \leq \mathrm{d}(P)$*, H is in full mwHNF, and* $\vdash_{\mathcal{MW}} P = H$*.*

Theorem 5.17 *$P \approx_+ Q$ iff $\vdash_{\mathcal{MW}} P = Q$.*
Proof: Soundness is easily established, and we turn to completeness. We here only outline how the proof differs from previous completeness proofs. It uses induction on the depth of the agents involved. Lemma 5.16 uses Theorem 5.17 for agents of strictly smaller depth than the agents in the lemma. Theorem 5.17 uses Lemma 5.16 for agents of equal or smaller depth than the agents in the theorem.

For Lemma 5.16 first apply Lemma 5.15 repeatedly to P; there are only a finite number of substitutions that matter (those affecting $\mathrm{fn}(P)$) and a finite number of derivatives. The resulting agent P' has all terms implied by Lemma 5.15. But it may not be in full mwHNF because of condition 2 in the definition of mwHNF. Let Q and R be two summands in P' such that $Q \approx_+ Q + R$. We can then prove $\vdash_{\mathcal{MW}} Q = Q + R$ much as in the proof of Lemma 5.5 in Section 5.1.1. Thus we can repeatedly remove such summands R from P' until we gain a mwHNF. Obviously removing summands in that way will preserve the condition in the definition of a full mHNF.

Also the proof of Theorem 5.17 follows the proof of Theorem 5.6 in Section 5.1.1 closely. The differences here (and also in the proof that $\vdash_{\mathcal{MW}} Q = Q + R$ mentioned above) are the following. Obviously $\approx_+$ replaces $\sim$, and therefore simulating transitions are $\stackrel{\gamma}{\Longrightarrow}_+$, but by saturation we obtain that there are simulating transitions $\stackrel{\gamma}{\longrightarrow}$. Resulting derivatives are related by $\approx$ (rather than $\sim$) and we then use Lemma 5.12 to consider three cases. Any actions $\mathbf{1}$ introduced by that lemma will disappear by **T1**, just as in the proof of Theorem 5.4. □

5.2.2 A More Traditional T3

The law **T3b** (and similarly **Tm3b**) captures the circumstance when two fusions can be simulated by one. In the same way, in CCS and in the π-calculus, the tau-laws say when two actions can be simulated by one. In those calculi one of the actions has to be the unobservable action τ. It is therefore natural to ask what the effect would be if **T3b** were restricted to the case $\psi = \mathbf{1}$. Call this restricted form **T3b′**:

$$\textbf{T3b}' \quad \varphi\,.\,(P + \tilde{M}\mathbf{1}\,.\,Q) \;=\; \varphi\,.\,(P + \tilde{M}\mathbf{1}\,.\,Q) + \tilde{M}\varphi\,.\,Q \quad \text{if } \forall x, y : \tilde{M} \Rightarrow x \neq y \text{ implies } \neg x\,\varphi\,y$$

This is structurally more similar to **T3a**. In fact, by defining "$x\,\alpha\,y$" to never hold when α is a communication, **T3a** and **T3b′** can now easily be formulated as one law:

$$\textbf{T3}' \quad \alpha\,.\,(P + \tilde{M}\mathbf{1}\,.\,Q) \;=\; \alpha\,.\,(P + \tilde{M}\mathbf{1}\,.\,Q) + \tilde{M}\alpha\,.\,Q \quad \text{if } \forall x, y : \tilde{M} \Rightarrow x \neq y \text{ implies } \neg x\,\alpha\,y$$

Given the tau-laws of CCS and the π-calculus, the laws **T1**, **T2** and **T3′** might be thought of as the "traditional" laws of the fusion calculus. Let $\mathcal{W}'$ consist of these axioms plus the axioms for hyperequivalence in tables 5.1 and 5.2. We will here characterize the equivalence generated by $\mathcal{W}'$ coinductively, and demonstrate that it is strictly finer than weak hypercongruence.

For the purpose of this section, make the following changes in the previous definitions: In Definition 28, redefine the *weak transition* $\stackrel{\gamma}{\Longrightarrow}$ by the following:

> $P \stackrel{\gamma}{\Longrightarrow} Q$ means that for some $n \geq 0$, $P \stackrel{\gamma_1}{\longrightarrow} \circ \cdots \circ \stackrel{\gamma_n}{\longrightarrow} Q$ and $\gamma = \gamma_i$ for some i and $\gamma_j = \mathbf{1}$ for all $j \neq i$. We allow $n = 0$ and $\gamma = \mathbf{1}$, in other words $P \stackrel{\mathbf{1}}{\Longrightarrow} P$ holds for all P.

Note that the special case when γ is a fusion has disappeared. This has consequences for the definitions of weak hyperequivalence and weak hypercongruence, which with the amended definition are denoted $\approx'$ and $\approx'_+$. That these are smaller than $\approx$ and $\approx_+$ can be seen with a simple instance of **T3b**:

$$\varphi \,.\, \psi \,.\, P \not\approx' \varphi \,.\, \psi \,.\, P + (\varphi \wedge \psi) \,.\, P$$

since RHS $\xrightarrow{\varphi \wedge \psi} P$ can no longer be simulated by LHS. However, fusions still exhibit some absorptions that communications do not. For example,

$$\varphi \,.\, \varphi \,.\, P \approx' \varphi \,.\, P$$

holds by **DF2** and **T1**.

Theorem 5.18 $P \approx'_+ Q$ *iff* $\vdash_{\mathcal{W}'} P = Q$.
PROOF: Soundness is easily established. For completeness we only indicate where the proofs of Theorem 5.13 and its supporting lemmas change. The only significant change is in the proof of Lemma 5.10. Here Case 4 disappears because of the amended definition of $\stackrel{\gamma}{\Longrightarrow}$. On the other hand Case 2 and 3 must now also consider the case that γ is a fusion. For Case 3 this represents no problem, the demonstrated derivation works also for γ a fusion. For Case 2, if γ is a fusion we need to apply **T3b**$'$. This is proved exactly as in Case 4 with the additional requirement that $\psi = \mathbf{1}$ (since **T3b**$'$ is just the special case of **T3b** where $\psi = \mathbf{1}$). The rest of the completeness proof is unchanged. □

In the subcalculus without mismatch a similar effect is obtained by requiring $\psi = \mathbf{1}$ in **Tm3b**. Since **Tm3a** admits α to be a fusion this means that **Tm3b** can be dropped altogether. Let $\mathcal{MW}'$ be the axioms of $\mathcal{M}$ plus **T1**, **T2** and **Tm3b**. Note that these three laws correspond exactly to the three tau-laws of CCS!

Theorem 5.19 *In the subcalculus without mismatch,* $P \approx'_+ Q$ *iff* $\vdash_{\mathcal{MW}'} P = Q$.
PROOF: The proof is completely analogous to the proof of Theorem 5.18. Only Lemma 5.15 changes in that in case 4, $\psi = 1$ is now sufficient. So **Tm3a** suffices for this case. Cases 2 and 3 need to be strengthened to consider that γ is a fusion, but that is trivial. □

An overview of the axiom systems is shown in Figure 5.1.

5.3 Symbolic Open Bisimulation

To complete the picture of the π-calculus symbolic open bisimulation equivalence we defined in Section 4.4.1, we describe an axiomatization of symbolic open D-bisimilarity, based on the axiomatization of hyperequivalence in Section 5.1.

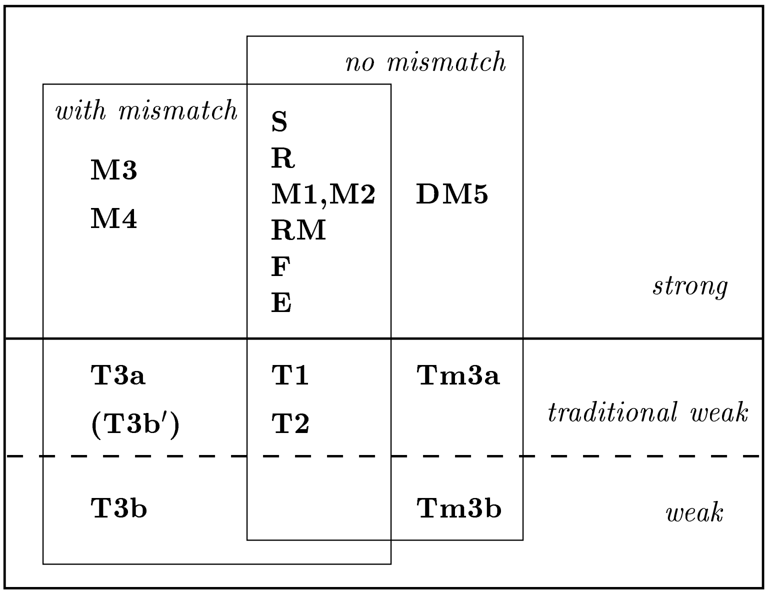

Figure 5.1: Overview of the axiom systems for the fusion calculus.

From Sangiorgi's axiomatization of open bisimulation equivalence (without mismatch) [San96c] we add axioms regarding distinctions (**P**,**W**), and a weak form of congruence for scope (**R5**). Our axiom **R5** incorporates Sangiorgi's two axioms for restriction and input (bound) prefix. The axiom **M1′** is changed from the **M1** axiom of hyperequivalence to take the distinction into account; here $\tilde{M} \wedge D$ stands for $\tilde{M}$ extended with mismatches corresponding to D, and $D \cup \tilde{M}^-$ for D extended corresponding to the mismatch part of $\tilde{M}$. **M1′** can now be used to derive rules such as

$$\begin{array}{rcll} (x)[x=y]P & = & \mathbf{0} & \\ (x)[x \neq y]P & = & (x)P & \\ [x \neq y]P & =_D & [x \neq y]Q & \text{if } P =_{D \cup \{(x,y)\}} Q \\ [x \neq y]\alpha\,.\,P & = & [x \neq y]\alpha\,.\,[x \neq y]P & \end{array}$$

We end up with the set of axioms in Table 5.6. Let $\mathcal{O}$ stand for this set. In the subcalculus f_π there are no fusion prefixes, so we omit the axioms regarding those, and rewrite the expansion axiom to avoid using fusion prefix there (this is possible since all objects of input prefixes are scoped and pairwise distinct); we name the new axiom **E′**. The axiom **RM2** from the hyperequivalence axiomatization, $(x)[x = y]P = \mathbf{0}$, is omitted since it is now derivable from **M1**, **R5** and **R0**. We take $P = Q$ to stand for $P =_\emptyset Q$ (so by **W**, $P =_D Q$ for any D).

Summation				
S1	$P+\mathbf{0}$	$=$	P	
S2	$P+Q$	$=$	$Q+P$	
S3	$P+(Q+R)$	$=$	$(P+Q)+R$	
S4	$P+P$	$=$	P	
Match				
M1′	$\tilde{M}P$	$=_D$	$\tilde{N}Q$	if $\tilde{M}\wedge D \Leftrightarrow \tilde{N}\wedge D$ and $P =_{D'} Q$ where $D' = D \cup \tilde{M}^-$
M2	$[x=y]P$	$=$	$[x=y](P\{x/y\})$	
M3	$MP+MQ$	$=$	$M(P+Q)$	
M4	$[x\neq x]P$	$=$	$\mathbf{0}$	
M5	P	$=$	$[x=y]P+[x\neq y]P$	
Scope				
R0	$(x)\mathbf{0}$	$=$	$\mathbf{0}$	
R1	$(x)(y)P$	$=$	$(y)(x)P$	
R2	$(x)(P+Q)$	$=$	$(x)P+(x)Q$	
R3	$(x)\alpha\,.\,P$	$=$	$\alpha\,.\,(x)P$	if $x\notin \mathrm{n}(\alpha)$
R4	$(x)\alpha\,.\,P$	$=$	$\mathbf{0}$	if x is the subject of α
Match and Scope				
RM1	$(x)[y=z]P$	$=$	$[y=z](x)P$	if $x\neq y, x\neq z$
Expansion				
E′	for $P\equiv \Sigma_i M_i(\tilde{x}_i)\alpha_i.P_i$, $Q\equiv \Sigma_j N_j(\tilde{y}_j)\beta_j.Q_j$, $P \mid Q \;=\; \sum_i M_i(\tilde{x}_i)\alpha_i\,.\,(P_i \mid Q) + \sum_j N_j(\tilde{y}_j)\beta_j\,.\,(P \mid Q_j) + \sum_{\alpha_i \mathrm{opp} \beta_j} M_iN_j(\tilde{x}_i)[u_i=v_j]\mathbf{1}\,.\,(P_i \mid Q_j\{\tilde{z}_i/\tilde{w}_j\})$ where $\alpha_i\mathrm{opp}\beta_j$ means $\alpha_i=\overline{u_i}\tilde{z}_i$ and $\beta_j=v_j\tilde{w}_j$ or vice versa, and $\lvert\tilde{z}_i\rvert=\lvert\tilde{w}_j\rvert$.			
Pruning and Weakening				
P	$[x=y]P$	$=_D$	$\mathbf{0}$	if $(x,y)\in D$
W	P	$=_D$	Q	if $P=_{D'}Q$ and $D'\subseteq D$
Scope and Distinction				
R5	$(x)P$	$=_{D-x}$	$(x)Q$	if $P=_D Q$

Table 5.6: Axioms of symbolic open D-bisimilarity.

The completeness of the axiom system is proved in the same fashion as Theorem 5.4. Here, we must also mind that the normal form and equivalence respect a distinction. We recall and extend some definitions:

Definition 48 *A match sequence $\tilde{M}$ is D-*complete on *a set of names V if it respects D and for some equivalence relation $\mathcal{R}$ on V, called the equivalence relation* corresponding to *$\tilde{M}$, it holds that $\tilde{M} \Rightarrow [x = y]$ iff $x \mathrel{\mathcal{R}} y$, and $\tilde{M} \Rightarrow [x \neq y]$ iff $\neg(x \mathrel{\mathcal{R}} y)$*

A D-complete match sequence $\tilde{M}$ is maximally consistent in the sense that adding something not already implied by $\tilde{M}$ makes $\tilde{M}$ inconsistent.

Lemma 5.20 *Let V be a set of names and let $\tilde{M}$ be D-complete on V.*

1. *If $\tilde{N}$ is another match sequence with names in V and respecting D, then either $\tilde{M}\tilde{N}$ is unsatisfiable or $\tilde{M}\tilde{N} \Leftrightarrow \tilde{M}$.*
2. *If $\tilde{N}$ is another match sequence D-complete on V such that $\tilde{M}$ and $\tilde{N}$ both agree with the same substitution σ, then $\tilde{M} \Leftrightarrow \tilde{N}$.*

PROOF:

1. Follows from the fact that $\tilde{M}$ is maximally consistent.
2. By 1 above, both $\tilde{M}\tilde{N} \Leftrightarrow \tilde{M}$ and $\tilde{N}\tilde{M} \Leftrightarrow \tilde{N}$. □

Lemma 5.21 $\vdash_{\mathcal{O}} \sum_{i \in I} \tilde{M}_i P =_D P$, *if each $\tilde{M}_i$ is D-complete on* $\mathrm{fn}(P)$ *and all equivalence relations of* $\mathrm{fn}(P)$ *which respect D are in $\{\tilde{M}_i : i \in I\}$.*
PROOF: Use **M1′** to rearrange match sequences and to remove mismatch operators from terms $[x \neq y]P'$ where $(x, y) \in D$, **M5** to remove match/mismatch pairs, and finally **S** to rearrange sums. □

Definition 49 *An agent P is in D-*head normal form (D-HNF) on *V (a finite set of names) if P is on the form*

$$\sum_{i \in I} \tilde{M}_i(\tilde{x}_i)\alpha_i \,.\, P_i$$

where for all i, $\tilde{x}_i \cap V = \emptyset$, $\tilde{x}_i \subseteq \mathrm{obj}(\alpha_i)$ and $\tilde{M}_i$ is D-complete on V.

Lemma 5.22 *For all agents P and distinctions D and finite V such that* $\mathrm{fn}(P) \subseteq V$*, there is an agent H such that* $\mathrm{d}(H) \leq \mathrm{d}(P)$*, H is in D-HNF on V, and $\vdash_{\mathcal{O}} P =_D H$ from the axioms of Table 5.6.*
PROOF: By induction on the structure of P, almost verbatim the proof of Lemma 5.3, using **M1′** and **E′** in place of **M1** and **E**, and using **R5** for scope congruence.

Theorem 5.23 *$P \simeq^o_D Q$ iff $\vdash_{\mathcal{O}} P =_D Q$ from the axioms in Table 5.6.*

PROOF: That $\vdash_{\mathcal{O}} P =_D Q$ implies $P \simeq^o_D Q$ is verified by exhibiting the appropriate bisimulations. We present the case for **M1**$'$. Show $\tilde{M}P \simeq^o_D \tilde{N}Q$ if $\tilde{M} \wedge D \Leftrightarrow \tilde{N} \wedge D$ and $P \simeq^o_{D'} Q$ where $D' = D \cup \tilde{M}^-$. If $\tilde{M}P \xmapsto{\tilde{M}',\gamma} P'$ then $P \xmapsto{\tilde{M}_0,\gamma} P'$ where $\tilde{M}' = \tilde{M}\tilde{M}_0$. Since $P \simeq^o_{D'} Q$, there is $T \subseteq \{Q \xmapsto{\tilde{N}_i,\delta_i} Q'_i : \gamma\sigma_i = \delta_i\sigma_i, \sigma_i \models D'\tilde{M}_0\tilde{N}_i\}$ such that $D'\tilde{M}_0 \Rightarrow \bigvee \tilde{N}_i$ and for all i, $P' \simeq^o_{L_i} Q'_i$ where $L_i = D'\tilde{M}_0\tilde{N}_i D''$ and $D'' = \bigwedge\{[x \neq y] : (x, y) \in (\mathrm{bn}(\gamma) \times \mathrm{fn}(P, Q))\}$ if γ is a bound output, and $D' = \emptyset$ otherwise. Adding $\tilde{M}$ and $\tilde{N}$, the only concern is to show that $D \cup \tilde{M}' \Rightarrow \bigvee(\tilde{N}_i\tilde{N})$, which is clear from the fact that $D \cup \tilde{M}' = D \cup (\tilde{M}\tilde{M}_0) = (D \cup \tilde{M}^-)\tilde{M}_0\tilde{M} = D'\tilde{M}_0\tilde{M} \Rightarrow \bigvee \tilde{N}_i$, and we know $\tilde{M} \wedge D \Leftrightarrow \tilde{N} \wedge D$ which implies $\tilde{M} \wedge D \Rightarrow \tilde{N}$, so indeed $D \cup \tilde{M}' \Rightarrow \bigvee(\tilde{N}_i\tilde{N})$.

The other implication is more interesting. We show that $P \simeq^o_C Q$ implies $\vdash_{\mathcal{O}} P\sigma =_D Q\sigma$, where D corresponds to C^- and $\sigma \models C$. By Lemma 5.22 we can assume P and Q to be in D-HNF on $\mathrm{fn}(P, Q)$, and proceed by induction on the depth of $P + Q$, showing that each summand of P is provably equal to a summand of Q.

If the depth is 0, $P \equiv Q \equiv \mathbf{0}$ and we are done. Otherwise, let $\tilde{M}(\tilde{x})\alpha \,.\, P'$ be a summand of P. Then $P \xmapsto{\tilde{M},((\tilde{x})\alpha)} P'$. By alpha-conversion $\tilde{x} \cap \mathrm{fn}(Q) = \emptyset$, and by $P \simeq^o_C Q$, there is a $T \subseteq \{Q \xmapsto{\tilde{N}_i,((\tilde{x})\beta_i)} Q'_i : \alpha\sigma_i = \beta_i\sigma_i, \sigma_i \models C\tilde{M}\tilde{N}_i\}$, and $C\tilde{M} \Rightarrow \bigvee \tilde{N}_i$ and for all i, $P' \simeq^o_{C'_i} Q'_i$ where $C'_i = C\tilde{M}\tilde{N}_i$.

But since P and Q are in D-HNF, T is a singleton. We write the summand of Q with the simulating transition $\tilde{N}(\tilde{x})\beta \,.\, Q'$, where $\tilde{M} \Leftrightarrow \tilde{N}$ (since $\tilde{M}$ and $\tilde{N}$ are D-complete), $\alpha\sigma' = \beta\sigma'$ for $\sigma' \models C'$ where $C' = C\tilde{M}\tilde{N} = C\tilde{M}$, and $P' \simeq^o_{C'} Q'$. $\sigma \models C$ and $\tilde{M}$ is consistent with C, so $\sigma_{\tilde{M}}\sigma \models C'$, where $\sigma_{\tilde{M}}$ agrees with $\tilde{M}$.

$$\begin{aligned}
(\tilde{M}(\tilde{x})\alpha \,.\, P')\sigma &\overset{\mathbf{M2}}{=} (\tilde{M}(\tilde{x})(\alpha\sigma_{\tilde{M}}) \,.\, (P'\sigma_{\tilde{M}}))\sigma \equiv (\tilde{M}\sigma)(\tilde{x})(\alpha\sigma_{\tilde{M}}\sigma) \,.\, (P'\sigma_{\tilde{M}}\sigma) \\
&= (\tilde{M}\sigma)(\tilde{x})(\beta\sigma_{\tilde{M}}\sigma) \,.\, (Q'\sigma_{\tilde{M}}\sigma) \equiv (\tilde{M}(\tilde{x})(\beta\sigma_{\tilde{M}}) \,.\, (Q'\sigma_{\tilde{M}}))\sigma \\
&\overset{\mathbf{M2}}{=} (\tilde{M}(\tilde{x})\beta \,.\, Q')\sigma \overset{\mathbf{M1}'}{=} (\tilde{N}(\tilde{x})\beta \,.\, Q')\sigma
\end{aligned}$$

Thus we have proven that each summand of P is provably equal to a summand of Q, and then it suffices to use the **S** rules to infer that P and Q are provably equal. □

5.4 Conclusion

There are several obvious avenues of further work. The axiomatizations can presumably be extended to cover the finite-control fragment of the fusion calculus, following ideas from Lin [Lin95b]. A variant for weak open equivalence in the π-calculus is also probably straightforward by extending the proofs in the previous section.

It is probably less straightforward to prove the axioms independent, i.e., that each axiom cannot be derived from the other. Indeed, formal independence proofs have attracted very little attention so far. For example, we are not aware of a formal proof that the three tau-laws of CCS are independent (though such a proof is probably not very hard). Of the laws presented in Section 5.2 we conjecture that **T3a**, with its sequences of matches and mismatches, cannot be derived from **Tm3a** (without these sequences) and the other laws. On the other hand it is not difficult to show that a version of **T3a** with only *mis*matches is sufficient.

Chapter 6

Expressiveness

In this chapter we demonstrate the expressiveness of the fusion calculus by encoding other foundational calculi. First we encode the strong lazy λ-calculus, which admits reduction under abstraction. We then turn to constraint calculi: three basic variants of the ρ-calculus are encoded in the fusion calculus, and as a comparison we encode the γ-calculus in the π-calculus.

In these encodings we add the standard replication operator of the π-calculus (first defined in [Mil92]) to our calculus. Essentially a replication $!\,P$ can be seen as "any number of copies of P in parallel". The replication operator is definable through recursion by $!P \stackrel{def}{=} P \mid !P$.

6.1 Encoding the λ-Calculus

Most process calculi with the ability of treating mobile processes have attempted to embed or encode the λ-calculus, e.g. [Bou89, Tho90, Mil92, San93, San94, San95a, Bou97].

The lazy λ-calculus [Abr89] has been modelled and analysed in the π-calculus first by Milner [Mil92] and Sangiorgi [San93, San95a]. It has the following reduction rules, where M and N range over λ-calculus terms:

$$(\lambda x\,.\,M)N \to_\lambda M\{N/x\} \qquad \frac{M \to_\lambda M'}{MN \to_\lambda M'N}$$

Below we present Milner's encoding [Mil92] into the π-calculus:

$$\begin{array}{rcl} [\![x]\!]u & = & \overline{x}\,u \\ [\![\lambda x\,.\,M]\!]u & = & u(xv)\,.\,[\![M]\!]v \\ [\![MN]\!]u & = & (v)([\![M]\!]v \mid (x)\overline{v}\,xu\,.\,[\![x := N]\!]) \end{array}$$

where $[\![x := N]\!]$ stands for $!\,(w)xw\,.\,[\![N]\!]w$, which acts as an environment binding x to N.

The *strong* lazy reduction strategy, where reduction is possible under abstraction, is more difficult to encode. It has the following additional reduction rule:

$$\frac{M \rightarrow_\lambda M'}{\lambda x \,.\, M \rightarrow_\lambda \lambda x \,.\, M'}$$

The problem is that input binding in π-calculus can only be done by temporal guarding, $u(x) \,.\, P$, while the λ-calculus reduction rule allows the body of an abstraction to reduce *before* the abstracted variable is instantiated. In the fusion calculus, by decoupling the input action from the binding of the object, we can use the delayed input construct $(x)(u\,x \mid P)$ with the desired property (see Section 3.2).

To encode the lazy reduction strategy in the fusion calculus we only need to change Milner's encoding of the (non-strong) lazy reduction strategy minutely in the case of an abstraction, replacing the input prefix by a delayed input:

$$[\![\lambda x \,.\, M]\!]u \quad = \quad (x\,v)(u\,xv \mid [\![M]\!]v)$$

We now show that the resulting encoding is adequate, in the sense that there is a strong operational correspondence between reduction in the λ-calculus term and its encoding into the fusion calculus, and that convergence and divergence properties are preserved by the encoding. Below, we write $\tilde{N}$ for $N_1, \ldots, N_k$ and $[\![\tilde{x} := \tilde{N}]\!]$ for $[\![x_1 := N_1]\!] \mid \cdots \mid [\![x_n := N_n]\!], n \geq 0$. Let $\mathcal{L}$ stand for the family of λ-calculus terms, and $\mathcal{F}$ for the family of fusion calculus agents.

We define a correspondence between a λ term and its encoding by handling substitutions explicitly: a substitution $\{N/x\}$ on a λ term is represented by a fusion agent $[\![x := N]\!]$.

Definition 50 *The relation $\mathcal{S} \subseteq \mathcal{L} \times \mathcal{F}$ contains all pairs (L, P) such that for some $k \geq 0$, $M, N_1, \ldots, N_k \in \mathcal{L}$, the following holds:*

1. $\mathrm{fv}(N_i) \cap \{x_1, \ldots, x_i\} = \emptyset$ *for* $1 \leq i \leq k$,
2. $L \equiv M\{\tilde{N}/\tilde{x}\}$, *and*
3. $P \equiv (\tilde{x})([\![M]\!]u \mid [\![\tilde{x} := \tilde{N}]\!])$ *for any fresh* u.

So especially, for $k = 0$, $(M, [\![M]\!]u) \in \mathcal{S}$.

Theorem 6.1 *For any $(L, P) \in \mathcal{S}$,*

1. *if $L \rightarrow_\lambda L'$, then $P \rightarrow^+ P'$ such that $(L', P') \in \mathcal{S}$.*
2. *if $P \rightarrow P'$, then either $(L, P') \in \mathcal{S}$ or $L \rightarrow_\lambda L'$ such that $(L', P') \in \mathcal{S}$.*

PROOF: The proof goes by induction on the depth of the deduction of the reduction of L and transition of P, respectively, and induction on $k - i$.

1. By induction on the depth of the deduction of the reduction of L.

   Base case: $M \equiv (\lambda x_0.M_0)M_2$.
   Then $L \equiv M\{\tilde{N}/\tilde{x}\} \to_\lambda M_0\{M_2/x_0\}\{\tilde{N}/\tilde{x}\} \equiv L'$, and
   $P \equiv (\tilde{x})((v)((x_0\, w)(v\, zw \mid [\![M_0]\!]w) \mid (z)\overline{v}\, zu\,.\,[\![z := M_2]\!]) \mid [\![\tilde{x} := \tilde{N}]\!]) \to$
   $(x_0)(\tilde{x})([\![M_0]\!]u \mid [\![x_0 := M_2]\!] \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$, and $(L', P') \in \mathcal{S}$.

   Induction case:

   (a) $M \equiv \lambda x_0.M_0$. Then $L \equiv \lambda x_0.M_0\{\tilde{N}/\tilde{x}\} \to_\lambda \lambda x_0.M_0'\{\tilde{N}/\tilde{x}\}$ because $L_0 \equiv M_0\{\tilde{N}/\tilde{x}\} \to_\lambda M_0'\{\tilde{N}/\tilde{x}\} \equiv L_0'$.
   $P \equiv (\tilde{x})((x_0\, v)(u\, x_0 v \mid [\![M_0]\!]v) \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv$
   $(x_0\, v)(u\, x_0 v \mid (\tilde{x})([\![M_0]\!]v \mid [\![\tilde{x} := \tilde{N}]\!]))$. Define
   $P_0 \equiv (\tilde{x})([\![M_0]\!]v \mid [\![\tilde{x} := \tilde{N}]\!])$. Since the pair (L_0, P_0) is in $\mathcal{S}$ and $L_0 \to_\lambda L_0'$, by induction $P_0 \to^+ P_0'$ s.t. $(L_0', P_0') \in \mathcal{S}$ and
   $P_0' \equiv (\tilde{x})([\![M_0']\!]v \mid [\![\tilde{x} := \tilde{N}]\!])$. Further, $P_0 \to^+ P_0'$ implies that
   $P \to^+ (x_0\, v)(u\, x_0 v \mid (\tilde{x})([\![M_0']\!]v \mid [\![\tilde{x} := \tilde{N}]\!])) \equiv (\tilde{x})([\![\lambda x_0.M_0']\!]u \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$, and $(L', P') \in \mathcal{S}$ as desired.

   (b) $M \equiv M_1 M_2$ and $M_1\{\tilde{N}/\tilde{x}\} \to_\lambda M_1'\{\tilde{N}/\tilde{x}\}$. Then $L \to_\lambda (M_1' M_2)\{\tilde{N}/\tilde{x}\} \equiv L'$.
   $P \equiv (\tilde{x})((v)([\![M_1]\!]v \mid (z)\overline{v}\, zu\,.\,[\![z := M_2]\!]) \mid [\![\tilde{x} := \tilde{N}]\!])$. Since $M_1\{\tilde{N}/\tilde{x}\} \to_\lambda M_1'\{\tilde{N}/\tilde{x}\}$, by induction we know that
   $(\tilde{x})([\![M_1]\!]v \mid [\![\tilde{x} := \tilde{N}]\!]) \to^+ (\tilde{x})([\![M_1']\!]v \mid [\![\tilde{x} := \tilde{N}]\!])$ which is related in $\mathcal{S}$ to $M_1'\{\tilde{N}/\tilde{x}\}$. By the PAR and PASS rules, this means that
   $P \to^+ (\tilde{x})((v)([\![M_1']\!]v \mid (z)\overline{v}\, zu\,.\,[\![z := M_2]\!]) \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv$
   $(\tilde{x})([\![M_1' M_2]\!] \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$, and again $(L', P') \in \mathcal{S}$.

   (c) $M \equiv x_i$ where $x_i \in \tilde{x}$, and
   $L \equiv x_i\{\tilde{N}/\tilde{x}\} \equiv N_i\{\tilde{N}/\tilde{x}\} \to_\lambda N_i'\{\tilde{N}/\tilde{x}\} \equiv L'$.
   $P \equiv (\tilde{x})(\overline{x_i}\, u \mid [\![\tilde{x} := \tilde{N}]\!]) \to (\tilde{x})([\![N_i]\!]u \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P''$. Since $(L, P'') \in \mathcal{S}$ and $\mathrm{fv}(N_i) \cap \{x_1, \ldots, x_i\} = \emptyset$, then by induction on $k - i$ or by previous cases, the lemma holds.

2. By induction on the depth of the deduction of the reduction of $P \equiv (\tilde{x})([\![M]\!]u \mid [\![\tilde{x} := \tilde{N}]\!])$.

   Base case: $M \equiv x_i$ and $x_i \in \tilde{x}$. Then $P \to (\tilde{x})([\![N_i]\!]u \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$, and $L \equiv x_i\{\tilde{N}/\tilde{x}\} \equiv N_i\{\tilde{N}/\tilde{x}\}$, and thus $(L, P') \in \mathcal{S}$.

   Induction case:

   (a) $M \equiv \lambda x_0.M_0$. Then $P \equiv (\tilde{x})((x_0\, w)(u\, x_0 w \mid [\![M_0]\!]w) \mid [\![\tilde{x} := \tilde{N}]\!])$
   $\equiv (x_0\, w)(u\, x_0 w \mid (\tilde{x})([\![M_0]\!] \mid [\![\tilde{x} := \tilde{N}]\!]))$, so by induction
   $P \to (x_0\, w)(u\, x_0 w \mid (\tilde{x})([\![M_0']\!]w \mid [\![\tilde{x} := \tilde{N}]\!])) \equiv P'$, s.t.
   $M_0\{\tilde{N}/\tilde{x}\} \to_\lambda M_0'\{\tilde{N}/\tilde{x}\}$ or $M_0'\{\tilde{N}/\tilde{x}\} \equiv M_0\{\tilde{N}/\tilde{x}\}$. Now,
   $P' \equiv (\tilde{x})([\![\lambda x_0.M_0']\!]u \mid [\![\tilde{x} := \tilde{N}]\!])$, and either
   $L \to_\lambda (\lambda x_0.M_0')\{\tilde{N}/\tilde{x}\} \equiv L'$ s.t. $(L', P') \in \mathcal{S}$, or $(L, P') \in \mathcal{S}$.

(b) $M \equiv M_1 M_2$. There are two cases; either M_1 is an abstraction $\lambda x_0.M_0$ and
$P \equiv (\tilde{x})((v)((x_0\, w)(v\, x_0 w \mid [\![M_0]\!]w) \mid (z)\overline{v}\, zu\,.\, [\![z := M_2]\!]) \mid [\![\tilde{x} := \tilde{N}]\!])$
reduces by communication over v, or P reduces in some other way.

i. $P \rightarrow (x_0)(\tilde{x})([\![M_0]\!]u \mid [\![x_0 := M_2]\!] \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$, and
$L \rightarrow_\lambda M_0\{M_2/x_0\}\{\tilde{N}/\tilde{x}\} \equiv L'$, and $(L', P') \in \mathcal{S}$.

ii. $P \equiv (\tilde{x})((v)([\![M_1]\!]v \mid (z)\overline{v}\, zu\,.\, [\![z := M_2]\!]) \mid [\![\tilde{x} := \tilde{N}]\!]) \rightarrow$
$(\tilde{x})((v)([\![M_1']\!]v \mid (z)\overline{v}\, zu\,.\, [\![z := M_2]\!]) \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv$
$(\tilde{x})([\![M_1' M_2]\!]u \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$ like previously.
$L \rightarrow_\lambda (M_1' M_2)\{\tilde{N}/\tilde{x}\} \equiv L'$ and again, $(L', P') \in \mathcal{S}$. □

Lemma 6.2 *For $(L, P) \in \mathcal{S}$, if $L \not\rightarrow_\lambda$, then $P \rightarrow^* P'$ such that $P' \not\rightarrow$ and $(L, P') \in \mathcal{S}$.*

Proof: By structural induction on M, where $L \equiv M\{\tilde{N}/\tilde{x}\}$.
The base case where $M \equiv z$ and $z \notin \tilde{x}$ is trivial.
Induction:

1. $M \equiv \lambda x_0.M_0$ s.t. $M_0\{\tilde{N}/\tilde{x}\} \not\rightarrow_\lambda$. Then
$P \equiv (z\, w)(u\, zw \mid (\tilde{x})([\![M_0]\!]w \mid [\![\tilde{x} := \tilde{N}]\!])$ and since
$P_0 \equiv (\tilde{x})([\![M_0]\!]w \mid [\![\tilde{x} := \tilde{N}]\!])$ is related in $\mathcal{S}$ to $L_0 \equiv M_0\{\tilde{N}/\tilde{x}\}$, and by induction $P_0 \rightarrow^* P_0'$ s.t. $P_0' \not\rightarrow$ and $(L_0, P_0') \in \mathcal{S}$. This implies that
$P \rightarrow^+ P'$ s.t. $P' \not\rightarrow$ and $(L', P') \in \mathcal{S}$.

2. $M \equiv M_1 M_2$. Holds by similar reasoning.

3. $M \equiv x_i$ where $x_i \in \tilde{x}$. Then $P \rightarrow (\tilde{x})([\![N_i]\!]u \mid [\![\tilde{x} := \tilde{N}]\!]) \equiv P'$, and since $(L, P') \in \mathcal{S}$ and $\mathrm{fn}(N_i) \cap \{x_1, \ldots, x_i\} = \emptyset$, by induction on $k - i$ or previous cases the lemma holds. □

Definition 51 *We say that L* converges to *L', written $L \downarrow L'$, if $L \rightarrow_\lambda^* L' \not\rightarrow_\lambda$, and that P converges to P', written $P \downarrow P'$, if $P \rightarrow^* P' \not\rightarrow$.*

The preservation of convergence (and thereby divergence) properties follows from the preceding Lemma and Theorem.

Theorem 6.3 *For $(L, P) \in \mathcal{S}$, if $L \downarrow L'$ then $P \downarrow P'$ with $(L', P') \in \mathcal{S}$, and if $P \downarrow P'$ then $L \downarrow L'$ with $(L', P') \in \mathcal{S}$.*

Proof: Follows by induction on the number of reductions/transitions before convergence, using of Theorem 6.1, and from Lemma 6.2. □

Using other calculi where variants of delayed input can be defined, strong lazy reduction in the λ-calculus has been considered by Fu [Fu97], by Sangiorgi [San96b], and by Merro and Sangiorgi [MS98]. All three have postulated solutions though not published any formal analysis of their encodings. Fu extends the π-calculus to something more complicated, where we instead simplify it. The latter solutions use comparatively complicated encodings in the πI-calculus and πA-calculus (a variant of π_a) respectively, where we instead can formulate a straightforward modification of Milner's original encoding and proofs for the (non-strong) lazy λ-calculus.

6.2 Encoding Constraint Calculi

In concurrent constraints, the effect of computation consists of changes to a shared *constraint store* by adding variable constraints to it by *tell* operations. The store combines the constraints, and can subsequently be queried by *ask* operations to find out if a formula containing constrained variables is entailed.

The ρ-calculus of Niehren and Müller [NM95] is a concurrent calculus which serves as a foundation for the concurrent constraint language Oz [Smo95b], developed at DFKI in Saarbrücken. Oz integrates concurrent constraint programming with higher-order functional programming and concurrent object-oriented programming, and requires a richer foundational calculus than a pure constraint calculus as in, e.g., [SRP91]. The process part of the ρ-calculus is a subcalculus of the π-calculus [NM95], but it is still expressive enough to model functional and object-oriented programming. Its development has been inspired by process calculi such as the π-calculus, but behavioural equivalences and proof methods have not been developed for it.

The ρ-calculus can be extended by an arbitrary constraint system $\mathcal{CS}$, resulting in the calculus $\rho(\mathcal{CS})$. This takes the calculus beyond traditional process calculi and it thus cannot directly use the rich flora of associated theories and tools. An example of a term in $\rho(x = y, x \neq y)$, with constraints being equalities or inequalities, is

$$x = u \;\wedge\; u \neq y \;\wedge\; \texttt{if}\ x = y\ \texttt{then}\ E_1\ \texttt{else}\ (z = w \wedge E_2)$$

which can evolve into $x = u \;\wedge\; u \neq y \;\wedge\; z = w \;\wedge\; E_2$. The top-level equality and inequality conjuncts in this way represent the store and the `if`-conjunct corresponds to *ask*. In the ρ-calculus, the constraint store can be seen as transparently distributed over the terms, in contrast with the monolithic constraint store of, e.g., [SRP91]. The constraint store can be asked if a formula or its negation is entailed by using a conditional term, and a constraint is implicitly added when it is not guarded by a conditional or abstraction.

In this section we relate the fusion calculus to three basic instances of the ρ-calculus: $\rho(x = y)$, which uses constraints over name equations and conjunction; $\rho(x = y, \mathcal{C})$, which adds constants to the constraint system; and $\rho(x = y, x \neq y)$, which instead adds inequalities between names.

We relate the calculi by giving a simple encoding of a term E in one of the instances of the ρ-calculus such that if E can reduce to another term F, then its encoding can reduce to the encoding of F, and vice versa. Correspondence between observations and of convergence properties is also shown.

6.2.1 The ρ-Calculus

We give a brief presentation of the syntax and semantics of the ρ-calculus. For a fuller treatment we refer the reader to [NM95].

$E \wedge \dot{\top} \equiv E$	$E \wedge \top \equiv E$	$E \wedge F \equiv F \wedge E$	
$E \wedge (F \wedge G) \equiv (E \wedge F) \wedge G$	$\exists x\, \dot{\top} \equiv \dot{\top}$	$\exists x \top \equiv \top$	$\exists x \exists y E \equiv \exists y \exists x E$
$E \wedge \exists x F \equiv \exists x (E \wedge F)$ if $x \notin \mathrm{fv}(E)$		$\phi \wedge \psi \equiv \phi \mathbin{\dot{\wedge}} \psi$	$\phi \equiv \varphi$ if $\phi \Leftrightarrow \varphi$

Table 6.1: Structural congruence relation of the ρ-calculus.

$\phi \wedge x{:}\tilde{y}/E \wedge z\tilde{w} \rightarrow \phi \wedge x{:}\tilde{y}/E \wedge E\{\tilde{w}/\tilde{y}\}$	if $\phi \models x = z$ and $\lvert\tilde{y}\rvert = \lvert\tilde{w}\rvert$
$\phi \wedge \texttt{if } \psi \texttt{ then } E \texttt{ else } F \rightarrow \phi \wedge E$	if $\phi \models \psi$
$\phi \wedge \texttt{if } \psi \texttt{ then } E \texttt{ else } F \rightarrow \phi \wedge F$	if $\phi \models \neg\psi$

$$\frac{E \rightarrow F}{\exists x E \rightarrow \exists x F} \qquad \frac{E \rightarrow E'}{E \wedge F \rightarrow E' \wedge F} \qquad \frac{E \equiv E' \quad E' \rightarrow F' \quad F' \equiv F}{E \rightarrow F}$$

Table 6.2: Reduction rules of the ρ-calculus.

There are three basic entities in the ρ-calculus: *variables*, *constraints* and *terms*. Variables are ranged over by lower-case letters, constraints by $\phi, \varphi, \psi, \ldots$, and terms by $E, F, \ldots$. Their syntax is given by the following BNF equation:

$$E \quad ::= \quad \top \;\Big|\; E_1 \wedge E_2 \;\Big|\; \exists x E \;\Big|\; x\tilde{y} \;\Big|\; x{:}\tilde{y}/E \;\Big|\; \texttt{if } \varphi \texttt{ then } E_1 \texttt{ else } E_2 \;\Big|\; \phi$$

The terminology in the following explanation comes from [NM95]: Briefly, $\top$ represents an inactive term; each term in a *composition* $E_1 \wedge E_2$ can reduce in parallel; a *declaration* $\exists x E$ introduces a new variable x with scope E; an *abstraction* $x{:}\tilde{y}/E$ can communicate with an *application* $x\tilde{z}$, producing the new term $E\{\tilde{z}/\tilde{y}\}$. A *conditional* $\texttt{if } \varphi \texttt{ then } E_1 \texttt{ else } E_2$ tests if the current constraint store entails the condition φ or its negation. A *constraint* term ϕ is added to the current constraint store. We write $\phi \models \psi$ if $\phi \Rightarrow \psi$ is valid in all models of the constraint system. The constraint $\dot{\top}$ stands for logical truth, and conjunction of constraints is written $\dot{\wedge}$. The terms $\exists y E$ and $x{:}\tilde{y}/E$ bind y and $\tilde{y}$, respectively; we write $\mathrm{fv}(E)$ for all variables in E which are not bound.

The formal semantics of the ρ-calculus is given by a structural congruence in Table 6.1 and reduction rules in Table 6.2.

We say that a constraint is *guarded* if it occurs in the body of an abstraction or the clauses of a conditional. We show that a ρ-calculus term E can be written in a head normal form where all unguarded constraints are merged. For example, $x = y \wedge \exists z(z = w \wedge aw)$ has the head normal form $\exists z(x = y \mathbin{\dot{\wedge}} z = w \wedge aw)$.

Definition 52 *A ρ-calculus term E is in* head normal form *if E is of the form $\exists\tilde{x}(\phi \wedge E')$, where E' has no unguarded constraints and $\tilde{x} \subseteq \mathrm{fv}(\phi)$.*

In this head normal form ϕ can be thought of as representing the store.

Proposition 6.4 *Any ρ-calculus term E is structurally congruent to a term F which is in head normal form.*
PROOF: By simple structural induction on terms.

We further define the observation predicate of the ρ-calculus in a straightforward way:

Definition 53
$$\begin{array}{ll} x\tilde{y} \downarrow x & \\ x{:}\,\tilde{y}/E \downarrow x & \\ (E \mid F) \downarrow x & \textit{if } E \downarrow x \textit{ or } F \downarrow x \\ \exists x E \downarrow y & \textit{if } E \downarrow y \textit{ and } x \neq y \end{array}$$

6.2.2 Constraints Over Name Equations

We first instantiate the constraint system in ρ with simple name equations and conjunction. This calculus is $\rho(x = y)$ of [NM95]. We give a simple encoding for it into the fusion calculus in Table 6.3. For simplicity, we only describe the encoding where the condition of a conditional is of the form $x = y$. The general case can be handled by nesting conditionals. The encoding of a ρ-calculus name x is written $\underline{x}$. In Table 6.3 it suffices to define $\underline{x} = x$, assuming a unique name in the fusion calculus for each name in the ρ-calculus. Note that in $\rho(x = y)$, it is *never* the case that $\phi \models \neg(x = y)$ for any ϕ, so the "else" branch of a conditional can be completely ignored.

In [NM95] a *cell* construct is given as primitive. The cell $x{:}\,y$ can be seen as located at x and holding the value y, and can interact with an application $x\,vw$, updating the cell's value to w and simultaneously extracting the old value rendering the equation $y = v$. Formally cells have the reduction rule

$$\phi \wedge x{:}\,y \wedge x'\,uv \rightarrow \phi \wedge x{:}\,u \wedge v = y \quad \text{if } \phi \models x = x'$$

and can easily be encoded by $\mathit{Cell}(x,u) \stackrel{\mathit{def}}{=} (w)x\,uw\,.\,\mathit{Cell}(x,w)$, nicely utilizing the free input of the fusion calculus.

For the correctness of the encoding, we treat top-level constraints specially, and define $\mathcal{V}(\exists\tilde{u}(\phi\wedge E)) \stackrel{\mathit{def}}{=} (\underline{\tilde{u}})[\![E]\!]\sigma_\phi$ where $\exists\tilde{u}(\phi\wedge E)$ is in head normal form, σ_ϕ agrees with $[\![\phi]\!]$ and $\mathrm{ran}(\sigma_\phi) \cap \underline{\tilde{u}} = \emptyset$. In other words, the effect of the equations of the store is applied onto E. For example, $\mathcal{V}(\exists x(x = y \wedge axy))$ is $\overline{a}\,yy$.

The correctness is stated by the following theorem. The addition of scopes $\tilde{z}$ is a technical device, necessary to turn guarded fusions of free names in $\mathcal{V}(E)$ into reductions.

Theorem 6.5 *For E, F terms of $\rho(x = y)$ in head normal form,*

1. *If $E \rightarrow F$, then for some $\tilde{z}$, $(\tilde{z})\mathcal{V}(E) \rightarrow^+ P \approx_b \mathcal{V}(F)$.*
2. *For any $\tilde{z}$, if $(\tilde{z})\mathcal{V}(E) \rightarrow^* P$, then $E \rightarrow^* F$ and $P \approx_b (\tilde{z})\mathcal{V}(F)$.*
3. *$E \downarrow x$ iff $\mathcal{V}(E) \downarrow x'$, where $E = \phi \wedge E'$ and $\phi \models x = x'$.*

$$
\begin{array}{rcl}
\underline{x} & \equiv & x \\
[\![\top]\!] & \stackrel{def}{=} & \mathbf{0} \\
[\![x\tilde{w}]\!] & \stackrel{def}{=} & \overline{x}\,\underline{\tilde{w}} \\
[\![x\colon\tilde{y}/E]\!] & \stackrel{def}{=} & !\,(\underline{\tilde{y}})x\,\underline{\tilde{y}}\,.\,[\![E]\!] \\
[\![x_1 = y_1 \mathbin{\dot\wedge} \cdots \mathbin{\dot\wedge} x_n = y_n]\!] & \stackrel{def}{=} & \{\underline{x_1} = \underline{y_1}, \cdots, \underline{x_n} = \underline{y_n}\} \\
[\![E \wedge F]\!] & \stackrel{def}{=} & [\![E]\!] \mid [\![F]\!] \\
[\![\exists x E]\!] & \stackrel{def}{=} & (\underline{x})[\![E]\!] \\
[\![\mathtt{if}\ x = y\ \mathtt{then}\ E\ \mathtt{else}\ F]\!] & \stackrel{def}{=} & [x = y]\mathbf{1}\,.\,[\![E]\!]
\end{array}
$$

Table 6.3: The encoding of $\rho(x = y)$.

4. *If E converges to F, then for some $\tilde{z}$, $(\tilde{z})\mathcal{V}(E)$ converges to $P \approx_b \mathcal{V}(F)$.*

5. *For any $\tilde{z}$, if $(\tilde{z})\mathcal{V}(E)$ converges to P and P has no fusion-labelled transition, then E converges to F with $P \approx_b (\tilde{z})\mathcal{V}(F)$.*

PROOF: By inductions on the depths of inferences.

1. (a) $E \equiv \phi \wedge x\colon\tilde{y}/E' \wedge z\tilde{w} \to \phi \wedge x\colon\tilde{y}/E' \wedge E'\{\tilde{w}/\tilde{y}\} \equiv F$ if $\phi \models x = z$. Then $\mathcal{V}(E) = (!\,(\underline{\tilde{y}})x\tilde{y}\,.\,[\![E']\!] \mid \overline{z}\,\underline{\tilde{w}})\sigma_\phi \to (!\,(\underline{\tilde{y}})x\tilde{y}\,.\,[\![E']\!] \mid [\![E']\!]\{\underline{\tilde{w}}/\underline{\tilde{y}}\})\sigma_\phi \equiv \mathcal{V}(F)$, where σ_ϕ agrees with ϕ and thus $\sigma_\phi(x) = \sigma_\phi(z)$.

   (b) $E \equiv \phi \wedge \mathtt{if}\ \varphi\ \mathtt{then}\ E_1\ \mathtt{else}\ E_2 \to \phi \wedge E_1 \equiv F$, if $\phi \models \varphi$. Then $\mathcal{V}(E) \equiv ([\varphi]\mathbf{1}\,.\,[\![E_1]\!])\sigma_\phi \to [\![E_1]\!]\sigma_\phi$.

   (c) if $E \to F$ otherwise, then by induction and corresponding reduction rules for the fusion calculus, $\mathcal{V}(E) \to^+\approx_b \mathcal{V}(F)$.

2. As case 1 in opposite order, plus the case $(\tilde{z})\mathcal{V}(\phi \wedge E) \equiv (\tilde{z})([\![\phi]\!] \mid [\![E]\!]) \to [\![E]\!]\sigma_\phi \equiv \mathcal{V}(\phi \wedge E)$, where σ_ϕ agrees with ϕ, $\mathrm{ran}(\sigma_\phi) \cap \tilde{z} = \emptyset$, and $\mathrm{dom}(\sigma_\phi) = \tilde{z}$, so $\tilde{z} \cap \mathrm{fn}([\![E]\!]\sigma_\phi) = \emptyset$.

3. Since $\mathcal{V}(\phi \wedge E') \equiv [\![E']\!]\sigma_\phi$, it follows by simple induction on the depth of inference of $E' \downarrow x$ and $[\![E']\!] \downarrow x'$.

4. By induction on the number of reductions made before converging. If $E \not\to$, then if E is an abstraction or application, $\mathcal{V}(E) \not\to$. If $E \equiv \phi$, then for a suitable $\tilde{z}$, $(\tilde{z})\mathcal{V}(E) \equiv (\tilde{z})[\![\phi]\!]\,.\,\mathbf{0} \to \mathbf{0}\sigma_\phi \not\to$. If $E \equiv \phi \wedge \mathtt{if}\ \varphi\ \mathtt{then}\ E_1\ \mathtt{else}\ E_2$ and $\phi \not\models \varphi$ and $\phi \not\models \neg\varphi$, then either $[\![\varphi]\!]\sigma_\psi$ is not true or $\varphi \equiv x = y$ for some x, y s.t. $\sigma_\psi(x) \neq \sigma_\psi(y)$, and in both cases $\mathcal{V}(E) \not\to$.

   The induction case follows from case 1.

$$
\begin{array}{rcll}
\underline{u} & \equiv & u, u_c & \\
C(a_c) & \stackrel{def}{=} & !\,\overline{a_c} & \\
[\![\exists a E]\!] & \stackrel{def}{=} & (\underline{a})(C(a_c) \mid [\![E]\!]) & \text{if } a \in \mathcal{C} \\
[\![\exists x E]\!] & \stackrel{def}{=} & (\underline{x})[\![E]\!] & \text{if } x \notin \mathcal{C} \\
\multicolumn{4}{l}{[\![\texttt{if } x = y \texttt{ then } E \texttt{ else } F]\!]} \\
& \stackrel{def}{=} & \multicolumn{2}{l}{[x = y](\mathbf{1}\,.\,[\![E]\!]\,, x_c\,.\,y_c\,.\,[x = y](\mathbf{1}\,.\,[\![E]\!]\,, \mathbf{1}\,.\,[\![F]\!]))}
\end{array}
$$

Table 6.4: The encoding of $\rho(x = y, \mathcal{C})$.

5. By induction on the number of reductions made before converging. The base case is just like in case 4 in the opposite order, and the induction follows from case 2. Note that the side condition that P has no fusion-labelled transition is necessary since, e.g., $\mathcal{V}(\texttt{if } x = x \texttt{ then } y = z \texttt{ else } E') \rightarrow \{y = z\} \not\rightarrow$, but $\texttt{if } x = x \texttt{ then } y = z \texttt{ else } E' \rightarrow y = z \not\rightarrow$, and $\mathcal{V}(y = z) = \mathbf{0}\sigma_{\{y=z\}} \not\approx_b \{y = z\}$. Under a scope such as (y) the equality holds. $\square$

6.2.3 Adding Constants

Adding constants to the constraint system of $\rho(x = y)$ gives a system which is closer to the γ-calculus of Smolka [Smo94]. Here, the variables are divided into two families: normal variables and constants, and the axiom $\forall x, y \in \mathcal{C}, x \not\equiv y : x \neq y$ is added to the constraint system, where $\mathcal{C}$ is the family of constants. This means that two constants which are not literally the same are always different. This instance of the ρ-calculus is called $\rho(x = y, \mathcal{C})$. Below we encode it into the fusion calculus. Here and in the following encoding, we write $[x = y](P\,, Q)$ for the fusion calculus agent $[x = y]P + [x \neq y]Q$.

As we have seen in the encoding of $\rho(x = y)$, variables of the ρ-calculus correspond directly to names in the fusion calculus. Constants, however, are explicitly encoded in the fusion calculus by adding a *handler process* for each constant, which can tell its environment that it is a constant. The encoding of a conditional can interrogate these handler processes to find out if $\neg\varphi$ is entailed. (Cf. Section 6.3, where the γ-calculus is encoded in the π-calculus using much more complex handler processes for constants, variables and updated variables.)

For each variable or constant u in $\rho(x = y, \mathcal{C})$ we introduce *two* names in the encoding: u and u_c. We redefine $\underline{u}$ to refer to these two together. u_c is used for communicating with a handler process for u, if it exists. A handler process for u can synchronize repeatedly on u_c, thereby signalling that u is a constant. We update the encoding in Table 6.3 by Table 6.4.

The encoding of a conditional deserves a short explanation: If x and y are the same, we choose the E branch directly. Otherwise, to check if $\neg(x = y)$ is entailed, we must verify that both x and y are constants. Then we must *again* test if $x = y$ or not, for although they were not equal, a concurrent constraint may since have fused them.

For the correctness of this encoding, we must put the encoded terms in a context where all free a_c are scoped, and all free constants have a handler process. When I is a finite set we use $\prod_{i \in I} P_i$ to mean the Composition of all P_i for $i \in I$, and define $\mathcal{V}(\exists \tilde{u}(\phi \wedge E)) \stackrel{def}{=} (\underline{\tilde{u}})(\tilde{v}_c)((\prod_{a \in \tilde{a}} C(a_c) \mid [\![E]\!])\sigma_\phi)$ where $\exists \tilde{u}(\phi \wedge E)$ is in head normal form, $\tilde{v} = \mathrm{fv}(\exists \tilde{u}(\phi \wedge E))$, $\tilde{a} = \tilde{v} \cap \mathcal{C}$, σ_ϕ agrees with $[\![\phi]\!]$ and $\mathrm{ran}(\sigma_\phi) \cap \underline{\tilde{u}} = \emptyset$.

Theorem 6.6 *Theorem 6.5 holds for $\rho(x = y, \mathcal{C})$ with this definition of $\mathcal{V}$.*
PROOF:

1. We only need to reconsider the case where

$$\begin{array}{rcl} E & \equiv & \phi \wedge \texttt{if } x = y \texttt{ then } E_1 \texttt{ else } E_2 \\ \mathcal{V}(E) & \equiv & (\underline{\tilde{a_c}})(\prod C(a_{ci}) \\ & & \quad \mid [x = y](\mathbf{1} \,.\, [\![E_1]\!] \,, x_c \,.\, y_c \,.\, [x = y](\mathbf{1} \,.\, [\![E_1]\!] \,, \mathbf{1} \,.\, [\![E_2]\!])))\sigma_\phi \end{array}$$

   The case where $x \equiv y$ is trivial. Otherwise, if $x \equiv a_i$ for some i, $\mathcal{V}(E) \rightarrow (\underline{\tilde{a_c}})(\prod C(a_{ci}) \mid y_c \,.\, [a_i = y](\mathbf{1} \,.\, [\![E_1]\!] \,, \mathbf{1} \,.\, [\![E_2]\!]))\sigma_\phi \equiv P'$. If $y \equiv a_j$ then $P' \rightarrow (\underline{\tilde{a_c}})(\prod C(a_{ci}) \mid [a_i = a_j](\mathbf{1} \,.\, [\![E_1]\!] \,, \mathbf{1} \,.\, [\![E_2]\!]))\sigma_\phi \equiv P''$, and now if $a_i = a_j$ then $P'' \rightarrow \mathcal{V}(E_1)$, or if $a_i \neq a_j$, $P'' \rightarrow \mathcal{V}(E_2)$.

2. As for Theorem 6.5, using the above case.

3. As for Theorem 6.5.

4. As for Theorem 6.5, and in addition the case where E is a conditional testing two different variables, and then it is apparent that $\mathcal{V}(E) \rightarrow^* \not\rightarrow$, since it will not get past the u_c or v_c prefix.

5. As for Theorem 6.5, using the above case. □

An inconsistent constraint store can be detected by redefining $C(a_c) \stackrel{def}{=} \overline{a_c} \,.\, C(a_c) + a_c \,.\, \mathit{fail}$. Then *fail* is a weak observation if the store becomes inconsistent.

6.2.4 Name Inequations

Instead of constants, we here add explicit inequations to the constraint system, acquiring $\rho(x = y, x \neq y)$. Although $x \neq y$ could be encoded in $\rho(x = y, \mathcal{C})$ as $\exists a \exists b(x = a \wedge y = b)$, in $\rho(x = y, x \neq y)$ we avoid splitting the variables into constants and normal variables. In this encoding we also need additional names and processes, but the processes now explicitly encode inequations rather than constants.

The encoding in Table 6.5 updates the one in Table 6.3; note again the redefinition of $\underline{x}$. The presence of a process $D(\underline{x}, \underline{y})$ means that the inequation

$\underline{x}$	$\equiv$	x, x_d
$D(\underline{x}, \underline{y})$	$\stackrel{def}{=}$	$!\,(w,n)x_d\,wn\,.\,[w=y](\overline{n}\,,\overline{x_d}\,wn)$
$[\![\texttt{if } x=y \texttt{ then } E \texttt{ else } F]\!]$		
	$\stackrel{def}{=}$	$(n)(\overline{x_d}\,yn \mid \overline{y_d}\,xn \mid ([x=y](\mathbf{1}\,.\,[\![E]\!]\,,n\,.\,[\![F]\!])))$
$[\![x_1 \neq y_1 \mathbin{\dot\wedge} \cdots \mathbin{\dot\wedge} x_n \neq y_n]\!]$		
	$\stackrel{def}{=}$	$D(\underline{x_1}, \underline{y_1}) \mid \cdots \mid D(\underline{x_n}, \underline{y_n})$

Table 6.5: The encoding of $\rho(x=y, x \neq y)$.

$x \neq y$ is present in the store. This process handles queries along x_d with objects w, n, meaning "is x different from w"? If $w = y$ it replies affirmatively by synchronizing on the supplied name n; otherwise it re-iterates the query to see if some other handler can decide it. This introduces divergence in the encoding, since the new query can be handled by the same process. We regard this as processing internal to the constraint system, which should not be a concern of the external observer. The weak barbed congruence respects this view and ignores divergence: $\mathbf{0} \approx_b \Omega$ where $\Omega \stackrel{def}{=} \mathbf{1}\,.\,\Omega$. The encoding of a conditional simply asks both names in the equation if they are different.

The correctness again uses a context at top level, which scopes all free x_d. We redefine $\mathcal{V}(\exists\tilde{u}(\phi \mathbin{\dot\wedge} \psi \wedge E)) \stackrel{def}{=} (\underline{\tilde{u}})(\tilde{v}_d)[\![\psi \wedge E]\!]\sigma_\phi$ where $\exists\tilde{u}(\phi \mathbin{\dot\wedge} \psi \wedge E)$ is in head normal form, ϕ contains only positive constraints and ψ only negative, $\tilde{v} = \mathrm{fv}(\exists\tilde{u}(\phi \mathbin{\dot\wedge} \psi \wedge E))$, σ_ϕ agrees with $[\![\phi]\!]$ and $\mathrm{ran}(\sigma_\phi) \cap \underline{\tilde{u}} = \emptyset$.

Theorem 6.7 *For $\rho(x=y, x\neq y)$ and this definition of $\mathcal{V}$,*

- *Theorem 6.5(1–3) hold.*
- *If we disregard divergences introduced by D agents, Theorem 6.5(4) holds.*
- *Whether or not we disregard divergences introduced by D agents, Theorem 6.5(5) holds.*

PROOF:

1. We only need to reconsider the case where

$$\begin{array}{rcl} E & \equiv & \phi \wedge \texttt{if } x=y \texttt{ then } E_1 \texttt{ else } E_2 \\ \mathcal{V}(E) & \equiv & (\tilde{z_d})(\prod D(\underline{z_i}, \underline{z_j}) \\ & & \qquad \mid (n)(\overline{x_d}\,yn \mid \overline{y_d}\,xn \mid [x=y](\mathbf{1}\,.\,[\![E_1]\!]\,,n\,.\,[\![E_2]\!])))\sigma_\phi \end{array}$$

If $x = y$, then $\mathcal{V}(E) \rightarrow (\tilde{z_d})(\prod D(\underline{z_i}, \underline{z_j}) \mid (n)(\overline{x_d}\,yn \mid \overline{y_d}\,xn) \mid [\![E_1]\!])\sigma_\phi \equiv P'$. The term $(n)(\overline{x_d}\,yn \mid \overline{y_d}\,xn)$ may introduce divergence in interaction with the D terms, but will have no other effect since no other processes can interact on n, and since our weak equivalence ignores divergence, $P' \approx_b \mathcal{V}(\phi \wedge E_1)$.

2. As for Theorem 6.5, using the above case.
3. As for Theorem 6.5.
4. As for Theorem 6.5, and in addition the case where E is a conditional testing two different variables, and then it is apparent that $\mathcal{V}(E)$ will diverge, but only as in case 1 above.
5. As for Theorem 6.5; in addition by the above case divergence can be ignored. $\square$

By using $D'(\underline{x}, \underline{y}) \stackrel{def}{=} [x = y]fail \mid D(\underline{x}, \underline{y})$ in place of D, we can detect inconsistencies of the store.

6.3 Encoding the γ-Calculus in the π-Calculus

The relative expressiveness of the fusion calculus and the π-calculus can be studied by comparing the encodings of the ρ-calculus with a π-calculus encoding of the γ-calculus, another foundational constraint calculus. Here the major obstacle is expressing logical variables, which can be updated from several processes and not only at their binding occurrence. The variables are modelled by separate processes which can be read, updated etc. by sending requests and receiving responses.

6.3.1 The γ-Calculus

The γ-calculus is a predecessor to the ρ-calculus, very similar to $\rho(x = y, \mathcal{C})$. In this section we give a brief presentation of its syntax and semantics. For a fuller treatment we refer the reader to [Smo94].

The syntax of the γ-calculus is the same as for the ρ-calculus. In this section we let $a, b, c, \ldots$ range over constants and $x, y, z, \ldots$ over variables. Constants and variables are collectively referred to as *references* (ranged over by $u, v, w, \ldots$). We also include the cell construct in the syntax.

The semantics of the γ-calculus is defined by the structural congruence of the ρ-calculus (Table 6.1) and a reduction relation (Table 6.6). Reduction in γ differs from ρ in some fine points: only constants can be used as subjects in interactions, and equational constraint entailment is implicit by the ELIM rule, which effectuates an equation in much the same way as the SCOPE rule in the fusion calculus effectuates a fusion.

An example of a reduction sequence shows that even though variables cannot be subjects in interactions in the APPL or EXCH rules, they can take part in reductions:

$$\exists x(x = y \wedge \texttt{if } x = y \texttt{ then } E \texttt{ else } E') \xrightarrow{\text{ELIM}}_{\gamma} \texttt{if } y = y \texttt{ then } E \texttt{ else } E' \xrightarrow{\text{THEN}}_{\gamma} E$$

$$
\begin{array}{llll}
\text{STRUCT} & \dfrac{E \equiv E' \quad E' \to_\gamma F' \quad F' \equiv F}{E \to_\gamma F} & & \\
\text{COMP} & \dfrac{E \to_\gamma E'}{E \wedge F \to_\gamma E' \wedge F} & \text{DECL} & \dfrac{E \to_\gamma E'}{\exists u E \to_\gamma \exists u E'} \\
\text{APPL} & \multicolumn{3}{l}{a\tilde{u} \wedge a{:}\,\tilde{x}/E \to_\gamma E\{\tilde{u}/\tilde{x}\} \wedge a{:}\,\tilde{x}/E \quad \text{if } |\tilde{u}| = |\tilde{x}|} \\
\text{THEN} & \multicolumn{3}{l}{\texttt{if } u = u \texttt{ then } E \texttt{ else } F \to_\gamma E} \\
\text{ELSE} & \multicolumn{3}{l}{\texttt{if } a = b \texttt{ then } E \texttt{ else } F \to_\gamma F \ \text{, if } a \neq b} \\
\text{EXCH} & \multicolumn{3}{l}{a{:}\,u \wedge avw \to_\gamma a{:}\,w \wedge u = v} \\
\text{ELIM} & \multicolumn{3}{l}{\exists x(x = u \wedge E) \to_\gamma E\{u/x\} \ \text{ if } x \neq u}
\end{array}
$$

Table 6.6: Reduction rules for the γ-calculus.

6.3.2 Handlers

The main difficulty when encoding the γ-calculus is the treatment of variables. As a result of the ELIM rule a variable can be substituted by another reference. The only mechanism for substitution in the π-calculus is in the interaction between input and output prefixes. Unfortunately we cannot use that mechanism directly to emulate the ELIM rule. A π-calculus name which can be substituted in an interaction must be bound in an input prefix, and this prefix must reduce in the COMM rule before the rest of the agent can reduce. In contrast, in the γ-calculus a term can reduce before its variables have been eliminated.

Our solution to this difficulty is to introduce one π-calculus *handler* agent for each reference not bound by abstraction. A handler interacts with its environment (the encoding of the term where the reference occurs) to perform the substitutions corresponding to the ELIM rule, and to report its current *value*. Originally the value of a handler is the corresponding reference, but the value can change as a result of *updates* emanating from γ-calculus equations. The handlers thus collectively contain a global state corresponding to the eliminations in a reduct.

We distinguish between three kinds of handlers. $C(\underline{a})$ is a handler for a constant a (the underscore will be explained below). It will not update to another reference and always gives its value as a. A variable x which has been updated to a reference u is handled by $R(\underline{x}, \underline{u})$. When queried for its value or asked to update it will relay the request to the handler for u. A variable x not yet eliminated is handled by $V(\underline{x})$. It gives x as value, and when asked to update to u it changes state to $R(\underline{x}, \underline{u})$. In this way we think of C as standing for "constant", V for "variable" and R for "relay" in our encoding.

Since the reference u in $R(\underline{x}, \underline{u})$ may itself be represented by a handler of type R (if u is eliminated after x) the handlers will form a forest. At the root of each tree in the forest sits a handler of type V or C; interior nodes are type R handlers who will relay any queries to its parent. An update of a V root will

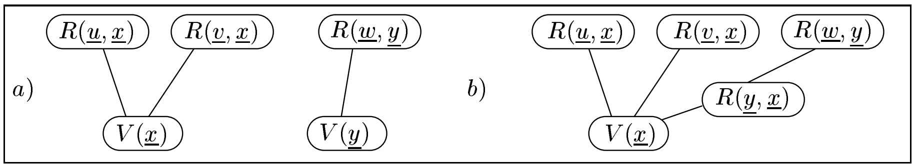

Figure 6.1: Equivalence trees before (a) and after (b) w updates to v. The handler for w relays the update to its root y, which updates to the value of v, namely x.

graft its tree to another tree. Each tree forms an equivalence class of references under the substitutions from ELIM rules. We call such a tree an *equivalence tree*. See Figure 6.1 for an example of how an update affects the equivalence trees.

The γ-calculus is not symmetric in its treatment of variables and constants — e.g., the ELSE rule only applies when a and b are constants — so a handler additionally must report whether its reference is a constant. For the same reason it is convenient to let handlers deal with the queries of equality between references that arise from the conditional terms in the γ-calculus. Such queries can only be resolved in case the references are identical references or distinct constants; an equality between distinct variables must be deferred to a later stage.

To let the handlers perform their tasks we introduce, for each γ-calculus reference u, the four π-calculus names $u{\triangleright}\mathsf{constant}, u{\triangleright}\mathsf{value}, u{\triangleright}\mathsf{update}, u{\triangleright}\mathsf{eq}$, called the *handler interface* names, and the π-calculus name u. We refer collectively to these five names using the notation $\underline{u}$. The symbol $\triangleright$ is part of the names. Below we use a notation $x\colon(a, b, c)$ to specify that names a, b, c are transmitted along x in the following explanation.

- u itself is used for the interaction between encodings of applications and abstractions or cells.
- $u{\triangleright}\mathsf{constant}\colon(a)$ ascertains that u has the γ-calculus constant a as root in its equivalence tree.
- $u{\triangleright}\mathsf{value}\colon(c, v)$ queries u for its value, by passing a π-calculus name c where the value is to be received, and the identity v of the source of the query. The value of a reference is the root of its equivalence tree. The source of the query, v, is used to avoid circular queries.
- $u{\triangleright}\mathsf{update}\colon(\underline{v}, w)$ tells u to update to the reference v. Care must be taken to avoid updates resulting in circular equivalence trees. The name w is used to detect a constant being updated to another constant.
- $u{\triangleright}\mathsf{eq}\colon(\underline{v}, y, n)$ asks u to decide whether it is in the same equivalence tree as v, and if it is, to signal on y, or if they are both names and not equal, to signal on n.

$$
\begin{array}{rcl}
\underline{u} & \equiv & u, u\triangleright\mathsf{constant}, u\triangleright\mathsf{value}, u\triangleright\mathsf{update}, u\triangleright\mathsf{eq} \\
 & & \\
C(\underline{a}) & \equiv & \ !\,\overline{a\triangleright\mathsf{constant}}\langle\underline{a}\rangle \\
 & & \mid\ !\,a\triangleright\mathsf{value}(c,\_)\,.\,\overline{c}\langle\underline{a}\rangle \\
 & & \mid\ !\,a\triangleright\mathsf{update}(\underline{u},w)\,.\,[a=u](\mathbf{0},[a=w](\mathit{fail},\overline{u\triangleright\mathsf{update}}\langle\underline{a}\rangle)) \\
 & & \mid\ !\,a\triangleright\mathsf{eq}(\underline{u},y,n)\,.\,u\triangleright\mathsf{constant}(\underline{b})\,.\,[a=b](\overline{y},\overline{n}) \\
 & & \\
V(\underline{x}) & \stackrel{def}{=} & \ x\triangleright\mathsf{value}(c,\_)\,.\,\overline{c}\langle\underline{x}\rangle\,.\,V(\underline{x}) \\
 & & +x\triangleright\mathsf{update}(\underline{u},\_)\,. \\
 & & \quad [x=u]\Big(\tau\,.\,V(\underline{x}),\overline{u\triangleright\mathsf{value}}\langle\boldsymbol{\nu}c,x\rangle\,.\,c(\underline{u})\,. \\
 & & \qquad\qquad [x=u](\tau\,.\,V(\underline{x}),\tau\,.\,R(\underline{x},\underline{u}))\Big) \\
 & & +x\triangleright\mathsf{eq}(\underline{u},y,n)\,. \\
 & & \quad \Big(\ V(\underline{x}) \\
 & & \quad\ \mid \overline{u\triangleright\mathsf{value}}\langle\boldsymbol{\nu}c,x\rangle\,.\,c(\underline{u})\,.\,[x=u](\overline{y},\overline{x\triangleright\mathsf{eq}}\langle\underline{u},y,n\rangle)\Big) \\
 & & \\
R(\underline{x},\underline{u}) & \equiv & \ !\,u\triangleright\mathsf{constant}(\underline{a})\,.\,\overline{x\triangleright\mathsf{constant}}\langle\underline{a}\rangle \\
 & & \mid\ !\,x\triangleright\mathsf{value}(c,v)\,.\,[u=v]\Big(\overline{c}\langle\underline{u}\rangle,\overline{u\triangleright\mathsf{value}}\langle c,v\rangle\Big) \\
 & & \mid\ !\,x\triangleright\mathsf{update}(\underline{v},w)\,.\,\overline{u\triangleright\mathsf{update}}\langle\underline{v},w\rangle \\
 & & \mid\ !\,x\triangleright\mathsf{eq}(\underline{v},y,n)\,.\,\overline{u\triangleright\mathsf{eq}}\langle\underline{v},y,n\rangle
\end{array}
$$

Table 6.7: Definitions of handler processes.

Finally, note that variables bound by abstraction have no handlers. This is because they cannot interact until the abstraction has reduced, and so the APPL rule can be emulated with the COMM rule of π.

6.3.3 The Encoding

The definitions of handlers can now be presented in Table 6.7.

A handler $C(\underline{a})$ never changes state to another type of handler; it can always handle requests and therefore uses replication in its definition. If it is asked to update to some other reference u through $a\triangleright\mathsf{update}(\underline{u}, w)$ it first checks if it is asked to update to itself, it not so it checks to see if the request has been reflected to itself, otherwise it simply reflects the request and asks u to update a instead. This possibly leads to *fail* being observable, if u is itself a constant. In an equality check $a\triangleright\mathsf{eq}$, the other reference is established to be a γ-calculus constant, and this constant and a itself are compared using a π-calculus conditional, and y or n is signalled appropriately.

The definition of a handler $V(\underline{x})$ is more intricate. When handling $x\triangleright\mathsf{update}(\underline{u})$ to change into $R(\underline{x}, \underline{u})$, we must make sure the equivalence tree does not become circular; if it did, the handlers would end up relaying requests to each other.

$$
\begin{array}{rcl}
[\![\top]\!] & = & \mathbf{0} \\
[\![E \wedge F]\!] & = & [\![E]\!] \mid [\![F]\!] \\
[\![\exists a E]\!] & = & (\boldsymbol{\nu}\underline{a})(C(\underline{a}) \mid [\![E]\!]) \\
[\![\exists x E]\!] & = & (\boldsymbol{\nu}\underline{x})(V(\underline{x}) \mid [\![E]\!]) \\
[\![u\colon \tilde{x}/E]\!] & = & !\, u\triangleright\mathsf{constant}(\underline{a}) \,.\, a(\underline{\tilde{x}}) \,.\, [\![E]\!] \\
[\![u\colon v]\!] & = & u\triangleright\mathsf{constant}(\underline{a}) \,.\, a(\underline{v}, \underline{w}) \,.\, \Big([\![u = v \wedge a\colon w]\!]\Big) \\
[\![u\tilde{v}]\!] & = & u\triangleright\mathsf{constant}(\underline{a}) \,.\, \overline{a}\langle\underline{\tilde{v}}\rangle \\
[\![u = v]\!] & = & \overline{u\triangleright\mathsf{update}}\langle\underline{v}, v\rangle \\
\multicolumn{3}{l}{[\![\texttt{if } u = v \texttt{ then } E \texttt{ else } F]\!]} \\
 & = & (\boldsymbol{\nu}y, n)(\overline{u\triangleright\mathsf{eq}}\langle\underline{v}, y, n\rangle \mid y \,.\, [\![E]\!] \mid n \,.\, [\![F]\!])
\end{array}
$$

Table 6.8: Encoding of γ-calculus terms.

This could happen in two cases: (1) if $V(\underline{x})$ is asked to update to itself or to a reference in the same equivalence tree, so we must test for this situation by reading the value of the target u and check if it is x using a π-calculus conditional. Circularity can also occur (2) if two or more variables try to update to each other simultaneously. This can be avoided by not allowing any other variable to update to x while $V(\underline{x})$ is handling an $x\triangleright\mathsf{update}$ request. This is accomplished by *not* handling $x\triangleright\mathsf{value}$ requests during update sections, and vice versa, since another variable must read the value of the reference it is updating to. But this means that in the case that x is told to update to itself, $V(\underline{x})$ now cannot respond to its own $x\triangleright\mathsf{value}$ query. We therefore add a conditional which tests the "un-evaluated" target for equality with x. The case that x is told to update to itself through a number of relay handlers must also be taken care of: the last relay handler in the chain cannot get a reply to its $x\triangleright\mathsf{value}$ query. This situation is dealt with in the $x\triangleright\mathsf{value}$ case of the R handlers, where the R handler will respond on behalf of the source of the request, which in this case will be $V(\underline{x})$.

The treatment of $x\triangleright\mathsf{eq}$ requests in a $V(\underline{x})$ handler is straightforward: the value of the other reference is read, and if it is x itself, we generate a positive response. Otherwise the request is reiterated, deferring the response until a decision can be reached, i.e., until either both have updated to constants, or both are in the same equivalence tree.

The $R(\underline{x}, \underline{u})$ handler is almost transparent. On receipt of $u\triangleright\mathsf{constant}(\underline{a})$, the relay can respond to $x\triangleright\mathsf{constant}$ requests. The $x\triangleright\mathsf{update}$ and $x\triangleright\mathsf{eq}$ requests are simply relayed to the handler for u. The treatment of $x\triangleright\mathsf{value}$ was mentioned above: if the source of the request is the handler of u, it may be in a state where it cannot respond to its own query, so the R handler responds by proxy. If the source was some other reference, the request is relayed.

Given the handling of references, the encoding of γ-calculus terms themselves can be presented in Table 6.8. Null of the γ-calculus ($\top$) corresponds to null of the π-calculus ($\mathbf{0}$), and $\wedge$ (composition) to $\mid$. Declaration $\exists$ corresponds to restriction $\boldsymbol{\nu}$, with the addition of a handler for the reference being declared.

The encoding of a γ-calculus cell ensures that its location is a γ-calculus constant, inputs the "output location" and new value, and proceeds as the encoding of an equation and the cell holding the new value. An encoded γ-calculus application must also ensure its location to be a γ-calculus constant, and then outputs the vector of references together with their handler interface names. A γ-calculus abstraction is encoded as a replication which first makes sure its location is a γ-calculus constant, inputs the references along with their handler interface names, and proceeds as the encoding of the body of the abstraction. The replication is necessary since, in the APPL rule, the abstraction recurs.

Variants of the Encoding

The use of summation (the $+$ operator) in the definition of type V reference handlers can be avoided by using a standard π-calculus technique: we replace the summation of inputs on "selector" names $a_1, \ldots, a_n$, $a_1(\tilde{x}_1) . P_1 + \cdots + a_n(\tilde{x}_n) . P_n$, with *one* input on a "contact" name b: $b(a_1, \ldots, a_n) . (a_1(\tilde{x}_1) . P_1 \mid \cdots \mid a_n(\tilde{x}_n) . P_n)$, where $\forall i : \{a_j : j \neq i\} \cap \mathrm{fn}(P_i) = \emptyset$, and replace the "selector" outputs $\overline{a_i}\langle\tilde{c}_i\rangle . P_i$ with $\overline{b}\langle\boldsymbol{\nu} a_1, \ldots, \boldsymbol{\nu} a_n\rangle . \overline{a_i}\langle\tilde{c}_i\rangle . P_i$, with the same restriction on $\mathrm{fn}(P_i)$. This also makes handler interface names more local. Our encoding introduces divergence in several places. The reiteration of the $\overline{x \triangleright \mathsf{eq}}\langle\underline{u}, y, n\rangle$ request in the definition of $V(\underline{x})$ can also result in a divergence. This can be avoided by "freezing" the request by putting it under a prefix (local to $V(\underline{x})$), which is activated if x is later updated, possibly deciding the condition. Finally, indefinitely iterated checks that a reference is a constant, i.e., interactions along $u \triangleright \mathsf{constant}$, occur in relay handlers and in abstractions. Again divergence can be avoided if these are modified to only perform the check once. This is enough since whenever the check succeeds it will always continue to succeed — a constant can never cease to be a constant.

Divergence does not affect the formal correctness proofs to follow, since termination is not among the observables in our notion of correctness. In other words, the fact that a γ-term terminates but its encoding diverges is not considered relevant. This can be construed as a weakness from the point of view of the γ-calculus and Oz programming. By adopting the variants above we can eliminate divergence at the expense of a more complicated encoding. Although we have not carried out all details of the proofs, we strongly conjecture that the correctness results of Section 6.3.6 are still valid in these variants.

In the following subsections we show that our encoding is correct, in the sense that if two γ-calculus terms are behaviourally equivalent, we know that the encodings of these terms into the π-calculus are also equivalent, and vice versa. We also show that γ-calculus reductions correspond to π-calculus reductions in our encoding.

6.3.4 Free References

As in [Smo94], we assume that there are no free γ-calculus variables at top level of a term. However, free γ-calculus constants may still appear. To complete the encoding of terms $[\![\cdot]\!]$, we must add a C handler for each free constant. Also there may be more free π-calculus names than there are free γ-calculus constants. E.g., for the term $E \equiv a\!: x/ax$, $\mathrm{fn}(E) = \{a\}$, while $\mathrm{fn}([\![E]\!]) = \{\underline{a}\} = \{a, a{\triangleright}\mathsf{constant}, a{\triangleright}\mathsf{value}, a{\triangleright}\mathsf{update}, a{\triangleright}\mathsf{eq}\}$. To correct this and make sure the handler interface names cannot be used improperly by some external agent, we add a restriction on them.

We form the set $\mathcal{H}$ by all handler interface names. We define a closing operation $\mathcal{V}$ to add the constant handlers and the restriction:

Definition 54 *If* $\mathrm{fn}(P) \setminus \mathcal{H} = \{a_1, \ldots, a_n\}, n \geq 0$,

$$\mathcal{V}(P) \equiv (\boldsymbol{\nu}\tilde{a})(C(\underline{a_1}) \mid \cdots \mid C(\underline{a_n}) \mid P)$$

where $\tilde{a} \equiv \bigcup_{i=1}^{n} \{a_i{\triangleright}\mathsf{constant}, a_i{\triangleright}\mathsf{value}, a_i{\triangleright}\mathsf{update}, a_i{\triangleright}\mathsf{eq}\}$

So $\mathrm{fn}(P) \setminus \mathcal{H} = \mathrm{fn}(\mathcal{V}(P))$. When there is no risk of confusion, we sometimes write $\mathcal{V}(E)$ for $\mathcal{V}([\![E]\!])$. An example:

$$\begin{aligned} E &\equiv a\!: x/ax \\ [\![E]\!] &\equiv\ !\,a{\triangleright}\mathsf{constant}(\underline{a}) \,.\, a(\underline{x}) \,.\, a{\triangleright}\mathsf{constant}(\underline{a}) \,.\, \overline{a}\langle \underline{x} \rangle \\ \mathcal{V}([\![E]\!]) &\equiv (\boldsymbol{\nu} a{\triangleright}\mathsf{constant}, a{\triangleright}\mathsf{value}, a{\triangleright}\mathsf{update}, a{\triangleright}\mathsf{eq})\Big(C(\underline{a}) \mid [\![E]\!]\Big) \end{aligned}$$

6.3.5 Barbed Equivalences

The barbed bisimulation framework is easy to apply to the γ-calculus reduction system once we have defined an observation predicate. We choose to observe only constant subjects, as they are the only ones capable of interaction.

Definition 55 (Observations in the γ-calculus)

$$\begin{array}{ll} a\tilde{v} \downarrow_\gamma a & \\ a\!: \tilde{x}/E \downarrow_\gamma a & \\ a\!: u \downarrow_\gamma a & \\ (\exists u E) \downarrow_\gamma a & \textit{if } E \downarrow_\gamma a \textit{ and } a \neq u \\ (E \wedge F) \downarrow_\gamma a & \textit{if } E \downarrow_\gamma a \textit{ or } F \downarrow_\gamma a \end{array}$$

We write $E \stackrel{\bullet}{\approx}_\gamma F$ is E is weak barbed bisimilar to F, using Definition 35 of weak barbed bisimulation, and construct the corresponding congruence by closing under arbitrary γ-calculus contexts:

Definition 56 *Two terms E and F are* weak γ-barbed congruent*, written $E \approx_\gamma F$, if for all γ-calculus contexts $C[\cdot]$, it holds that $C[E] \stackrel{\bullet}{\approx}_\gamma C[F]$.*

In the π-calculus case, we use the standard definition of barbed bisimulation to obtain $\dot{\approx}_\pi$, but choose to close not under all contexts of the calculus as in Section 4.3. Instead we close only under γ contexts, i.e., contexts occurring in encodings of γ-calculus terms. Formally, these can be defined by introducing term variables; the γ contexts are then encodings of γ terms with one occurrence of a term variable. We also close the congruence under the $\mathcal{V}$ operation defined above.

Definition 57 *Two agents P and Q are* weak π_γ-barbed congruent*, written $P \approx_\pi Q$, if for all γ contexts $C[\cdot]$, it holds that $\mathcal{V}(C[P]) \dot{\approx}_\pi \mathcal{V}(C[Q])$.*

We recall that the standard weak barbed congruence in the π-calculus coincides with weak early bisimulation [San93]. Our congruence is coarser since only contexts corresponding to the γ-calculus are considered. For example, the terms $a{:}\,x/(x = x)$ and $a{:}\,x/\top$ are equivalent in the sense that their encodings have the same reductions and observations in arbitrary γ-calculus contexts. But there is a π-calculus context which distinguishes them, for example $(\overline{a}\langle \underline{u} \rangle \mid \cdot)$. Inserting the encoding of $a{:}\,x/(x = x)$ we will get observations along the interface names in $\underline{u}$, in contrast to when inserting $a{:}\,x/\top$. A similar situation is present in the encoding of the λ-calculus in Section 6.1. To make an encoding robust under alien contexts, the approach of Fournet and Gonthier [FG96] of adding a "firewall" around the encoding proper could be applied; we choose not to.

We now have definitions of the congruences we will use in the correctness theorem below. However, to make the proofs easier, we use the *up-to* $\dot{\approx}$ technique of [MS92b, San93]. we give a parameterized definition. Let $\Box$ be γ or π.

Definition 58 *A symmetric relation $\mathcal{R}$ on $\Box$ expressions is a* weak barbed bisimulation up-to $\dot{\approx}_\Box$ *if for all $(X, Y) \in \mathcal{R}$,*

1. *if $X \rightarrow^*_\Box X'$, then $Y \rightarrow^*_\Box Y'$ such that $X' \dot{\approx}_\Box \mathcal{R} \dot{\approx}_\Box Y'$*
2. *if $X \rightarrow^*_\Box \downarrow_\Box a$, then $Y \rightarrow^*_\Box \downarrow_\Box a$.*

We write $X \dot{\approx}^+_\Box Y$ if X and Y are related by a weak barbed bisimulation up-to $\dot{\approx}_\Box$.

Again, we close these bisimilarities under contexts.

Definition 59 *Two terms E and F are* weak γ-barbed congruent up-to $\dot{\approx}_\gamma$*, written $E \approx^+_\gamma F$, if for all γ-calculus contexts $C[\cdot]$, it holds that $(C[E], C[F]) \in \mathcal{R}$ for some $\mathcal{R}$ which is a weak barbed bisimulation up-to $\dot{\approx}_\gamma$.*

Definition 60 *Two agents P and Q are* weak π_γ-barbed congruent up-to $\dot{\approx}_\pi$*, written $P \approx^+_\pi Q$, if for all γ contexts $C[\cdot]$, it holds that $(\mathcal{V}(C[P]), \mathcal{V}(C[Q])) \in \mathcal{R}$ for some $\mathcal{R}$ which is a weak barbed bisimulation up-to $\dot{\approx}_\pi$.*

Proposition 6.8

1. *If* $E \approx_\gamma^+ F$*, then* $E \approx_\gamma F$*.*
2. *If* $P \approx_\pi^+ Q$*, then* $P \approx_\pi Q$*.*

PROOF: Diagram-chasing, analogous to Lemma 4.5 in [Mil89], page 93.

6.3.6 Correctness

We now have enough machinery to proceed to show the correctness of the encoding, i.e., that two γ-calculus terms are congruent iff their encodings in the π-calculus are congruent.

Theorem 6.9 $E \approx_\gamma F$ *iff* $[\![E]\!] \approx_\pi [\![F]\!]$*.*

In the rest of this section we sketch the proof.

We use straight-forward Lemmas concerning operational correspondence between γ-calculus terms E and their encodings $\mathcal{V}(E)$. The first Lemma says that if a γ-calculus term E can reduce to E', its encoding $\mathcal{V}(E)$ can reduce to an agent congruent to the encoding of E'.

Lemma 6.10 $E \rightarrow_\gamma E'$ *implies* $\mathcal{V}(E) \rightarrow_\pi^* \dot{\approx}_\pi \mathcal{V}(E')$*.*
PROOF: By induction over reduction rules. The interesting case is ELIM.
Then

$$\begin{array}{rcl} E & \equiv & \exists x(x = u \wedge E') \rightarrow_\gamma E'\{u/x\} \\ \mathcal{V}(E) & \equiv & \mathcal{V}((\boldsymbol{\nu}\underline{x})(V(\underline{x}) \mid \overline{x \triangleright \mathsf{update}}\langle \underline{u}, u\rangle \mid [\![E']\!])) \end{array}$$

and $\mathcal{V}(E) \rightarrow_\pi^* \mathcal{V}((\boldsymbol{\nu}\underline{x})(R(\underline{x}, \underline{u}) \mid [\![E']\!]))$. We need to show that

$$\mathcal{V}((\boldsymbol{\nu}\underline{x})(R(\underline{x}, \underline{u}) \mid [\![E']\!])) \dot{\approx}_\pi \mathcal{V}([\![E'\{u/x\}]\!]))$$

Since the R handler is mainly "transparent", the only interesting case is when $R(\underline{x}, \underline{u})$ interacts over $x \triangleright \mathsf{value}$; this can only happen when a Variable handler $V(\underline{v})$ is asked to update to x (or a Relay node above x in its equivalence tree), i.e., when E' contains the subterm $v = x$ (or correspondingly). The case to study here is when x and v are in the same equivalence tree, and v is the immediate ancestor of x (i.e., $v \equiv u$). Then the response to the $x \triangleright \mathsf{value}$ request is $\underline{v}$, which makes $V(\underline{v})$ avoid updating, which is also the result in what $\mathcal{V}(E'\{u/x\})$, where the subterm is simply $x = x$. □

An immediate corollary is the following:

Corollary 6.11 $E \rightarrow_\gamma^* E'$ *implies* $\mathcal{V}(E) \rightarrow_\pi^* \dot{\approx}_\pi \mathcal{V}(E')$*.*

The next Lemma goes in the other direction: it states that however $\mathcal{V}(E)$ reduces in the π-calculus, it reduces to an agent which is congruent to the encoding of a γ-calculus term to which E can reduce in the γ-calculus.

Lemma 6.12 *$\mathcal{V}(E) \to_\pi^* P'$ implies $\exists E' : E \to_\gamma^* E'$ and $P' \dot{\approx}_\pi \mathcal{V}(E')$.*
PROOF: By induction on the structure of terms E.

We move on to the observations: if we can observe a on E, we can also observe a on the encoding of E.

Lemma 6.13 *$E \downarrow_\gamma a$ implies $\mathcal{V}(E) \to_\pi^* \downarrow a$.*
PROOF: By induction on the depth of inference of the observation.

A corollary of this (and Corollary 6.11) is the following:

Corollary 6.14 *$E \to_\gamma^* \downarrow_\gamma a$ implies $\mathcal{V}(E) \to_\pi^* \downarrow a$.*

The other direction is also true: if we can observe a on the encoding of a term E, we can observe it on E itself.

Lemma 6.15 *$\mathcal{V}(E) \to_\pi^* \downarrow a$ implies $E \to_\gamma^* \downarrow_\gamma a$.*
PROOF: By induction on the structure of terms E.

Proposition 6.16 *$E \approx_\gamma^+ F$ iff $[\![E]\!] \approx_\pi^+ [\![F]\!]$.*
PROOF: First, show that

$$\mathcal{S} = \{(E, F) : \mathcal{V}([\![E]\!]) \dot{\approx}_\pi^+ \mathcal{V}([\![F]\!])\}$$

is a weak barbed bisimulation up-to $\dot{\approx}_\gamma$. If $E \to_\gamma^* E'$, then by Corollary 6.11 we know $\mathcal{V}(E) \to_\pi^* \dot{\approx}_\pi \mathcal{V}(E')$, and thus $\mathcal{V}(F) \to_\pi^* Q'$ s.t. $\mathcal{V}(E') \dot{\approx}_\pi Q'$. But then by Lemma 6.12, $F \to_\gamma^* F'$ such that $Q' \dot{\approx}_\pi \mathcal{V}(F')$, so $\mathcal{V}(E') \dot{\approx}_\pi \mathcal{V}(F')$, and $(E', F') \in \mathcal{S}$. If $E \to_\gamma^* \downarrow_\gamma x$, then by Corollary 6.14 $\mathcal{V}(E) \to_\pi^* \downarrow x$, and $\mathcal{V}(F) \to_\pi^* \downarrow x$, so by Lemma 6.15 $F \to_\gamma^* \downarrow_\gamma x$. For the congruence, we need that $\forall C[\cdot] : C[E] \dot{\approx}_\gamma^+ C[F]$ if $\mathcal{V}([\![C[E]]\!]) \dot{\approx}_\pi^+ \mathcal{V}([\![C[F]]\!])$, which follows from the compositionality of the encoding $[\![\cdot]\!]$.

For the other direction, show that $\mathcal{S} = \{(\mathcal{V}(E), \mathcal{V}(F)) : E \dot{\approx}_\gamma^+ F\}$ is a weak barbed bisimulation up-to $\dot{\approx}_\pi$. If $\mathcal{V}(E) \to_\pi^* P'$, then by Lemma 6.12 $E \to_\gamma^* E'$ s.t. $P \dot{\approx}_\pi \mathcal{V}(E')$, and $F \to_\gamma^* F'$ s.t. $E' \dot{\approx}_\gamma F'$, which by Corollary 6.11 means that $\mathcal{V}(F) \to_\pi^* \dot{\approx}_\pi \mathcal{V}(F')$ and $(\mathcal{V}(E'), \mathcal{V}(F')) \in \mathcal{S}$. Furthermore if $E \to_\gamma^* \downarrow_\gamma x$ then by Corollary 6.14 $\mathcal{V}(E) \to_\pi^* \downarrow x$, and $F \to_\pi^* \downarrow_\gamma x$ so $\mathcal{V}(F) \to_\pi^* \downarrow x$ by the same corollary. For congruence, we need $\forall C[\cdot] : \mathcal{V}([\![C[E]]\!]) \dot{\approx}_\pi^+ \mathcal{V}([\![C[F]]\!])$ if $C[E] \dot{\approx}_\gamma^+ C[F]$, which again follows from the compositionality of the encoding. □

Now, Theorem 6.9 is a consequence of Propositions 6.16 and 6.8.

6.4 Conclusion

The expressiveness of the fusion calculus makes it possible to give a very direct encoding of the strong lazy reduction strategy of λ-calculus, by use of the delayed input construct, which needs no special treatment in the fusion calculus. The encodings of the ρ-calculus variants are also quite direct, although we need extra names and processes. Encoding the γ-calculus in the π-calculus turns out to be much more complex, with an involved protocol for communication between the processes representing constants, variables and relays.

Chapter 7

Automatic Verification

A formal method is practically useless if it does not come with tools for automated analysis. To quote Gerard Holzmann [Hol91, p 243] in the context of communication protocols,

> Without the proper tools it may be possible to design a correct protocol, but in most cases it will be impossible to formally establish its correctness with any measure of reliability.

It is therefore natural to work towards automated tools for formal methods, and in this section we apply the theory of symbolic bisimulation to derive algorithms for automatic equivalence checking for the fusion calculus, and describe our implementation in the Mobility Workbench [VM94].

We give a simple example of fusion calculus verification using the Mobility Workbench and a larger one for the π-calculus.

7.1 Algorithms

In this section we present algorithms to decide the symbolic bisimulation equivalences. These equivalences are decidable for *finite-control* processes, corresponding to finite state processes in CCS; these have finite symbolic transition systems, and can be characterized syntactically by disallowing parallel composition within recursively defined agents.

We use *on-the-fly* algorithms which build the state spaces of the agents as they are explored, in contrast with the partition-refinement algorithms [PT87, KS90] usually employed for finite-state systems in, e.g., CCS. There, the state spaces of the agents are constructed independently of each other, and then their union is partitioned in successively smaller parts, terminating when a bisimulation relation is found. Because of the side conditions on bound names in Definitions 36 and 40, it is not simple to generate the state spaces of fusion calculus or π-calculus agents independently. Pistore and Sangiorgi [PS96b] have developed a partition refinement algorithm for checking open bisimulation for π-calculus agents. As future work, we intend to investigate their work in the setting of the fusion calculus; we expect to find simplifications since we do not have the complication of distinctions. On the other hand, as we have seen, a mismatch operator forces us to find a simulating *set* of transitions rather than only a single one.

The advantages of an on-the-fly algorithm is that it is easy to implement, and that it may return a negative answer even for non-finite-control agents. It can also be more efficient than partition refinement algorithms on agents with a limited degree of non-determinism.

The same basic algorithm can be used to decide symbolic hyperequivalences and symbolic open bisimulation for finite-control agents. The algorithm follows Definitions 36 and 40 closely. Given two agents P and Q, a match/mismatch sequence C and a relation R, check if there is a triple $(P', Q', C') \in R$ such that $C \Rightarrow C'$ and $C' \Rightarrow (P \equiv P' \wedge Q \equiv Q')$, and if so, return the relation unchanged. Otherwise, for each transition $P \xmapsto{M,\gamma} P'$, find a subset T of the (strong or weak) transitions of Q such that the conditions match appropriately and the actions are equivalent under a substitution agreeing with the conditions. First each singleton subset of the transitions of Q is tried, then incrementally larger subsets not containing transitions which have been seen not to simulate; if there is no such subset P and Q are not bisimilar. To efficiently decide the implication $CM \Rightarrow \bigvee N_i$, we first eliminate all tautological pairs ($[x = y] \vee [x \neq y]$) in $\bigvee N_i$, and then see if CM implies one of the remaining matches. If so then assume that P and Q are bisimilar under the condition C (by adding the triple (P, Q, C) to R), recurse over the derivatives with the condition L_i according to the appropriate definition (36 or 40). This either fails, causing the current recursion to try another subset T (or fail if there is none), or returns a relation containing (P, Q, C) which is used in subsequent recursive calls when matching the other transitions of P. Finally, when all transitions of P have been simulated, find simulating transition sets of P for each transition of Q in the same way, returning the resulting relation.

The problem of deciding bisimulation equivalence of data-independent agents in value-passing CCS is known to be NP-hard in the size of the agents [JP93], which implies that the decision problem for the fusion calculus is at least this hard.

7.2 The Mobility Workbench

The main motivation for the symbolic equivalences presented in sections 4.4 and 2.2 is to efficiently decide the equivalences in an automated tool, and we have implemented the above algorithms in the Mobility Workbench (MWB). The MWB is available through the URL `http://www.docs.uu.se/~victor/mwb.html`.

The Mobility Workbench is an automated tool for analyzing systems of mobile processes, initially developed for the π-calculus by the present author and Faron Moller [VM94] based on the symbolic characterization of open bisimulation equivalences by Davide Sangiorgi [San96c]. It was later extended with weak open bisimulation checking [Vic94], a sort inference algorithm [Gay93], and by Mads Dam and Lars-Henrik Eriksson with a model checker [Dam96]. Recently the model checker was reimplemented using symbolic semantics [Fra96, Bes98], with a significant speedup as a result. Finally we are now extending the MWB to handle the fusion calculus.

To improve efficiency in the MWB, agents are represented using De Bruijn indexing for names [dB72], making alpha-conversion unnecessary. In this representation every name is represented by an index (a number) representing its *reference depth*, saying how far it is to its binding. The scope operator no longer takes an argument saying which name is bound, but only indicates that a binding is taking place. For example, the agent $(uxy)(\overline{u}\,x \mid u\,y)$ is represented as $()()()(\overline{2}\,1 \mid 2\,0)$. All alpha-equivalent agents have the same representation; e.g., $(abc)(\overline{a}\,b \mid a\,c)$ has the same representation.

An occurrence of a name is *free* in an agent if its reference depth is greater than or equal to its level (the number of scope operators surrounding the occurrence). For example, the name c is free in

$$(a)(b)a\,b\,.\,[b=c]\overline{b}\,a\,.\,\mathbf{0}$$

since the index in its representation is equal to the number of scope operators surrounding it (two):

$$()()1\,0\,.\,[0=2]\overline{0}\,1\,.\,\mathbf{0}$$

Names which are free on the top level are represented in correspondence to this, using an ordered list of names as a "global binding". So the first free name is represented by 0 (at top level), the second by 1, etc.

Computing the transitions of an agent can be quite costly if the structure of the process is complex. To reduce the amount of work, every time we compute the transitions of a process we record them in a hash table. Whenever we want to find the transitions of an agent, we first look up the process in the hash table, and only if there are no recorded transitions do we actually go to the trouble of computing them. The hash tables are at present kept at most 30% full, and automatically resize and rehash when they become fuller.

This speeds up verification considerably in the MWB, but requires more space since we store all agents (and subagents) we have explored. In the current version the space requirements are significantly reduced by minimizing the number of substitutions performed on agents. This is done by representing equivalence between names caused by fusion actions explicitly in the condition C of the bisimulation definition. This avoids copying of agents when a substitution is performed.

An example of the decrease in space usage is the verification of the GSM handover protocol specified in [OP92], conducted in Section 7.4, which in the previous version of the MWB used 274 megabyte of heap space, while the new version needs only 53 megabyte, making it reasonable to use a normal workstation for verifying such examples without a large time penalty for virtual memory handling.

7.3 A Simple Example: Critical Sections

As a simple example of using the MWB for fusion calculus examples, consider the well-known Critical Section problem: two processes share some data, and need to have exclusive access to the data while modifying them. This can be achieved by defining a *critical section* in the processes with the properties that if one process is in its critical section, no other process can be in its critical section, and if a process wants to enter its critical section, it cannot be stopped by processes not wanting to enter theirs.

An abstract specification in the fusion calculus of a solution to the problem for two processes is

$$\mathit{Spec} \stackrel{def}{=} \mathbf{1} \,.\, b_1 \,.\, \mathbf{1} \,.\, e_1 \,.\, \mathit{Spec} + \mathbf{1} \,.\, b_2 \,.\, \mathbf{1} \,.\, e_2 \,.\, \mathit{Spec}$$

which states that without external control, a process may enter its critical section (b_i for "begin"), then modify the data (the internal or invisible action $\mathbf{1}$), and exit the critical section (e_i for "end"), whereupon any process may again enter its critical section.

A simple protocol for solving the critical section problem is by using a *test-and-set* instruction, which atomically tests and sets a (shared) flag variable. A type of test-and-set be specified in the fusion calculus by combining mismatch and fusion prefix. The specification of the solution would then be

$$\begin{array}{lcl} P_i(f) & \stackrel{def}{=} & [f \neq s]\{f = s\} \,.\, b_i \,.\, \mathbf{1} \,.\, e_i \,.\, (f')\overline{f}\, f' \,.\, P_i(f') \\ & & +(f')f\, f' \,.\, P_i(f') \end{array}$$

A process P_i can atomically test if the flag is not set ($[f \neq s]$) and if so set it ($\{f = s\}$). Then it enters its critical section (b_i), modifies the data ($\mathbf{1}$) and exits the critical section (e_i). It must then reset the flag: this is done by creating a new flag and sending it to the other process, itself continuing with the new flag. A process which has not entered its critical section can receive a new flag from another process (which has been in its critical section and thus "used up" the old flag). It must then retry entering its critical section using the new flag.

We expect the implementation to be equivalent to the specification, stated as

$$\mathit{Spec} \approx (f)(P_1(f) \mid P_2(f))$$

In Figure 7.1 the MWB code to specify the specification and system of processes can be seen. The syntax of a prefix $a\tilde{x} \,.\, P$ is `a<x1,...,xn>.P`, where a is written simply `a` for an input or `'a` for an output. The command `fagent`

```
fagent A2(flag,set,b,e) = \
        [flag#set]{flag=set}.b.1.e. \
                (g)'flag<g>.A2(g,set,b,e) \
      + (g)flag<g>.A2(g,set,b,e)

fagent Spec2(b1,b2,e1,e2) = \
        1.b1.e1.Spec2(b1,b2,e1,e2) \
      + 1.b2.e2.Spec2(b1,b2,e1,e2)

fagent Sys2(b1,b2,e1,e2) = \
        (flag,set) \
          (A2(flag,set,b1,e1) | A2(flag,set,b2,e2))
```

Figure 7.1: The MWB code for the critical section problem.

```
% mwb

   The Mobility Workbench
   (Version 4.133, built Mon Apr 13 14:11:45 MET DST 1998)

MWB>input "cs.mwb"
MWB>fweq Sys2(b1,b2,e1,e2) Spec2(b1,b2,e1,e2)
The two agents are equal.
Bisimulation relation size = 10.
MWB>quit
```

Figure 7.2: Running the MWB.

defines a fusion agent; note that the definition must be closed — in our formal definitions we sometimes let names be implicit in the argument lists (as in *Spec* or P_i above), but the MWB must be explicitly told about all names in argument lists. It will complain if a definition is not closed.

The code in Figure 7.1 can be entered interactively or into a file, say `cs.mwb`, and then read by MWB using the `input` command. Once the agents have been defined, we can ask the MWB to verify the correctness of the implementation by issuing a `fweq` command (for `f`usion `w`eak `eq`uivalence). The MWB tries to find a symbolic hyperbisimulation and if it does, it reports its size. In Figure 7.2 we give an example session showing how this can all be done.

7.4 An Extended Example: Mobile Telephones

As a larger case study of using the MWB for π-calculus verification, we have specified and verified the core of the handover protocol used in the GSM Public Land Mobile Network (PLMN) proposed by the European Telecommunication Standards Institute (ETSI). The formal specification of the protocol, and its service specification, are due to Orava and Parrow [OP92], who also verified the protocol algebraically. Fredlund and Orava [FO92] later verified the protocol automatically by specifying the protocol in LOTOS [vEVD89], which was translated to labelled transition systems using the Cæsar tool [GS90], which were in turn minimized using the Aldébaran tool [Fer89], and finally compared using Aldébaran. The following informal presentation of the protocol is based on the presentation in [FO92].

The PLMN is a cellular system which can be seen as consisting of Mobile Stations (MSs), Base Stations (BSs), and Mobile Switching Centres (MSCs). The MS, mounted in e.g. a car, provides service to an end user. The BS manages the interface between the MS and a stationary network, controlling all radio communication within a geographical area (a cell). All communication with the MS in a cell is routed through the BS responsible for the cell. The MSC manages a set of BSs, and communicates with them and with other MSCs using a stationary network.

When a MS moves across a cell boundary, the handover procedure changes the communication partner of the MS from the BS of the old cell to the BS of the new cell, ensuring that the MS is constantly in contact with the MSC. The MSC initiates the handover by transmitting a *handover command* message to the MS via the old BS. The handover command message contains parameters enabling the MS to locate the new BS. When transmitting this message the MSC suspends transmission of all messages except for messages related to the handover procedure. When the MS receives the handover command message, it disconnects the old radio links and initiates the new radio links. To establish these connections the MS sends *handover access* messages to the new BS, in order to synchronize with the new BS. When the connections are successfully established, the BS sends a *handover complete* message to the MSC via the new BS. When this message has been received, the network resumes normal operations and releases the old radio links, which are now free and can be allocated to another MS.

In Figure 7.3 we present a π-calculus specification of the protocol. For a discussion on the modelling of the protocol, see [OP92], from which this is taken. In Figure 7.5 we present the MWB code for this specification, which differs from Figure 7.3 in that the argument lists of agent identifier definitions must contain all free names which appear in the agent. (This can sometimes be inconvenient, e.g., in this Figure we had to cut the argument lists to fit in the paper width available).

Correctness of this specification would come from showing that it matched some (ideally simple) service specification which would clearly define the desired

$$
\begin{array}{lcl}
CC(f_a, f_p, l) & \stackrel{def}{=} & \mathtt{in}(v).\overline{f_a}\mathtt{data}.\overline{f_a}v.CC(f_a, f_p, l) \\
& & +l(m_{new}).\overline{f_a}\mathtt{ho\_cmd}.\overline{f_a}m_{new}. \\
& & \quad \Big(\ f_p(c).[c = \mathtt{ho\_com}]\overline{f_a}\mathtt{ch\_rel}.f_a(m_{old}). \\
& & \qquad\qquad \bar{l}m_{old}.CC(f_p, f_a, l) \\
& & \quad +f_a(c).[c = \mathtt{ho\_fail}]\bar{l}m_{new}.CC(f_a, f_p, l)\Big) \\
HC(l, m) & \stackrel{def}{=} & \bar{l}m.l(m).HC(l, m) \\
MSC(f_a, f_p, m) & \stackrel{def}{=} & (\boldsymbol{\nu} l)\Big(HC(l, m) \mid CC(f_a, f_p, l)\Big) \\
BS_a(f, m) & \stackrel{def}{=} & f(c).\Big(\ [c = \mathtt{data}]f(v).\overline{m}\mathtt{data}.\overline{m}v.BS_a(f, m) \\
& & \quad +[c = \mathtt{ho\_cmd}]f(v).\overline{m}\mathtt{ho\_cmd}.\overline{m}v. \\
& & \qquad \Big(\ f(c).[c = \mathtt{ch\_rel}]\overline{f}m.BS_p(f, m) \\
& & \qquad +m(c).[c = \mathtt{ho\_fail}]\overline{f}\mathtt{ho\_fail}.BS_a(f, m)\Big)\Big) \\
BS_p(f, m) & \stackrel{def}{=} & m(c).[c = \mathtt{ho\_acc}]\overline{f}\mathtt{ho\_com}.BS_a(f, m) \\
MS(m) & \stackrel{def}{=} & m(c).\Big(\ [c = \mathtt{data}]m(v).\overline{\mathtt{out}}v.MS(m) \\
& & \quad +[c = \mathtt{ho\_cmd}]m(m_{new}). \\
& & \qquad \Big(\ \overline{m_{new}}\mathtt{ho\_acc}.MS(m_{new}) \\
& & \qquad +\overline{m}\mathtt{ho\_fail}.MS(m)\Big)\Big) \\
P(f_a, f_p) & \equiv & (\boldsymbol{\nu} m)\Big(MSC(f_a, f_p, m) \mid BS_p(f_p, m)\Big) \\
Q(f_a) & \equiv & (\boldsymbol{\nu} m)\Big(BS_a(f_a, m) \mid MS(m)\Big) \\
\mathit{System} & \equiv & (\boldsymbol{\nu} f_a)(\boldsymbol{\nu} f_p)\Big(P(f_a, f_p) \mid Q(f_a)\Big)
\end{array}
$$

Figure 7.3: A formal specification of the handover procedure.

$$
\begin{array}{lcl}
\mathit{Spec} & \stackrel{def}{=} & \mathtt{in}(v).S_1(v) \ + \ \tau.\mathit{Spec} \\
S_1(v) & \stackrel{def}{=} & \mathtt{in}(v).S_2(v_1, v) \ + \ \overline{\mathtt{out}}v_1.\mathit{Spec} \ + \ \tau.\overline{\mathtt{out}}v_1.\mathit{Spec} \\
S_2(v_1, v_2) & \stackrel{def}{=} & \mathtt{in}(v).S_3(v_1, v_2, v) \ + \ \overline{\mathtt{out}}v_1.S_1(v_2) \ + \ \tau.\overline{\mathtt{out}}v_1.\overline{\mathtt{out}}v_2.\mathit{Spec} \\
S_3(v_1, v_2, v_3) & \stackrel{def}{=} & \overline{\mathtt{out}}v_1.S_2(v_2, v_3)
\end{array}
$$

Figure 7.4: A formal specification of the service specification.

```
agent CC(fa,fp,l,in,data,ho_cmd,ho_com,ho_fail,ch_rel) =
     in(v).'fa<data>.'fa<v>.CC(fa,fp,l,in,data,...)
   + l(mnew).'fa<ho_cmd>.'fa<mnew>.
       (fp(c).[c=ho_com]'fa<ch_rel>.fa(mold).'l<mold>.CC(fp,fa,l,...)
       +fa(c).[c=ho_fail]'l<mnew>.CC(fa,fp,l,...)

agent HC(l,m) = 'l<m>.l(m).HC(l,m)

agent MSC(fa,fp,m,in,data,ho_cmd,ho_com,ho_fail,ch_rel) =
     (^l)(HC(l,m) | CC(fa,fp,l,in,data,ho_cmd,ho_com,ho_fail,ch_rel))

agent BSa(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail) =
     f(c).( [c=data]f(v).'m<data>.'m<v>.BSa(f,m,ho_acc,...)
          + [c=ho_cmd]f(v).'m<ho_cmd>.'m<v>.
               (f(c).[c=ch_rel]'f<m>.BSp(f,m,ho_acc,...)
               +m(c).[c=ho_fail]'f<ho_fail>.BSa(f,m,...)

agent BSp(f,m,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail) =
     m(c).[c=ho_acc]'f<ho_com>.BSa(f,m,ho_acc,ho_com,data,ho_cmd,...)

agent MS(m,data,ho_cmd,out,ho_acc,ho_fail) =
     m(c).( [c=data]m(v).'out<v>.MS(m,data,ho_cmd,out,ho_acc,ho_fail)
           +[c=ho_cmd]m(mnew).
               ('mnew<ho_acc>.MS(mnew,data,ho_cmd,out,ho_acc,ho_fail)
               +'m<ho_fail>.MS(m,data,ho_cmd,out,ho_acc,ho_fail)))

agent P(fa,fp,in,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail) =
     (^m)( MSC(fa,fp,m,in,data,ho_cmd,ho_com,ho_fail,ch_rel)
         | BSp(fp,m,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail))

agent Q(fa,out,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail) =
     (^m)( BSa(fa,m,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail)
         | MS(m,data,ho_cmd,out,ho_acc,ho_fail))

agent System(in,out,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail) =
     (^fa)(^fp)( P(fa,fp,in,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail)
               | Q(fa,out,ho_acc,ho_com,data,ho_cmd,ch_rel,ho_fail))
```

Figure 7.5: The MWB code for the protocol specification.

```
agent Spec(in,out) = in(v).S1(v,in,out) + t.Spec(in,out)
agent S1(v1,in,out) =
   in(v).S2(v1,v,in,out) + 'out<v1>.Spec(in,out)
  + t.'out<v1>.Spec(in,out)
agent S2(v1,v2,in,out) =
   in(v).S3(v1,v2,v,in,out) + 'out<v1>.S1(v2,in,out)
  + t.'out<v1>.'out<v2>.Spec(in,out)
agent S3(v1,v2,v3,in,out) = 'out<v1>.S2(v2,v3,in,out)
```

Figure 7.6: The MWB code for the service specification.

behaviour of the system. In Figures 7.4 and 7.6, we present the service specification of the handover protocol and its rendering into MWB code, respectively.

When checking the protocol specification (`System`) against the more abstract service specification (`Spec`), we must express the fact that the parameters `in`, `out`, `ho_acc`, `ho_com`, `data`, `ho_cmd`, `ho_fail` and `ch_rel` are constants, i.e. they are distinct from all other free names. This is done by using the `weqd` (weak open bisimulation with distinctions) command of the MWB, in the following way:

```
weqd (i,o,a,co,d,cm,f,r) Spec(i,o) System(i,o,a,co,d,cm,f,r)
```

The bisimulation found by the MWB has 323 tuples. Running on a Sun UltraSPARC 1/167, it took less than 13 CPU minutes (user mode). The ML heap size was at most 53 Mb. Because of the use of hash tables, the locality of references is very poor, which means that unless enough physical memory is available, a great deal of real time will be spent paging. The decrease in memory usage of the new algorithm, based on the symbolic open bisimulation of Section 4.4.1, is therefore quite significant since it allows the verification of larger examples on smaller machines than previously possible. The elapsed time for this verification is slightly greater than in the previous version, but we argue that the time used for paging would dominate this runtime if the space necessary exceeds the available memory.

Chapter 8

Conclusion and Discussion

We have presented a broad foundational theory of the fusion calculus. We defined its labelled and unlabelled operational semantics, and treated strong and weak bisimulation equivalences for both semantics in some detail, also giving complete axiom systems. The equivalences were characterized symbolically, leading to algorithms for equivalence checking implemented in an automatic tool. We also applied the expressive power of the fusion calculus to give simple encodings of foundational calculi for functional and concurrent constraint programming.

The fusion calculus cleans up the π-calculus by reducing the number of binding operators and the number of bisimulation equivalences, and by making input and output symmetric like in pure CCS. We attain a calculus where concepts from other models of computation are more easily expressed than in the π-calculus, thereby taking a step towards a unified yet *simple* model of computation. Another significant simplification of the π-calculus, the πI-calculus, admittedly has even simpler theory than fusion, but expressing, e.g., the delayed input can only be done *indirectly* through encodings. The fusion calculus is therefore better suited for relating mobile processes to other paradigms.

There are of course gaps in the presented work needing further investigation. One is the study of the weak barbed congruence in the absence of mismatch and match operators. Without mismatch the context used in the proof of Lemma 4.20 is not applicable. An interesting question is if we can still characterize hyperequivalence by barbed congruence, and if not, what a coinductive definition of the induced congruence would look like. If both match and mismatch are dropped from the calculus, it seems clear that the congruence changes. The following is an example due to Boreale and Sangiorgi [BS98] of the effect of dropping the match operator in the π-calculus:

$$\overline{u}\,v\,.\,(!\,\overline{v} \mid !\,v \mid \overline{w} \mid !\,w) \sim \overline{u}\,w\,.\,(!\,\overline{v} \mid !\,v \mid \overline{w} \mid !\,w)$$

An agent performing an input $u(x)$ in this context can not distinguish between the cases where it receives v or w for x. This example is most likely applicable also to the fusion calculus.

The correspondence between a ρ-calculus term and its encoding (Theorem 6.5) can probably be strengthened to imply that two terms are equivalent iff their encodings are equivalent, in a similar way to what is done in Theorem 6.9 for the γ-calculus encoding. The question is which equivalence would be relevant in such a result. Convergence is of prime importance in the ρ-calculus and our equivalences do not currently take that into account. Refining the equivalences in that respect is probably desirable.

The question whether more complex constraint systems can be handled in the same way is largely open. Clearly the fusion calculus is geared particularly towards constraints on the identity of names. But it may be that unification can be represented cleanly, implying that ordinary logic programming has a good encoding.

A variant of the labelled transition semantics can be given which would simplify the definition of hyperequivalence and perhaps some proofs. The idea is to let the substitutive effect of actions take place immediately, without the need of a scope operator. The transition rule would be

$$\frac{P \xrightarrow{\gamma} P'}{P \overset{\gamma}{\leadsto} P'\sigma_\gamma}$$

where σ_γ is a substitutive effect of γ. Using $\overset{\gamma}{\leadsto}$ transitions, basic bisimilarity (Definition 5) would coincide with hyperequivalence.

A graphical representation of the fusion calculus has been developed by Parrow, based on his Interaction Diagrams for the π-calculus [Par95]. The interaction diagrams are significantly simplified, and an interactive simulation tool is being developed based on this. This provides animation of fusions and a highly experimental tool for structured design of diagrams.

To model full λ-calculus reduction, we will look at extensions to *multiway* communication, where more than two processes synchronize in an interaction. This would allow transitions like the following:

$$u\,x\,.\,P \mid \overline{u}\,y\,.\,Q \mid u\,z\,.\,R \xrightarrow{\{x=y=z\}} P \mid Q \mid R$$

This could be used to encode a form of reduction under replication: if $P \xrightarrow{\alpha} P'$ then $!\,P \xrightarrow{\alpha} !\,P'$, which would make it possible to encode the λ-calculus rule for reduction in the RHS of an application term:

$$MN \;\rightarrow_\lambda\; MN' \qquad \text{if } N \rightarrow_\lambda N'$$

This rule allows N to reduce before being copied by a β-reduction; in the encoding, we must let all copies of N (encoded by replication) reduce synchronously, before being copied when accessed by the encoding of M as a result of β-reduction. Multiway extensions are probably simpler in the fusion calculus than in the π-calculus where the distinction between input and output is a complication.

Investigating *asynchronous* versions of the fusion calculus, where communication is not by handshaking, should be interesting: first from the viewpoint of the calculi for concurrent constraints encoded in this thesis, which are in a sense inherently asynchronous in their nature; the "application" terms of ρ- and γ-calculi are asynchronous, as are often the *tell* operations of the concurrent constraint framework. Second, and perhaps more interesting, a *symmetrically asynchronous* fusion calculus can be defined, where both input and output are asynchronous and the only prefixes are of the type $\alpha\,.\,\mathbf{0}$. Such a variant would not make sense in the π-calculus: since input is always binding, and the scope of these bindings would be null, it would make the objects completely insignificant, and very little expressiveness would be left. In the fusion calculus we conjecture that such a subcalculus might have considerable expressiveness, if input prefix could be encoded by clever use of extra objects and scope, similar to the way Honda and Tokoro [HT91] encode output prefix in the asynchronous π-calculus. The work of Honda mentioned in Section 3.5 should be of interest when investigating this idea, as there is no synchronization in his calculus.

Action calculi [Mil93] is a framework used to study interactive systems at low level, breaking up actions in their "molecules", making it possible to classify and interrelate models arising from their combinations. This framework has successfully been used to describe various models of interactive system such as Petri Nets, λ-calculus and the π-calculus, and by applying it to the fusion calculus, it can likely provide useful insight in how they relate.

Two areas which have not been addressed in the thesis are *type systems* and *modal logics*. Type systems (often synonymously called sorting systems) of varying degrees of sophistication have been used for the π-calculus. Milner [Mil91b, Mil91a] defined a system where each name is given a sort describing what kind of names can be passed over it. For example, the name u in $u(xy)\,.\,P$ would have a sort which is the sequence of the sorts of x and y, such as $\textsc{usort} \mapsto (\textsc{xsort},\textsc{ysort})$. With this simple system agents can, e.g., be analysed to see if they are well-behaved in the sense that input and output actions with the same subject always have the same number of objects. For example, the encoding of the λ-calculus in Section 6.1 can be shown to respect the following sorting:

$$\textsc{args} \mapsto (\textsc{var},\textsc{args}), \qquad \textsc{var} \mapsto (\textsc{args})$$

where the sort of u in $[\![E]\!]u$ would be $\textsc{args}$: u always carries two names, the first of which can carry a name of type $\textsc{args}$, and the second of which also carries two names in the same way as u.

More advanced type systems have been studied in, e.g., [VH93, Vas94, Tur95, LW95, Yos96, AD96, PS96a, San96a, PS97, San97, Sew98], where polymorphic types, input-output capability typing, subtyping, etc., are treated. A very interesting application of type systems is to let the definition of equivalence take types into account. This strengthens the equivalence and simplifies the proofs.

In the fusion calculus the use of types to differentiate between uses of names could be an elegant way of distinguishing between normal names and "constant" names, which cannot be fused with other constants, or "π" names, which are used as in the f_π subcalculus. These types are useful when considering a fusion calculus with other values than names: if, e.g., we consider a higher-order fusion calculus, we would probably like to avoid fusing processes (unless we can find a clear semantics of such an operation), but allowing "reception" of a process into a "π" name; on the other hand we may still allow fusion of first-order names.

Modal logics together with model checking algorithms can be very powerful tools for analysing properties of agents. Suppose we want to verify a property such as

> "after inserting 5 SEK in the slot, pressing the button will always deliver a cup of coffee"

of a vending machine specification. With only equivalence checking available, we would have to specify the behaviour of all other types of interaction and see if the complete system is equivalent to the specification; with modal logics such as the modal μ-calculus [Koz83, HM85], the property can be concisely formulated as a formula such as

$$[\mathit{5SEK}][\mathit{button}]\langle\overline{\mathit{coffee}}\rangle\,\mathit{True}$$

By *model checking* we can verify that the specification is in the model of the logic formula, i.e., that the specification satisfies the formula.

Previous work on modal logics and model checking for the π-calculus, such as [MPW93, Dam96], should be useful as a basis for developing fusion calculus logics. We would expect simplifications due to the single binder and symmetric input and output of the fusion calculus. Efficient model checking algorithms based on, e.g., [Fra96, Bes98], should then be designed and implemented in the Mobility Workbench.

In conclusion, with this thesis we have demonstrated that the fusion calculus simplifies and extends known work, but perhaps more importantly we have laid the grounds for many exciting and rewarding lines of research.

References

[Abr89] S. Abramsky. The lazy lambda calculus. In D. Turner, ed, *Research Topics in Functional Programming.* Addison-Wesley, 1989.

[ACS96] R. M. Amadio, I. Castellani and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer-Verlag, 1996. Full version to appear in *Theoretical Computer Science.*

[AD96] R. M. Amadio and M. Dam. Toward a modal theory of types for the π-calculus. In B. Jonsson, ed, *Proceedings of FTRTFT'96*, volume 1135 of *Lecture Notes in Computer Science.* Springer-Verlag, 1996.

[Agh86] G. Agha. *Actors: a Model of Concurrent Computation in Distributed Sysems.* MIT Press, 1986.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[Bes98] F. B. Beste. The model prover - a sequent-calculus based modal μ-calculus model checker tool for finite control π-calculus agents. M.Sc. thesis DoCS 98/97, Department of Computer Systems, Uppsala University, Sweden, Mar. 1998.

[Bor96] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, 1996.

[Bou89] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. In J. Díaz and F. Orejas, eds, *Proceedings of TAPSOFT '89, Volume 1*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer-Verlag, 1989.

[Bou92] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sofia-Antipolis, May 1992.

[Bou97] G. Boudol. The π-calculus in direct style. In *Proceedings of POPL '97*, pages 228–241. ACM, Jan. 1997.

[BS98] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proceedings of LICS'98*, 1998. To appear.

[BW90] J. Baeten and W. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[CGR93] D. Craigen, S. Gerhard and T. Ralston. An international survey of industrial applications of formal methods. US Department of Commerce, NISTGCR 93/626 1993, 1993. p ii.

[Chu41] A. Church. The calculi of lambda-conversion. *Annals of Mathematics Studies*, 6, 1941. Princeton University Press, Princeton, N.J.

[Dam96] M. Dam. Model checking mobile processes. *Journal of Information and Computation*, 129(1):35–51, 1996. Also available as Research Report R94:01, SICS, Sweden. A summary appeared in the *Proceedings of CONCUR '93*, LNCS 715.

[dB72] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[dBP91] F. de Boer and C. Palamidessi. A fully abstract model for concurrent constraint programming. In S. Abramsky and T. Maibaum, eds, *Proceedings of TAPSOFT/CAAP*, volume 493 of *Lecture Notes in Computer Science*, pages 296–319. Springer-Verlag, 1991.

[EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label-passing. Technical Report DAIMI PB-208, Comp. Sc. Department, Univ. of Aarhus, Denmark, 1986.

[Fer89] J.-C. Fernandez. Aldébaran: A tool for verification of communicating processes. Technical Report RTC 14, IMAG, Laboratoire de Génie Informatique, 1989.

[FFG+97] G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. An automata based verification environment for mobile processes. In E. Brinksma, ed, *Proceedings of TACAS '97*, volume 1217 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, Apr. 1997.

[FG96] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.

[FGM+98] G. Ferrari, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. Verifying mobile processes in the HAL environment. In *Proceedings of CAV'98*, 1998. To appear. Available from `ftp://ftp.di.unipi.it/pub/Papers/pistore/cav98.ps.gz`.

[FMQ95] G. Ferrari, G. Modoni and P. Quaglia. Towards a semantic-based verification environment for the pi-calculus. In *Proceedings of 5th Italian Conference on Theoretical Computer Science: ICTCS-95*. World Scientific, 1995.

[FO92] L. Fredlund and F. Orava. Modelling dynamic communication structures in LOTOS. In K. R. Parker and G. A. Rose, eds, *Proceedings of FORTE '91*, pages 185–200. North-Holland, 1992.

[Fra96] T. Franzén. A theorem-proving approach to deciding properties of finite-control agents (extended abstract). Technical Report T96:02, Swedish Institute of Computer Science, Kista, 1996.

[Fu97] Y. Fu. A proof-theoretical approach to communication. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, eds, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 325–335. Springer-Verlag, 1997.

[Gay93] S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proceedings of POPL '93*. ACM, 1993.

[GJS96] V. Gupta, R. Jagadeesan and V. A. Saraswat. Truly concurrent constraint programming. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[GS90] H. Garavel and J. Sifakis. Compilation and verification of LOTOS specifications. In L. Logrippo, R. Probert and H. Ural, eds, *Proceedings of PSTV X*. North-Holland, 1990.

[Hew77] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artifical Intelligence*, 8(3):323–364, 1977.

[HL95] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995. Earlier version published as Technical Report 1/92, School of Cognitive and Computing Sciences, University of Sussex, UK.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[Hoa85] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hol91] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991. Available as `http://cm.bell-labs.com/cm/cs/who/gerard/popd.html`.

[Hon97] K. Honda. Process structures, Mar. 1997. Submitted for publication.

[HSW93] M. Henz, G. Smolka and J. Würtz. Oz—a programming language for multi-agent systems. In R. Bajcsy, ed, *Proceedings of IJCAI '93 (13th International Joint Conference on Artificial Intelligence)*, volume 1, pages 404–409, Chambéry, France, 30 August–3 September 1993. Morgan Kaufmann Publishers.

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, ed, *Proceedings of ECOOP '91*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147, Geneva, Switzerland, July 1991. Springer-Verlag.

[JP93] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Journal of Information and Computation*, 107:272–302, 1993.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KS90] P. Kanellakis and S. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Journal of Information and Computation*, 86(1):43–68, May 1990.

[Li94] B. Li. A π-calculus specification of Prolog. In D. Sannella, ed, *Proceedings of ESOP '94*, volume 788 of *Lecture Notes in Computer Science*, pages 379–393, Edinburgh, 1994. Springer-Verlag.

[Lin94] H. Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical Report 7/94, School of Cognitive and Computing Sciences, University of Sussex, UK, 1994.

[Lin95a] H. Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. In P. D. Mosses, M. Nielsen and M. I. Schwarzbach, eds, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, 1995. Presented in the CAAP-section. Available as Technical Report ISCAS-LCS-94-11, Institute of Software, Chinese Academy of Sciences, 1994.

[Lin95b] H. Lin. Unique fixpoint induction for mobile processes. In I. Lee and S. A. Smolka, eds, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 88–102. Springer-Verlag, 1995.

[LW95] X. Liu and D. Walker. A polymorphic type system for the polyadic π-calculus. In I. Lee and S. A. Smolka, eds, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 103–116. Springer-Verlag, 1995.

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

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:269–310, 1983.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil91a] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, Oct. 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[Mil91b] R. Milner. Sorts in the π-calculus (extended abstract). In E. Best and G. Rozenberg, eds, *Proceedings of the 3rd Workshop on Concurrency and Compositionality*, volume 191 of *GMD-Studien*. GMD Bonn, St. Augustin, 1991. Also available as Report 6/91 from University of Hildesheim.

[Mil92] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil93] R. Milner. Action calculi, or concrete action structures. In A. M. Borzyszkowski and S. Sokołowski, eds, *Proceedings of MFCS '93*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer-Verlag, Sept. 1993. Also titled *Action Calculi I: Axioms and Applications*.

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(2):95–132, Apr. 1974. Also Technical Report 71-1, Dept. of Computer Science, Carnegie Mellon University, January 1971.

[MPW92] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.

[MPW93] R. Milner, J. Parrow and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[MR95] U. Montanari and F. Rossi. A concurrent semantics for concurrent constraint programming via contextual nets. In V. Saraswat and P. V. Hentenryck, eds, *Principles and Practice of Constraint Programming: The Newport Papers*. MIT Press, Cambridge, MA, 1995.

[MS92a] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, ed, *Proceedings of ICALP '92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[MS92b] R. Milner and D. Sangiorgi. The problem of "weak bisimulation up-to". In R. Cleaveland, ed, *Proceedings of CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992. Revised version available as ftp://ftp.dcs.ed.ac.uk/pub/sad/x.ps.Z, entitled *Techniques of "weak bisimulation up-to"*.

[MS98] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proceedings of ICALP '98*, 1998. To appear.

[NM95] J. Niehren and M. Müller. Constraints for free in concurrent computation. In K. Kanchanasut and J.-J. Lévy, eds, *Asian Computer Science Conference*, volume 1023 of *Lecture Notes in Computer Science*, pages 171–186, Pathumthani, Thailand, 11–13 Dec. 1995. Springer-Verlag.

[NP96] U. Nestmann and B. C. Pierce. Decoding choice encodings. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 1996. Revised full version as report ERCIM-10/97-R051, European Research Consortium for Informatics and Mathematics, 1997.

[OP92] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4:497–543, 1992.

[Pal97] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of POPL '97*, pages 256–265. ACM, Jan. 1997.

[Par81] D. M. Park. Concurrency on automata and infinite sequences. In P. Deussen, ed, *Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[Par95] J. Parrow. Interaction diagrams. *Nordic Journal of Computing*, 1995.

[PS95] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Journal of Information and Computation*, 120(2):174–197, 1995.

[PS96a] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[PS96b] M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In R. Alur, ed, *Proceedings of CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, 1996. Full version to appear in *Information and Computation.*

[PS97] B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *Proceedings of POPL '97*, pages 241–255. ACM, Jan. 1997. Full version available as INRIA-Sophia Antipolis Rapport de Recherche No. 3042 and as Indiana University Computer Science Technical Report 468.

[PT87] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[PT98] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling and M. Tofte, eds, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1998. To appear.

[PV97] J. Parrow and B. Victor. The update calculus. In M. Johnson, ed, *Proceedings of AMAST'97*, volume 1349 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, Dec. 1997. Full version available as Technical report DoCS 97/93, Uppsala University.

[PV98a] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS '98*, 1998. To appear. Available from `http://www.docs.uu.se/victor/tr/fusion.html`.

[PV98b] J. Parrow and B. Victor. The tau-laws of fusion. Submitted for publication. Available from `http://www.docs.uu.se/~victor/tr/taufusion.html`, Mar. 1998.

[Ros93] B. Ross. A π-calculus semantics of logical variables and unification. In S. Purushothaman and A. Zwarico, eds, *Proceedings of First North American Process Algebra Workshop, Stony Brook, 1992*, Workshops in Computing, pages 216–230. Springer-Verlag, 1993.

[San93] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1993.

[San94] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Journal of Information and Computation*, 111(1):120–131, 1994. An extended abstract appeared in Proc. of LICS'92.

[San95a] D. Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche RR-2515, INRIA Sophia-Antipolis, 1995.

[San95b] D. Sangiorgi. On the proof method for bisimulation (extended abstract). In J. Wiederman and P. Hájek, eds, *Proceedings of MFCS '95*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer-Verlag, 1995. To appear in *Mathematical Structures in Computer Science*. Full version available electronically.

[San96a] D. Sangiorgi. An interpretation of typed objects into typed π-calculus. Rapport de Recherche RR-3000, INRIA Sophia-Antipolis, Aug. 1996. Accepted for publication in *Information and Computation*.

[San96b] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(1,2):235–274, 1996. Also as Rapport de Recherche RR-2539, INRIA Sophia-Antipolis, 1995. Extracts of parts of the material contained in this paper can be found in the Proceedings of TAPSOFT'95 and ICALP'95.

[San96c] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996. Earlier version published as Report ECS-LFCS-93-270, University of Edinburgh. An extended abstract appeared in the *Proceedings of CONCUR '93*, LNCS 715.

[San97] D. Sangiorgi. The typed π-calculus at work: A proof of Jones's parallelisation theorem on concurrent objects. In B. C. Pierce, ed, *Proceedings of FOOL 4*, page 12. Free electronic publication, Jan. 1997. To appear in *Theory and Practice of Object-Oriented Systems*.

[Sar93] V. A. Saraswat. *Concurrent Constraint Programming*. ACM Doctoral Dissertation Award and Logic Programming Series. MIT Press, Cambridge, MA, 1993.

[Sew98] P. Sewell. Global/local subtyping and capability inference for a distributed pi-calculus. In *Proceedings of ICALP '98*, 1998. To appear. Full version as Technical Report 435, Computer Laboratory, University of Cambridge.

[Smo94] G. Smolka. A foundation for higher-order concurrent constraint programming. In J.-P. Jouannaud, ed, *Proceedings of Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 50–72. Springer-Verlag, Sept. 1994. Available as Research Report RR-94-16 from DFKI Kaiserslautern.

[Smo95a] G. Smolka. The definition of Kernel Oz. In A. Podelski, ed, *Constraints: Basics and Trends*, volume 910 of *LNCS*, pages 251–292. Springer, 1995.

[Smo95b] G. Smolka. The Oz programming model. In J. van Leeuwen, ed, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 324–343. Springer-Verlag, Berlin, 1995.

[SRP91] V. A. Saraswat, M. Rinard and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of POPL '91*, pages 333–352. ACM, 1991.

[Tho90] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Imperial College, University of London, Sept. 1990.

[Tur36] A. M. Turing. On computable numbers with an application to the Entscheidungs-problem. *Proc. Longdon Math. Soc.*, 2(42):230–265, 1936. A correction, *ibid.*, **43**, pp. 544–546.

[Tur95] D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1995.

[Vas94] V. T. Vasconcelos. Predicative polymorphism in π-calculus. In C. Halatsis, D. Maritsas, G. Philokyprou and S. Theodoridis, eds, *Proceedings of PARLE '94*, volume 817 of *Lecture Notes in Computer Science*, pages 425–437. Springer-Verlag, July 1994.

[vEVD89] P. H. J. van Eijk, C. A. Vissers and M. Diaz. *The Formal Description Technique LOTOS*. North-Holland, 1989.

[VH93] V. T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In E. Best, ed, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 524–538. Springer-Verlag, 1993.

[Vic94] B. Victor. *A Verification Tool for the Polyadic π-Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.

[Vic98] B. Victor. Symbolic characterizations and algorithms for hyperequivalence and open bisimulation. Submitted for publication. Available from `http://www.docs.uu.se/~victor/tr/symhyper.html`, Mar. 1998.

[VM94] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, ed, *Proceedings of CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[VP96] B. Victor and J. Parrow. Constraints as processes. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 389–405. Springer-Verlag, 1996.

[VP98] B. Victor and J. Parrow. Concurrent constraints in the fusion calculus. In *Proceedings of ICALP '98*, 1998. To appear. Available from `http://www.docs.uu.se/~victor/tr/ccfc.html`.

[Wal95] D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.

[WK48] F. C. Williams and T. Kilburn. Electronic digital computers. *Nature*, 162:487, Sept. 1948. Letter to Nature. Available at `http://www.computer50.org/kgill/mark1/natletter.html`.

[Yos96] N. Yoshida. Graph types for monadic mobile processes. In V. Chandru and V. Vinay, eds, *Proceedings of FSTTCS '96*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 1996. Full version as Technical Report ECS-LFCS-96-350, University of Edinburgh.

Bibliography on Calculi for Mobile Processes

This bibliography was current at the printing of this thesis on April 28, 1998.

[1] M. Abadi, C. Fournet and G. Gonthier. Secure implementation of channel abstractions. In *Proceedings of LICS'98*, 1998. To appear.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security, Zürich, April 1997*. ACM Press, 1997. Long version as Technical Report 414, University of Cambridge.

[3] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of CONCUR '97*, volume 1243 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1997.

[4] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. In C. Hankin, editor, *Proceedings of ESOP '98*, volume 1381 of *Lecture Notes in Computer Science*, pages 12–26. Springer-Verlag, 1998.

[5] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135:5–9, 1994. An Introduction to [21].

[6] O. Aït-Mohamed. Mechanizing a pi-calculus equivalence in HOL. In E. T. Schubert, P. J. Windley and J. Alves Foss, editors, *Proceedings of HOL '95*, volume 971 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1995. See also Rapport de Recherche RR-2412, INRIA-Lorraine, 1994.

[7] O. Aït-Mohamed. *The theory of the pi-calculus in HOL*. PhD thesis, Henri Poincaré University Nancy I, France, July 1996.

[8] R. M. Amadio. A uniform presentation of CHOCS and π-calculus. Rapport de Recherche 1726, INRIA-Lorraine, Nancy, 1992.

[9] R. M. Amadio. On the reduction of CHOCS bisimulation to π-calculus bisimulation. In E. Best, editor, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 112–126. Springer-Verlag, 1993. Extended version as Rapport de Recherche, INRIA-Lorraine, 1993.

[10] R. M. Amadio. Translating Core Facile. Technical Report ECRC-94-3, European Computer-Industry Research Centre, München, Feb. 1994.

[11] R. M. Amadio. An asynchronous model of locality, failure, and process mobility. In *To appear in the Proceedings of COORDINATION '97*, 1997. Full version as Rapport Interne, LIM Marseille, and Rapport de Recherche RR-3109, INRIA Sophia-Antipolis, 1997.

[12] R. M. Amadio and O. Aït-Mohamed. An analysis of π-calculus bisimulation. Technical Report ECRC-94-2, European Computer-Industry Research Centre, Munich, Jan. 1992.

[13] R. M. Amadio, I. Castellani and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer-Verlag, 1996. Full version to appear in *Theoretical Computer Science*.

[14] R. M. Amadio and M. Dam. Reasoning about higher-order processes. In P. D. Mosses, M. Nielsen and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 202–216. Springer-Verlag, 1995. Full version as SICS Research Report RR:94/18, October 1994.

[15] R. M. Amadio and M. Dam. Toward a modal theory of types for the π-calculus. In B. Jonsson, editor, *Proceedings of FTRTFT'96*, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[16] R. M. Amadio, L. Leth and B. Thomsen. From a concurrent λ-calculus to the π-calculus. In H. Reichel, editor, *Proceedings of FCT '95*, volume 965 of *Lecture Notes in Computer Science*, pages 106–115. Springer-Verlag, 1995. Full version as Technical Report ECRC-95-18.

[17] R. M. Amadio and S. Prasad. Localities and failures. In P. S. Thiagarajan, editor, *Proceedings of FSTTCS '94*, volume 880 of *Lecture Notes in Computer Science*, pages 205–216. Springer-Verlag, 1994. Full version as Technical Report 94-18, ECRC Munich, 1994.

[18] S. J. Ambler. A de Bruijn notation for the π-calculus. Technical Report 569, Dept. of Computer Science, Queen Mary and Westfield College, London, May 1991.

[19] E. Astesiano and E. Zucca. Parametric channels via label expressions in CCS. *Theoretical Computer Science*, 33:45–64, 1984.

[20] R. Banach, J. Balász and G. Papadoupoulos. A translation of the pi-calculus into MONSTR. *Journal of Universal Computer Science*, 1(6):339–398, 1995.

[21] G. Bellin and P. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135:11–65, 1994. Also published as LFCS report ECS-LFCS-92-232, Laboratory for Foundations of Computer Science, University of Edinburgh, UK.

[22] J. Bergstra and G. Stefanescu. Network algebra for synchronous and asynchronous dataflow. Logic Group Preprint Series 122, Department of Philosophy, Utrecht University, The Netherlands, Oct. 1994.

[23] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[24] A. Bianchi, S. Coluccini, P. Degano and C. Priami. An efficient verifier of truly concurrent properties. In V. Malyshkin, editor, *Proceedings of PaCT '95*, volume 964 of *Lecture Notes in Computer Science*, pages 36–50. Springer-Verlag, 1995.

[25] C. Bodei, P. Degano and C. Priami. Mobile processes with a distributed envirionent. In F. M. auf der Heide and B. Monien, editors, *Proceedings of ICALP '96*, volume 1099 of *Lecture Notes in Computer Science*, pages 490–501. Springer-Verlag, 1996. Full version as Università di Pisa Technical Report *Handling Locally Names of Mobile Agents*, 1996.

[26] M. Boreale. *Process Algebraic Theories for Mobile Systems.* PhD thesis, Università degli Studi di Roma "La Sapienza", 1995. Available as PhD VII-95-2.

[27] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, 1996.

[28] M. Boreale and R. De Nicola. Testing equivalences for mobile processes. *Journal of Information and Computation*, 120:279–303, 1995. Available as Report SI 92 RR 04, Università "La Sapienza" di Roma; an extended abstract appeared in the Proceedings of CONCUR'92, LNCS 630.

[29] M. Boreale and R. De Nicola. A symbolic semantics for the π-calculus. *Journal of Information and Computation*, 126(1):34–52, 1996. Available as Report SI 94 RR 04, Università "La Sapienza" di Roma; an extended abstract appeared in the Proceedings of CONCUR'94, pages 299–314, LNCS 836.

[30] M. Boreale, R. De Nicola and R. Pugliese. Asynchronous observations of processes. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 95–109. Springer-Verlag, 1998.

[31] M. Boreale, C. Fournet and C. Laneve. Bisimulations for the join-calculus. In *Proceedings of PROCOMET '98*, 1998. To appear.

[32] M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the pi-calculus. In E. W. Mayr and C. Puech, editors, *Proceedings of STACS'95*, volume 900 of *Lecture Notes in Computer Science*, pages 243–254. Springer-Verlag, 1995. Full version in report ECS–LFCS–94–297, University of Edinburgh. Revised version to appear in *Acta Informatica*.

[33] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proceedings of LICS'98*, 1998. To appear.

[34] R. Borgia, P. Degano, C. Priami, L. Leth and B. Thomsen. Understanding mobile agents via a non interleaving semantics for Facile. In R. Cousot and D. Schmidt, editors, *Proceedings of SAS '96*, volume 1145 of *Lecture Notes in Computer Science*, pages 98–112. Springer-Verlag, 1996. Extended version as Technical Report ECRC-96-4, 1996.

[35] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. In J. Díaz and F. Orejas, editors, *Proceedings of TAPSOFT '89, Volume 1*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer-Verlag, 1989.

[36] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sofia-Antipolis, May 1992.

[37] G. Boudol. Some chemical abstract machines. In J. W. de Bakker, W.-P. de Roever and G. Rozenberg, editors, *A Decade of Concurrency 1993 (REX Workshop)*, volume 803 of *Lecture Notes in Computer Science*, pages 92–123. Springer-Verlag, 1994.

[38] G. Boudol. The π-calculus in direct style. In *Proceedings of POPL '97*, pages 228–241. ACM, Jan. 1997.

[39] G. Boudol. Typing the use of resources in a concurrent calculus. In *Proceedings of ASIAN '97*, volume 1345 of *Lecture Notes in Computer Science*, pages 239–253. Springer-Verlag, 1997.

[40] G. Boudol and C. Laneve. λ-calculus, multiplicites and the π-calculus. In G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1998. To appear.

[41] S. Brock and G. Ostheimer. Process semantics of graph reduction. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 471–485. Springer-Verlag, 1995.

[42] N. Busi and R. Gorrieri. A Petri Net semantics for π-calculus. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 145–159. Springer-Verlag, 1995.

[43] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 1998.

[44] G. L. Cattani, I. Stark and G. Winskel. Presheaf models for the pi-calculus. In E. Moggi and G. Rosolini, editors, *Proceedings of CTCS '97*, volume 1290 of *Lecture Notes in Computer Science*, pages 106–126. Springer-Verlag, 1997.

[45] C. Crasemann. *$\pi\lambda$-Kalküle für Prozesse und Funktionen.* PhD thesis, Christian-Albrechts-Universität Kiel, Feb. 1992. Insitut für Informatik und Praktische Mathematik, Bericht Nr. 9302 (in German).

[46] S. Dal-Zilio. *Modèles du π-calcul. Structures d'Actions et Relations avec l'Équivalence Par Permutations.* Mémoire de D.E.A., École Normale Superieure de Lyon, INRIA Sophia-Antipolis, 1994. (in French).

[47] S. Dal-Zilio. Implicit polymorphic type system for the blue calculus. Rapport de Recherche RR-3244, INRIA Sophia-Antipolis, Sept. 1997.

[48] M. Dam. Model checking mobile processes. *Journal of Information and Computation*, 129(1):35–51, 1996. Also available as Research Report R94:01, SICS, Sweden. A summary appeared in the *Proceedings of CONCUR '93*, LNCS 715.

[49] M. Dam. On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 183:215–228, 1997. Previously published as SICS Research Report R94:20, 1994.

[50] M. Dam. Proving trust in systems of second-order processes. In *Proceedings of HICSS 31*, volume VII, pages 255–264. IEEE Computer Science, 1998. Full version available electronically at `ftp://ftp.sics.se/pub/fdt/mfd/ptssop.ps.Z`.

[51] P. Degano, J.-V. Loddo and C. Priami. Mobile processes with local clocks. In M. Dam, editor, *Proceedings of the Fifth LOMAPS Workshop*, volume 1192 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[52] P. Degano and C. Priami. Causality for mobile processes. In Z. Fülöp and F. Gécseg, editors, *Proceedings of ICALP '95*, volume 944 of *Lecture Notes in Computer Science*, pages 660–671. Springer-Verlag, 1995.

[53] P. Degano and C. Priami. Enhanced operational semantics. *ACM Computing Surveys*, 28(2):352–354, 1996.

[54] P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, 1997. To appear.

[55] P. Degano, C. Priami, L. Leth and B. Thomsen. Analysis of Facile programs: a case study. In M. Dam, editor, *Proceedings of the Fifth LOMAPS Workshop*, volume 1192 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[56] S. Eisenbach and R. Patterson. π-calculus semantics for the concurrent configuration language Darwin. In *Proceedings of the 26th Annual Hawaii International Conference on System Sciences*, volume 2. IEEE Computer Society Press, 1993.

[57] U. Engberg and M. Nielsen. A calculus of communicating systems with label-passing. Technical Report DAIMI PB-208, Comp. Sc. Department, Univ. of Aarhus, Denmark, 1986.

[58] J. Engelfriet. A multiset semantics for the π-calculus with replication. *Theoretical Computer Science*, 153(1–2):65–94, 1996. Also published in LNCS 715 (Proc. CONCUR'93) and as Leiden University Report 94-26, August 1994.

[59] J. Engelfriet and T. Gelsema. Multisets and structural congruence of the π-calculus with replication. Report 2/95, Leiden University, 1996.

[60] L.-H. Eriksson and F. Orava. Analysing a π-calculus specification of a cache-coherence protocol. Technical Report SICS RR95-04, Swedish Institute of Computer Science, 1995.

[61] G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. An automata based verification environment for mobile processes. In E. Brinksma, editor, *Proceedings of TACAS '97*, volume 1217 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, Apr. 1997.

[62] G. Ferrari, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. Verifying mobile processes in the HAL environment. In *Proceedings of CAV'98*, 1998. To appear. Available from `ftp://ftp.di.unipi.it/pub/Papers/pistore/cav98.ps.gz`.

[63] G. Ferrari and U. Montanari. A tile-based coordination view of asynchronous π-calculus. In *Proceedings of MFCS'97*. Springer-Verlag, 1997. To appear.

[64] G. Ferrari, U. Montanari and P. Quaglia. A π-calculus with explicit substitutions: the late semantics. In I. Privara, B. Rovan and P. Ruzicka, editors, *Proceedings of MFCS '94*, volume 841 of *Lecture Notes in Computer Science*, pages 342–351. Springer-Verlag, Aug. 1994. Full version as Technical Report, Università di Pisa.

[65] G. Ferrari, U. Montanari and P. Quaglia. The weak late π-calculus semantics as observation equivalence. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 57–71. Springer-Verlag, 1995.

[66] G. Ferrari, U. Montanari and P. Quaglia. A π-calculus with explicit substitutions. *Theoretical Computer Science*, 168(1):53–103, 1996.

[67] M. Fiore, E. Moggi and D. Sangiorgi. A fully-abstract model for the π-calculus (extended abstract). In *Proceedings of LICS '96*, pages 43–54. IEEE, Computer Society Press, July 1996.

[68] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.

[69] C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proceedings of ICALP '98*, 1998. To appear.

[70] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget and D. Rémy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer-Verlag, 1996.

[71] C. Fournet, C. Laneve, L. Maranget and D. Rémy. Implicit typing à la ML for the join-calculus. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of CONCUR '97*, volume 1243 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, 1997.

[72] Y. Fu. A proof-theoretical approach to communication. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 325–335. Springer-Verlag, 1997.

[73] S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proceedings of POPL '93*. ACM, 1993.

[74] J. Glauert. Asynchronous mobile processes and graph rewriting. In D. Etiemble and J.-C. Syre, editors, *Proceedings of PARLE '92*, volume 605 of *Lecture Notes in Computer Science*, pages 63–80. Springer-Verlag, 1992.

[75] J. Glauert. Parallel implementation of functional languages using small processes. In *Parallel Implementation Workshop.* RWTH Aachen, 1992.

[76] J. Glauert, L. Leth and B. Thomsen. A new process model for functions. In *Term Graph Rewriting: Theory and Practice.* John Wiley & Sons Ltd, 1992.

[77] S. Gnesi and G. Ristori. Model checking a mobile telephone system. In S. Gnesi and D. Latella, editors, *Proceedings of FMICS '97*, pages 113–123. Consiglio Nazionale Ricerche di Pisa, 1997.

[78] K. Havelund. *The Fork Calculus — A Theory of Communication, Concurrency and Process Creation.* PhD thesis, University of Copenhagen, Jan. 1994.

[79] M. Hennessy. A model for the π calculus. Report 8/91, University of Sussex, Brighton, 1991.

[80] M. Hennessy. A fully abstract denotational model for higher-order processes. *Journal of Information and Computation*, 112(1):55–95, 1994. An extended abstract appeared in the *Proceedings of LICS '93*, pages 397–408.

[81] M. Hennessy. Higher-order processes and their models. In S. Abiteboul and E. Shamir, editors, *Proceedings of ICALP '94*, volume 820 of *Lecture Notes in Computer Science*, pages 286–303. Springer-Verlag, 1994.

[82] M. Hennessy. A fully abstract denotational model for the π-calculus. Report 96:04, University of Sussex, Brighton, June 1996.

[83] K. Hirata. Embedding of moded flat GHC into polyadic π-calculus. In *International Symposium on Fifth Generation Computer Systems 1994, Workshop on Parallel Logic Programming*, pages 3–16. ICOT, 1994.

[84] K. Honda. Types for dyadic interaction. In E. Best, editor, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer-Verlag, 1993.

[85] K. Honda. Composing processes. In *Proceedings of POPL '96*, pages 344–357. ACM, Jan. 1996.

[86] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proceedings of ECOOP '91*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147, Geneva, Switzerland, July 1991. Springer-Verlag.

[87] K. Honda and M. Tokoro. A small calculus for concurrent objects. *ACM OOPS Messenger*, 2(2):50–54, 1991.

[88] K. Honda and M. Tokoro. On asynchronous communication semantics. In M. Tokoro, O. Nierstrasz and P. Wegner, editors, *Object-Based Concurrent Computing*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51. Springer-Verlag, 1992.

[89] K. Honda, V. T. Vasconcelos and M. Kubo. Language primitives and type discipline for structured communication-based programming. In C. Hankin, editor, *Proceedings of ESOP '98*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.

[90] K. Honda and N. Yoshida. Combinatory representation of mobile processes. In *Proceedings of POPL '94*. ACM, 1994.

[91] K. Honda and N. Yoshida. Replication in concurrent combinators. In M. Hagiya and J. C. Mitchell, editors, *Proceedings of TACS'94*, volume 789 of *Lecture Notes in Computer Science*, pages 786–805. Springer-Verlag, 1994.

[92] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. Also published in LNCS 761 (FSTTCS '93).

[93] F. Honsell, M. Lensia, U. Montanari and M. Pistore. Final semantics for the pi-calculus. In *Proceedings of PROCOMET '98*, 1998. To appear.

[94] H. Hüttel and J. Kleist. Objects as mobile processes. Research Series RS-96-38, BRICS, Oct. 1996. Presented at MFPS '96.

[95] A. Igarashi and N. Kobayashi. Type reconstruction for linear pi-calculus with I/O subtyping. In *Proceedings of SAS '97*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[96] R. Jagadeesan and P. Panangaden. A domain-theoretic model for a higher-order process calculus. In M. S. Paterson, editor, *Proceedings of ICALP '90*, volume 443 of *Lecture Notes in Computer Science*, pages 181–194. Springer-Verlag, 1990. Extended version as Technical Report 89-1058, Cornell University.

[97] C. B. Jones. A pi-calculus semantics for an object based design notation. In E. Best, editor, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, 1993.

[98] C. B. Jones. Process algebra foundations for an object-based design notation. Technical Report UMCS-93-10-1, University of Manchester, Oct. 1993.

[99] C. B. Jones. Process algebra arguments about an object-based design notation. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 1–15. Prentice Hall, 1994.

[100] C. B. Jones. Fixing the semantics of some concurrent object-oriented concepts. In S. Brookes, M. Main, A. Melton and M. Mislove, editors, *Proceedings of MFPS '95*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1995.

[101] J. Kennaway and M. Sleep. Expressions as processes. In *Symposium on* LISP *and Functional Programming (Pittsburgh, PA)*, pages 21–28. ACM, Aug. 1982.

[102] J. Kennaway and M. Sleep. Syntax and informal semantics of DyNe, a parallel language. In B. T. Denvir, W. T. Harwood, M. I. Jackson and M. J. Wray, editors, *The Analysis of Concurrent Systems 1983*, volume 207 of *Lecture Notes in Computer Science*, pages 222–230. Springer-Verlag, 1985.

[103] J. Kleist and M. Hansen. *Process Calculi with Asynchronous Communication*. Master's thesis, Aalborg University, Aug. 1994. Available as Report R-94-2028, ISSN 0908-1216.

[104] J. Kleist and D. Sangiorgi. Imperative objects and mobile processes. In *Proceedings of PROCOMET '98*, 1998. To appear.

[105] N. Kobayashi. A partially deadlock-free typed process calculus. In *Proceedings of LICS '97*, pages 128–139. IEEE, Computer Society Press, July 1997. Full version as as Technical Report 97-02, University of Tokyo.

[106] N. Kobayashi, B. C. Pierce and D. N. Turner. Linearity and the pi-calculus. In *Proceedings of POPL '96*, pages 358–371. ACM, Jan. 1996.

[107] C. Laneve. May and must testing in the join-calculus. Technical Report UBLCS-96-4, DoCS, University of Bologna, March 1996. Revised: May 1996.

[108] C. Laneve and U. Montanari. Mobility in the cc-paradigm. In I. M. Havel and V. Koubek, editors, *Proceedings of MFCS '92*, volume 629 of *Lecture Notes in Computer Science*, pages 336–345. Springer-Verlag, 1992.

[109] C. Lavatelli. Non-deterministic lazy λ-calculus vs. π-calculus. Technical Report LIENS-93-15, Ecole Normale Supérieure, Paris, Sept. 1993.

[110] L. Leth. *Functional Programs as Reconfigurable Networks of Communicating Processes*. PhD thesis, Imperial College, University of London, Feb. 1991.

[111] L. Leth. A new direction in functions as processes. Technical Report ECRC-92-5, European Computer-Industry Research Centre, Munich, 1992.

[112] B. Li. A π-calculus specification of Prolog. In D. Sannella, editor, *Proceedings of ESOP '94*, volume 788 of *Lecture Notes in Computer Science*, pages 379–393, Edinburgh, 1994. Springer-Verlag.

[113] Z. Li and H. Chen. Checking strong/weak bisimulation equivalences and observation congruence for the pi-calculus. In *Proceedings of ICALP '98*, 1998. To appear.

[114] H. Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical Report 7/94, School of Cognitive and Computing Sciences, University of Sussex, UK, 1994.

[115] H. Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. In P. D. Mosses, M. Nielsen and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, 1995. Presented in the CAAP-section. Available as Technical Report ISCAS-LCS-94-11, Institute of Software, Chinese Academy of Sciences, 1994.

[116] H. Lin. Unique fixpoint induction for mobile processes. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 88–102. Springer-Verlag, 1995.

[117] H. Lin. Complete proof systems for observation congruences in finite-control pi-calculus. In *Proceedings of ICALP '98*, 1998. To appear.

[118] X. Liu. Characterizing bisimulation congruence in the π-calculus. In J. Parrow and B. Jonsson, editors, *Proceedings of CONCUR '94*, volume 836 of *Lecture Notes in Computer Science*, pages 331–350. Springer-Verlag, 1994. Extended version as Report 2/94, University of Sussex, Brighton.

[119] X. Liu and D. Walker. Confluence of processes and systems of objects. In P. D. Mosses, M. Nielsen and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 217–231. Springer-Verlag, 1995. Presented in the CAAP-section. Available as University of Warwick Research Report CS-RR-272, October 1994.

[120] X. Liu and D. Walker. A polymorphic type system for the polyadic π-calculus. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 103–116. Springer-Verlag, 1995.

[121] X. Liu and D. Walker. Concurrent objects as mobile processes. In G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1998. To appear.

[122] M. Lumpe, J.-G. Schneider and O. Nierstrasz. Using metaobjects to model concurrent objects with PICT. In *Proceedings of Langages et Modèles à Objects (Leysin, October 1996)*, pages 1–12, Oct. 1997.

[123] T. F. Melham. A mechanized theory of the π-calculus in HOL. *Nordic Journal of Computing*, 1(1):50–76, 1994.

[124] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proceedings of ICALP '98*, 1998. To appear.

[125] D. Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of the 1992 Workshop on Extensions to Logic Programming*, volume 660 of *Lecture Notes in Computer Science*, pages 242–265. Springer-Verlag, 1993.

[126] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, Oct. 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[127] R. Milner. Sorts in the π-calculus (extended abstract). In E. Best and G. Rozenberg, editors, *Proceedings of the 3rd Workshop on Concurrency and Compositionality*, volume 191 of *GMD-Studien*. GMD Bonn, St. Augustin, 1991. Also available as Report 6/91 from University of Hildesheim.

[128] R. Milner. Action structures. Technical Report ECS-LFCS-92-249, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1992.

[129] R. Milner. Action structures and polynomial π-calculus. In E. Best, editor, *What Good are Partial Orders?, Proceedings of a CALIBAN-Workshop, Sheffield*, pages 77–105. Universität Hildesheim, Oct. 1992. Hildesheimer Informatik-Berichte 13/92.

[130] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[131] R. Milner. Action calculi, or concrete action structures. In A. M. Borzyszkowski and S. Sokołowski, editors, *Proceedings of MFCS '93*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer-Verlag, Sept. 1993. Also titled *Action Calculi I: Axioms and Applications*.

[132] R. Milner. An action structure for the synchronous π-calculus. In Z. Ésik, editor, *Proceedings of FCT'93*, volume 710 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, 1993.

[133] R. Milner. Action structures and the π-calculus. In *Proceedings of the NATO Summer School on Logic and Computation*. Springer-Verlag, 1993.

[134] R. Milner. Elements of interaction. *Communications of the ACM*, 36(1):78–89, 1993. Turing Award Lecture.

[135] R. Milner. Pi-nets: a graphical form of pi-calculus. In D. Sannella, editor, *Proceedings of ESOP '94*, volume 788 of *Lecture Notes in Computer Science*, pages 26–42, Edinburgh, 1994. Springer-Verlag.

[136] R. Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, Nov. 1996.

[137] R. Milner. Action structures for the π-calculus. Technical Report ECS-LFCS-93-264, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, May 1993.

[138] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.

[139] R. Milner, J. Parrow and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[140] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[141] U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 42–56. Springer-Verlag, 1995.

[142] U. Montanari and M. Pistore. Concurrent semantics for the π-calculus. In S. Brookes, M. Main, A. Melton and M. Mislove, editors, *Proceedings of MFPS '95*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1995.

[143] U. Nestmann. *On Determinacy and Nondeterminacy in Concurrent Programming*. PhD thesis, Technische Fakultät, Universität Erlangen, November 1996. Arbeitsbericht IMMD-29(14).

[144] U. Nestmann. What is a 'good' encoding of guarded choice? In C. Palamidessi and J. Parrow, editors, *Proceedings of EXPRESS '97*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elesevier Science Publishers, 1997. Revised full version as report BRICS-RS-97-45, Basic Research In Computer Science, Aalborg University, Denmark, December 1997.

[145] U. Nestmann and B. C. Pierce. Decoding choice encodings. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 1996. Revised full version as report ERCIM-10/97-R051, European Research Consortium for Informatics and Mathematics, 1997.

[146] U. Nestmann and M. Steffen. Correct transformational design of concurrent search structures. In B. Freitag, C. Jones, C. Lengauer and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 23–42. Kluwer Academic Publishers, 1996.

[147] U. Nestmann and M. Steffen. Typing confluence. In S. Gnesi and D. Latella, editors, *Proceedings of FMICS '97*, pages 77–101. Consiglio

Nazionale Ricerche di Pisa, 1997. Also available as report ERCIM-10/97-R052, European Research Consortium for Informatics and Mathematics, 1997.

[148] J. Niehren. Functional computation as concurrent computation. In *Proceedings of POPL '96*. ACM, Jan. 1996.

[149] M. Odersky. Applying π: Towards a basis for concurrent imperative programming. In U. S. Reddy, editor, *Proceedings of SIPL '95*, pages 95–108. Department of Computer Science, University of Illinois at Urbana-Champaign (San Francisco, California), January 22 1995. Available as Technical Report UIUCDCS-R-95-1900.

[150] M. Odersky. Polarized name passing. In P. S. Thiagarajan, editor, *Proceedings of FSTTCS '95*, volume 1026 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 1995.

[151] F. Orava. *On the Formal Analysis of Telecommunication Protocols*. PhD thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994.

[152] F. Orava and J. Parrow. Algebraic description of mobile networks: an example. In L. Logrippo, R. Probert and H. Ural, editors, *Proceedings of PSTV X*, pages 275–291. North-Holland, 1990.

[153] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4:497–543, 1992.

[154] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of POPL '97*, pages 256–265. ACM, Jan. 1997.

[155] J. Parrow. Interaction diagrams. *Nordic Journal of Computing*, 1995.

[156] J. Parrow. Trios in concert. In G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1998. To appear.

[157] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Journal of Information and Computation*, 120(2):174–197, 1995.

[158] J. Parrow and B. Victor. The update calculus. In M. Johnson, editor, *Proceedings of AMAST'97*, volume 1349 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, Dec. 1997. Full version available as Technical report DoCS 97/93, Uppsala University.

[159] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS '98*, 1998. To appear. Available from `http://www.docs.uu.se/victor/tr/fusion.html`.

[160] J. Parrow and B. Victor. The tau-laws of fusion. Submitted for publication. Available from `http://www.docs.uu.se/~victor/tr/taufusion.html`, Mar. 1998.

[161] A. Philippou. *Reasoning about Systems with Evolving Structure.* PhD thesis, University of Warwick, Dec. 1996.

[162] A. Philippou and D. Walker. On sharing and determinacy in concurrent systems. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 456–470. Springer-Verlag, 1995.

[163] A. Philippou and D. Walker. On transformations of concurrent object programs. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 131–146. Springer-Verlag, 1996. Full version to appear in *Theoretical Computer Science.*

[164] A. Philippou and D. Walker. On confluence in the π-calculus. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 314–324. Springer-Verlag, 1997.

[165] A. Philippou and D. Walker. A rigorous analysis of concurrent operations on B-trees. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of CONCUR '97*, volume 1243 of *Lecture Notes in Computer Science*, pages 361–375. Springer-Verlag, 1997.

[166] B. C. Pierce. Foundational calculi for programming languages. In A. B. Tucker, editor, *Handbook of Computer Science and Engineering*, chapter 139. CRC Press, 1996.

[167] B. C. Pierce. Programming in the pi-calculus: An experiment in programming language design. Tutorial notes on the Pict language. Available electronically, 1997.

[168] B. C. Pierce, D. Rémy and D. N. Turner. A typed higher-order programming language based on the pi-calculus. In *Workshop on Type Theory and its Application to Computer Systems, Kyoto University*, July 1993.

[169] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[170] B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *Proceedings of POPL '97*, pages 241–255. ACM, Jan. 1997. Full version available as INRIA-Sophia Antipolis Rapport de Recherche No. 3042 and as Indiana University Computer Science Technical Report 468.

[171] B. C. Pierce and D. N. Turner. Concurrent objects in a process calculus. In T. Ito and A. Yonezawa, editors, *Proceedings of TPPP '94*, volume 907 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[172] B. C. Pierce and D. N. Turner. Pict: User manual. Available electronically, 1997.

[173] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1998. To appear.

[174] M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In R. Alur, editor, *Proceedings of CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, 1996. Full version to appear in *Information and Computation*.

[175] C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995. Proceedings of PAPM '95.

[176] C. Priami. *Enhanced Operational Semantics for Concurrency*. PhD thesis, Dipartimento di Informatica, Università di Pisa-Genova-Udine, Mar. 1996. Available as PhD number TD-08/96.

[177] C. Priami. Integrating behavioural and performance analysis with topology information. In *Proceedings of 29^{th} Hawaiian International Conference on System Sciences*, volume 1, pages 508–516, Maui, Hawaii, 1996. IEEE.

[178] C. Priami. Interleaving-based partial ordering semantics. In *Proceedings of Italian Conference on Theoretical Computer Science*, pages 264–278, Ravello, November 1995, 1996. World Scientific.

[179] C. Priami. Stochastic pi-calculus with general distributions. In CLUP, editor, *Proceedings of PAPM '96*, 1996.

[180] C. Priami. Stochastic analysis of mobile telephony networks. In E. Brinskma and A. Nymeyer, editors, *Proceedings of PAPM'97*, pages 145–171, 1997.

[181] P. Quaglia. *The π-Calculus with Explicit Substitutions*. PhD thesis, Dipartimento di Informatica, Università di Pisa-Genova-Udine, Mar. 1996. Available as PhD number TD-09/96.

[182] P. Quaglia. On the finitary characterization of pi-congruences. Report Series RS-97-52, BRICS, Denmark, Dec. 1997.

[183] M. Radestock and S. Eisenbach. Towards a minimal object-oriented language for distributed and concurrent programming. In *Proceedings of PODC '94*, page 383. ACM, 1994. Abstract at the Poster Session. Full version as Technical Report, Imperial College, University of London.

[184] M. Radestock and S. Eisenbach. What do you get from a π-calculus semantics? In C. Halatsis, D. Maritsas, G. Philokyprou and S. Theodoridis, editors, *Proceedings of PARLE '94*, volume 817 of *Lecture Notes in Computer Science*, pages 635–647. Springer-Verlag, July 1994.

[185] N. Raja and R. K. Shyamasundar. Combinatory formulations of concurrent languages. In K. Kanchanasut and J.-J. Levy, editors, *Proceedings of ACSC'95*, volume 1023 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, Dec. 1995.

[186] N. Raja and R. K. Shyamasundar. Type systems for mobile processes. In S. Bai, J. Fan and X. Li, editors, *The Fourth International Conference for Young Computer Scientists (ICYCS'95)*, pages 622–629, Beijing, P. R. China, July 1995. Peking University Press.

[187] N. Raja and R. K. Shyamasundar. Actors as a coordinating model of computation. In *Perspectives of System Informatics (PSI'96) Proc. of the Andrei Ershov Second International Memorial Conference, Novosibirsk, Russia.*, Lecture Notes in Computer Science. Springer-Verlag, June 1996. To appear.

[188] A. Ravara and V. T. Vasconcelos. Behavioural types for a calculus of concurrent objects. In C. Lengauer, M. Griebl and S. Gorlatch, editors, *Proceedings of 3rd International EURO-PAR Conference*, volume 1300 of *Lecture Notes in Computer Science*, pages 554–561. Springer-Verlag, 1997. Preprint as Technical Report 6/97 from Departamento de Matemática, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[189] J. Riely and M. Hennessy. Distributed processes and location failures. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 471–481. Springer-Verlag, 1997. Full version as Report 2/97, University of Sussex, Brighton.

[190] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings of POPL '98*. ACM, Jan. 1998.

[191] B. Robben, F. Piessens and W. Joosen. Formalizing Correlate — from practice to pi. In *Proceedings of the 2nd BCS-FACS Northern Formal Methods Workshop*. Springer-Verlag, 1997.

[192] A. W. Roscoe, editor. *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice Hall, 1994.

[193] B. Ross. A π-calculus semantics of logical variables and unification. In S. Purushothaman and A. Zwarico, editors, *Proceedings of First North American Process Algebra Workshop, Stony Brook, 1992*, Workshops in Computing, pages 216–230. Springer-Verlag, 1993.

[194] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1993.

[195] D. Sangiorgi. From π-calculus to Higher-Order π-calculus — and back. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proceedings of TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166. Springer-Verlag, 1993.

[196] D. Sangiorgi. An investigation into functions as processes. In S. Brookes, M. Main, A. Melton and D. Schmidt, editors, *Proceedings of MFPS '93*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer-Verlag, 1993.

[197] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Journal of Information and Computation*, 111(1):120–131, 1994. An extended abstract appeared in Proc. of LICS'92.

[198] D. Sangiorgi. Internal mobility and agent-passing calculi. In Z. Fülöp and F. Gécseg, editors, *Proceedings of ICALP '95*, volume 944 of *Lecture Notes in Computer Science*, pages 672–683. Springer-Verlag, 1995.

[199] D. Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche RR-2515, INRIA Sophia-Antipolis, 1995.

[200] D. Sangiorgi. On the proof method for bisimulation (extended abstract). In J. Wiederman and P. Hájek, editors, *Proceedings of MFCS '95*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer-Verlag, 1995. To appear in *Mathematical Structures in Computer Science.* Full version available electronically.

[201] D. Sangiorgi. pi-I: A symmetric calculus based on internal mobility. In P. D. Mosses, M. Nielsen and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, 1995.

[202] D. Sangiorgi. Bisimulation in higher-order process calculi. *Journal of Information and Computation*, 131:141–178, 1996. Available as Rapport de Recherche RR-2508, INRIA Sophia-Antipolis, 1995. An early version appeared in the *Proceedings of PROCOMET'94*, pages 207–224. IFIP. North Holland Publisher.

[203] D. Sangiorgi. An interpretation of typed objects into typed π-calculus. Rapport de Recherche RR-3000, INRIA Sophia-Antipolis, Aug. 1996. Accepted for publication in *Information and Computation*.

[204] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155, 1996. See also Report ECS-LFCS-94-282, University of Edinburgh. An extract appeared in *Proceedings of TACS'94*, LNCS 789.

[205] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(1,2):235–274, 1996. Also as Rapport de Recherche RR-2539, INRIA Sophia-Antipolis, 1995. Extracts of parts of the material contained in this paper can be found in the Proceedings of TAPSOFT'95 and ICALP'95.

[206] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996. Earlier version published as Report ECS-LFCS-93-270, University of Edinburgh. An extended abstract appeared in the *Proceedings of CONCUR '93*, LNCS 715.

[207] D. Sangiorgi. The name discipline of uniform receptiveness. In P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 303–313. Springer-Verlag, 1997. Full version as Technical Report, INRIA Sophia-Antipolis, December 1996.

[208] D. Sangiorgi. The typed π-calculus at work: A proof of Jones's parallelisation theorem on concurrent objects. In B. C. Pierce, editor, *Proceedings of FOOL 4*, page 12. Free electronic publication, Jan. 1997. To appear in *Theory and Practice of Object-Oriented Systems*.

[209] J.-G. Schneider and M. Lumpe. Synchronizing concurrent objects in the pi-calculus. In R. Ducournau and S. Garlatti, editors, *Proceedings of Langages et Modèles à Objects '97*, pages 61–76. Hermes, Roscoff, Oct. 1997.

[210] T. Sekiguchi and A. Yonezawa. A calculus with code mobility. In *Prodeedings of FMOODS'97, Canterbury, July*. IFIP, Chapman and Hall, 1997.

[211] P. Sewell. On implementations and semantics of a concurrent programming language. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of CONCUR '97*, volume 1243 of *Lecture Notes in Computer Science*, pages 391–405. Springer-Verlag, 1997.

[212] P. Sewell. Global/local subtyping and capability inference for a distributed pi-calculus. In *Proceedings of ICALP '98*, 1998. To appear. Full version as Technical Report 435, Computer Laboratory, University of Cambridge.

[213] G. Smolka. A foundation for higher-order concurrent constraint programming. In J.-P. Jouannaud, editor, *Proceedings of Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 50–72. Springer-Verlag, Sept. 1994. Available as Research Report RR-94-16 from DFKI Kaiserslautern.

[214] I. Stark. A fully abstract domain model for the pi-calculus. In *Proceedings of LICS '96*, pages 36–42. IEEE, Computer Society Press, July 1996.

[215] G. Stefanescu. The algebra of flownomials part 1: Binary flownomials; basic theory. Report, Technical University Munich, Nov. 1994.

[216] K. Takeuchi, K. Honda and M. Kubo. An interaction-based language and its typing system. In C. Halatsis, D. Maritsas, G. Philokyprou and S. Theodoridis, editors, *Proceedings of PARLE '94*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer-Verlag, July 1994.

[217] B. Thomsen. A calculus of higher order communicating systems. In *Proceedings of POPL '89*, pages 143–154. ACM, January 1989.

[218] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Imperial College, University of London, Sept. 1990.

[219] B. Thomsen. Plain CHOCS. A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[220] B. Thomsen. A theory of higher order communicating systems. *Journal of Information and Computation*, 116(1):38–57, 1995.

[221] B. Thomsen, L. Leth and T.-M. Kuo. A Facile tutorial. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 278–298. Springer-Verlag, 1996.

[222] D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1995.

[223] F. van Breugel. A labelled transition system for π_ϵ-calculus. In M. Bidoit and M. Dauchet, editors, *Proceedings of TAPSOFT '97*, volume 1214 of *Lecture Notes in Computer Science*, pages 312–336, Lille, France, Apr. 1997. Springer-Verlag.

[224] V. T. Vasconcelos. Predicative polymorphism in π-calculus. In C. Halatsis, D. Maritsas, G. Philokyprou and S. Theodoridis, editors, *Proceedings of PARLE '94*, volume 817 of *Lecture Notes in Computer Science*, pages 425–437. Springer-Verlag, July 1994.

[225] V. T. Vasconcelos. *A process-calculus approach to typed concurrent objects*. PhD thesis, Keio University, 1994.

[226] V. T. Vasconcelos. Typed concurrent objects. In M. Tokoro and R. Pareschi, editors, *Proceedings of ECOOP '94*, volume 821 of *Lecture Notes in Computer Science*, pages 100–117. Springer-Verlag, 1994.

[227] V. T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In E. Best, editor, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 524–538. Springer-Verlag, 1993.

[228] V. T. Vasconcelos and M. Tokoro. A typing system for a calculus of objects. In S. Nishio and A. Yonezawa, editors, *1st International Symposium on Object Technologies for Advanced Software*, volume 742 of *Lecture Notes in Computer Science*, pages 460–474. Springer-Verlag, Nov. 1993.

[229] A. Venet. Abstract interpretation of the π-calculus. In M. Dam, editor, *Proceedings of the Fifth LOMAPS Workshop*, volume 1192 of *Lecture Notes in Computer Science*, pages 51–75. Springer-Verlag, 1997.

[230] B. Victor. Symbolic characterizations and algorithms for hyperequivalence and open bisimulation. Submitted for publication. Available from `http://www.docs.uu.se/~victor/tr/symhyper.html`, Mar. 1998.

[231] B. Victor. *A Verification Tool for the Polyadic π-Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.

[232] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *Proceedings of CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[233] B. Victor and J. Parrow. Constraints as processes. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 389–405. Springer-Verlag, 1996.

[234] B. Victor and J. Parrow. Concurrent constraints in the fusion calculus. In *Proceedings of ICALP '98*, 1998. To appear. Available from `http://www.docs.uu.se/~victor/tr/ccfc.html`.

[235] D. Walker. π-calculus semantics of object-oriented programming languages. In T. Ito and A. Meyer, editors, *Proceedings of TACS '91*, volume 526 of *Lecture Notes in Computer Science*, pages 532–547. Springer-Verlag, 1991. Available as Report ECS-LFCS-90-122, University of Edinburgh.

[236] D. Walker. Some results on the π-calculus. In *Concurrency: Theory, Languages, and Architecture (UK/Japan Workshop 1989)*, volume 491 of *Lecture Notes in Computer Science*, pages 21–35. Springer-Verlag, 1991.

[237] D. Walker. Process calculus and parallel object-oriented programming languages. In P. Tvrdik, T. Casavant and F. Plasil, editors, *Parallel Computers: Theory and Practice*, pages 369–390. Computer Society Press, 1993. Available as Research Report CS-RR-242, Department of Computer Science, University of Warwick.

[238] D. Walker. Algebraic proofs of properties of objects. In D. Sannella, editor, *Proceedings of ESOP '94*, volume 788 of *Lecture Notes in Computer Science*, pages 501–516, Edinburgh, 1994. Springer-Verlag.

[239] D. Walker. On bisimulation in the π-calculus. In J. Parrow and B. Jonsson, editors, *Proceedings of CONCUR '94*, volume 836 of *Lecture Notes in Computer Science*, pages 315–330. Springer-Verlag, 1994.

[240] D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.

[241] S. Weber and B. Bloom. Metatheory of the π-calculus. Technical Report TR96-1564, Cornell University, Jan. 1996.

[242] N. Yoshida. Graph notation for concurrent combinators. In T. Ito and A. Yonezawa, editors, *Proceedings of TPPP '94*, volume 907 of *Lecture Notes in Computer Science*, pages 393–412. Springer-Verlag, 1995.

[243] N. Yoshida. Graph types for monadic mobile processes. In V. Chandru and V. Vinay, editors, *Proceedings of FSTTCS '96*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 1996. Full version as Technical Report ECS-LFCS-96-350, University of Edinburgh.

[244] X. Zhang and J. Potter. Class-based models in the pi-calculus. In C. Mingins, R. Duke and B. Meyer, editors, *Proceeding of The 25th International Conference in Technology of Object-Oriented Languages and Systems (TOOLS Pacific'97, Melbourne, Australia)*, pages 219–231, November 1997.

Department of Computer Systems Dissertation Series

85/03 Joachim Parrow, *Fairness Properties in Process Algebra*

87/09 Bengt Jonsson, *Compositional Verification of Distributed Systems*

90/21 Parosh A. Abdulla, *Decision Problems in Systolic Circuit Verification*

90/22 Ivan Christoff, *Testing Equivalences for Probabilistic Processes*

91/27 Hans A. Hansson, *Time and Probability in Formal Design of Distributed Systems*

91/31 Peter Sjödin, *From LOTOS Specifications to Distributed Implementations*

93/37 Linda Christoff, *Specification and Verification Methods for Probabilistic Processes*

93/40 Mats Björkman, *Architectures for High Performance Communication*

94/46 Fredrik Orava, *On the Formal Analysis of Telecommunication Protocols*

96/70 Lars Björnfot, *Specification and Implementation of Distributed Real-Time Systems for Embedded Applications*

97/80 Bengt Ahlgren, *Improving Computer Communication Performance by Reducing Memory Bandwidth Consumption*

98/98 Björn Victor, *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*