Imperial College of Science, Technology and Medicine
(University of London)
Department of Computing

Calculi for Higher Order Communicating Systems

by

Bent Thomsen

A thesis submitted for the degree of Doctor
of Philosophy of the University of London and for
the Diploma of Membership of the Imperial College.

September 20, 1990

Abstract

This thesis develops two Calculi for Higher Order Communicating Systems. Both calculi consider sending and receiving processes to be as fundamental as nondeterminism and parallel composition.

The first calculus called CHOCS is an extension of Milner's CCS in the sense that all the constructions of CCS are included or may be derived from more fundamental constructs. Most of the mathematical framework of CCS carries over almost unchanged. The operational semantics of CHOCS is given as a labelled transition system and it is a direct extension of the semantics of CCS with value passing. A set of algebraic laws satisfied by the calculus is presented. These are similar to the CCS laws only introducing obvious extra laws for sending and receiving processes. The power of process passing is underlined by a result showing that the recursion operator is unnecessary in the sense that recursion can be simulated by means of process passing and communication. The CHOCS language is also studied by means of a denotational semantics. A major result is the full abstractness of this semantics with respect to the operational semantics. The denotational semantics is used to provide an easy proof of the simulation of recursion.

Introducing processes as first class objects yields a powerful metalanguage. It is shown that it is possible to simulate various reduction strategies of the untyped λ–Calculus in CHOCS. As pointed out by Milner, CCS has its limitations when one wants to describe unboundedly expanding systems, e.g. an unbounded number of procedure invocations in an imperative concurrent programming language P with recursive procedures. CHOCS may neatly describe both call-by-value and call-by-reference parameter mechanisms for P. We also consider call-by-name and lazy parameter mechanisms for P.

The second calculus is called Plain CHOCS. Essential to the new calculus is the treatment of restriction as a static binding operator on port names. This calculus is given an operational semantics using labelled transition systems which combines ideas from the applicative transition systems described by Abramsky and the transition systems used for CHOCS. This calculus enjoys algebraic properties which are similar to those of CHOCS only needing obvious extra laws for the static nature

of the restriction operator. Processes as first class objects enable description of networks with changing interconnection structure and there is a close connection between the Plain CHOCS calculus and the π–Calculus described by Milner, Parrow and Walker: the two calculi can simulate one another.

Recently object oriented programming has grown into a major discipline in computational practice as well as in computer science. From a theoretical point of view object oriented programming presents a challenge to any metalanguage since most object oriented languages have no formal semantics. We show how Plain CHOCS may be used to give a semantics to a prototype object oriented language called O.

Acknowledgement

I would like to thank my supervisor Samson Abramsky and my "second supervisor" Iain Phillips for their guidance and encouragement. I will always remember their patience and understanding when I came with technical problems which I had spent days or even weeks thinking about and expected them not only to grasp the problems, but also to solve them in minutes. I am deeply grateful to them for letting me carry on during the long writing up period where I should have spent my time doing work for them.

Many people have commented on previous formal and informal presentations of this material; I would like to thank them all.
I thank the POPL 89 referees for their comments on the preliminary report on CHOCS [Tho89] especially for pointing me in the direction of the work by D. Austry and G. Boudol and for making me aware of the fact that my first definition of procedures in the semantics of P described dynamic binding.
I also thank the ICALP 90 referees for very thorough comments on the preliminary report on Plain CHOCS [Tho89b]. Their comments have been very useful for the presentations in chapter 5 and will be invaluable when I attempt to submit a revised paper on Plain CHOCS for some conference or journal.
Special thanks for comments go to the following people: J. Andersen, C. Atkinson, S. Brookes, J. Cozens, M. Dam, A. Giacalone, M. Hennessy, H. Hüttel, K. G. Larsen, L. Leth, R. Milner, P. Mishra, F. Nielson, L. Ong, S. Prasad, A. Skou, S. Smolka, B. Steffen and A. Stoughton.
I would like to express my gratitude to M. Dawson and P. Taylor for solving any problem concerning LaTeX and Unix which I have burdened them with on so many occasions.

To express enough thanks to my wife is impossible. I dedicate this work to you. Without our fruitful discussions the material presented here would not have been the same. This thesis would have contained much more bad English without your influence.

I would like to thank the many friends that I have made during my stay in London. In particular I would like to thank Pat and Ron Burnett for their warm friendship. Special thanks go to the family and friends at home. Without their help and support life in London would have been a lot more complicated.

Financial support for this work has been provided by the following institutions: From 1st of April 1987 to the 30th of September 1989 I was employed on a joint junior research fellowship from Institute of Mathematics, Århus University, Denmark and Department of Mathematics and Computer Science, Aalborg University Centre, Denmark.
Since the 1st of October 1989 I have been supported by a research assistantship on the Foundational Models for Software Engineering project (SERC GR-F 72475) within the Department of Computing, Imperial College.
I am grateful to The Danish Natural Science Research Council and The Danish Research Academy for travel funds, fees and extra living expenses.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

During the past two decades several notions for formal description of concurrent and nondeterministic systems have been proposed. Such systems may be hardware or software and they often involve some notion of processes which can evolve indefinitely. The motivation for such notions can be found in the need for rigorous specifications and formal verification of implementations meeting their specifications.

One may group the various theories into two classes according to their view of "true"-concurrency and nondeterministic interleaving. Theories such as Petri Nets [Rei85], Event Structures [Win80] and Mazurkiewicz Traces [Maz77] are representatives of the class which treats concurrency differently from nondeterministic interleaving. The second class mainly consists of the various notions of process algebras or process calculi such as CCS [Mil80, HenMil85, Mil89], CSP [Hoa85], SCCS [Mil83], ACP [BerKlo84], MEIJE [Sim85]. We shall use the term process calculus, as advocated by Milner in [Mil89], for the above class as opposed to the more common term process algebra, since not everything in this class is done algebraically, logic and other mathematical disciplines are used. Although much effort has recently been put into finding a commonly accepted theory for concurrent and nondeterministic systems, to date no such unifying theory has emerged, though the pioneering work by Boudol and Castellani [BouCas87], the work by Abramsky [Abr90b] and the work by Aceto [Ace89] show some promising hopes for the future.

In the past decade process calculus has proved extremely successful as a description tool for specification and formal reasoning about concurrent and nondeterministic systems, and the international standardization organization (ISO) have chosen to base its new standard for network specification LOTOS on the framework of process calculus [BolBri87].

Usually in process calculus the semantics of the specification language is given in terms of a labelled transition system in the style of Plotkin [Plo81] thus yielding an operational description of system behaviour.

On the basis of the operational semantics various notions of equivalence between processes (or systems) have been proposed reflecting various views on observability. Examples are *Trace equivalence* [Hoa81] identifying processes with the same behavioural language, *Failure equivalence* [Hoa85, Hen88] identifying processes with the same set of failures (impossible actions or behaviour) after a trace, *Bisimulation equivalence* [Par81, Mil83, Mil89] where processes must have matching states with identical action capabilities.

In process calculus, specifications and implementations are often both expressed in the same process language, the specification being a high level abstract process and the implementation being a more concrete description, usually constructed of several components in parallel. The equivalence relation is then used to relate process descriptions on different levels of abstraction and verification or correctness of the implementation with respect to the specification is then taken to be equivalence of the two. Usually we do not expect to derive the implementation directly from the specification. Rather we derive it through a series of small and successive refinements of the specification using the stepwise refinement approach. To ensure correctness of the implementation with respect to the specification it is necessary to prove equivalence of each refinement of the specification with its predecessor. To keep the process of refinement manageable each step of the refinement process usually consists of a small refinement of the current version of the specification. Let p be the small part considered to be too abstract and subject to refinement and let q be the more concrete version which replaces p. If C denotes the surrounding specification which is left unaffected by the refinement step then $C[p]$ is the specification before the refinement and $C[q]$ is the specification after. To complete the refinement step we have to prove that the new version is equivalent to its predecessor. Such a proof would in general not only have to deal with p and q but also the context C. Since C is usually expected to be large relative to p and q this calls for an increasing amount of work. However, if the equivalence enjoys the property of being a congruence relation then this ensures that to prove $C[p]$ equivalent to $C[q]$ it suffices to prove p equivalent to q. This is one of the main reasons why so much emphasis is put on ensuring that the equivalence relation under consideration in most process calculi is in fact a congruence relation.

For some purposes an inequality theory based on a behavioural preorder may refine the notion of equivalence. In [Hoa85] a preorder $p \sqsubseteq q$ with the interpretation of p being more nondeterministic than q is used. One can use this

to express a specification with nondeterminism built in. Implementations should then resolve this nondeterminism and successively become more deterministic. In [Mil81b, Hen84, Wal88, Abr87] preorders which refine the notion of bisimulation by an explicit treatment of divergence were presented. [LarTho88] presents a preorder which refines the notion of bisimulation by explicit treatment of under or partial specification. It is pointed out that the stepwise refinement strategy normally used in process calculus may have one drawback. The subspecification may be more complex than strictly necessary because it has to cover not only the behaviour in the particular system C but in order to ensure congruence it has to cover behaviour in any context. The preorder may be used to ease the task of stepwise refinement by leaving parts of the subsystem unspecified, the only constraint being that the overall specification is total. In [Wal88, LarTho88] the preorders under certain constraints (e.g. absence of divergence resp. total specification) degenerate to bisimulation equivalence.

Most process calculi provide a set of laws which are sound with respect to the underlying operational equivalence/preorder. This allows the task of verification to be subjected to algebraic reasoning. In many cases (see e.g. [HenMil85, Hen88, Mil89]) sound and complete proof systems for laws concerning various sublanguages are provided. Recently several proof checking systems have been constructed. Tools like the Concurrency Workbench [CleParSte89], TAV [GodLarZee89] and AUTO [LecMadVer88] may be used to construct specifications and implementations and leave the task of verification to be carried out automatically by the system.

Instead of using the same language for both specification and implementation logical languages have been developed to ease the task of specification. Languages such as Hennessy–Milner Logic (HML) [HenMil85], Synchronization Tree Logic (STL) [GraSif86] and also a Modal Process Logic (MPL) [LarTho88b] may be viewed as representatives for this discipline. The task of verification then boils down to showing that an implementation satisfies its logical specification. STL and MPL may be seen as attempts to extend logical specifications with composition operators relating to the constructs on the underlying process. This yields a form of compositionality in the specification that gets inherited in the implementation. Logical specifications may be used to do process synthesis automatically. Algorithms that construct a process from the logical specification have been made see e.g. [Lar87, BouLar89]. These processes may be taken as idealized implementations and may of course be further refined by the usual algebraic methods.

Although denotational semantics have been used to validate equational laws in process calculi [Hoa85, Hen88] the methods of denotational semantics have not been widely used for verification purposes in process calculus though its inherent compo-

sitional nature should encourage its use. To some extent the success of the operational approach has overshadowed the need for denotational descriptions. Another reason may be early failures to find denotational semantics which were fully abstract with respect to bisimulation equivalence, but recently very promising results have shown how to obtain fully abstract denotational semantics for bisimulation equivalence [Abr90a] enabling the use of denotational methods to be introduced in process calculus.

1.2 Motivation

A calculus for computation should provide a mathematical framework for the description of and reasoning about computing systems all inside the calculus. CCS has proved to be a very successful tool for reasoning about the complex nature of nondeterministic and concurrent systems and it is an excellent representative for the process calculus approach to specification and verification of such systems. Each constructor in this calculus has been carefully chosen to provide a minimal set of primitive constructions from which one may build more complex systems [Mil86]. At the same time CCS provides a rich and well developed theory and its expressive power compares favourably with other process calculi [Sim85]. According to [Mil80], one of the original intentions of CCS was that it should serve as the λ–Calculus of concurrent systems. Subsequent research shows that it serves well as such for a large range of applications. But, as already pointed out in [Mil80], it has its limitations when one wants to describe unboundedly expanding systems as e.g. an unbounded number of procedure invocations in an imperative concurrent programming language.

I believe that this deficiency is due to the first order nature of CCS and most process calculi in general. Later extensions of CCS, such as SCCS [Mil83], ECCS [EngNie86] and Mobile Processes [MilParWal89], which allow dynamic use of communication channels, may be used to take care of the above problem.

I take the view that it is natural to attempt to solve the above problems by introducing some notion of higher order constructs and treating objects of the language as first class citizens. The extensions of CCS proposed in ECCS and Mobile Processes seem low level and far removed from the λ–Calculus analogy, since they do not explicitly support any higher order constructions.

Higher order constructs arise in almost any branch of theoretical computer science. The justification for having process passing in a calculus of communicating systems may be found in the powerful and elegant abstraction technique it yields, just as with (higher order) functions or procedures in traditional programming lan-

guages. Many systems can be easily described using process passing, some are even most naturally described in this way. As an excellent example take the system consisting of a satellite and an earth station originally described by P. Christensen in [Chr88]. One interesting property of this system is that the satellite is physically far away from the earth station. If the program controlling the satellite has to be changed, either because of a program error or because the job of the satellite is to be changed, then it would be preferable to be able to send a new program to the satellite, stop the old program and run the new program instead. Alternatively we would have to send a space shuttle to take the satellite out of orbit to bring it back to earth for reprogramming and then relaunch it, a rather expensive strategy. A reprogrammable system consisting of two components could be specified, in a CCS-like syntax, as follows:

$$sat = newprg?x.(x \mid (int?.sat + error?.sat + end?.sat))$$

$$earth = newprg!job_1.newprg!job_2....$$

The satellite is ready to receive a new job on the *newprg* channel. After reception it acts according to this job until it is "interrupted" either by a new job or because a program error has occurred or because the job has finished. In this example we are beyond CCS because of $newprg?x.(x \mid \ldots)$, what we receive on the *newprg* channel is a program (a process), we then run this program in parallel with the rest of the system.

Recently some promising treatments of processes as first class objects in CCS-like languages have been proposed [AstReg87, KenSle88, Chr88, Bou89, GiaMisPra89, Nie89]. The SMoLCS-framework [AstReg87] is a general theory for specifying processes as abstract data types including processes as values and higher order functions. The framework includes operational semantics in the form of algebraic labelled transition systems [AstGioReg88] and denotational semantics [AstReg87b]. Both [KenSle88] and [Chr88] focus on formulating denotational semantics for CCS respectively CSP with processes as communicable values. The main emphasis in these references is to establish specialized functors for process passing and ensuring that these have properties which enable them to be used together with standard functors used in denotational semantics. In [Bou89] a CCS-like language with special operators for "application" and process passing is presented. The language can also be viewed as a variant of the λ–Calculus extended with communication and interleaving operators. It is shown how this language can describe an interpretation of the pure λ–Calculus. A symmetric integration of concurrent and functional programming is presented in [GiaMisPra89]. The language FACILE is a typed higher order functional language with statements for channel creation and includes

a process language with parallel composition and (nondeterministic) choice operators. The semantics of FACILE was given in terms of an abstract machine in [GiaMisPra89], but has later been given an operational semantics in terms of labelled transition systems [GiaMisPra90]. This paper also presents an abstracting equivalence and some of its algebraic properties. [Nie89] presents a mixture of a typed λ–Calculus and a CCS-like language, called TPL, with processes as first class objects. It is shown how the types (including sorts of processes) of programs may be used to detect certain errors statically. An operational semantics for TPL is described, but no abstracting equivalence or preorder is considered.

To my knowledge, no study of process passing in its purified form has yet been presented, except for [Tho89] where preliminary results from this thesis were presented. By process passing in its purified form I mean searching for a minimal set of operators necessary and sufficient for this purpose along the line of viewing the pure λ–Calculus as a set of minimal operators for the study of functions in their pure form.

We therefore set out to crystallize the foundations of process passing and study calculi of communicating systems which consider sending and receiving processes to be as fundamental as nondeterminism and parallel composition.

This leads us to:

- Find a process language consisting of a set of well chosen operators which conforms to the principles of [Mil86] and provides a minimal but expressive set of constructors, preferably including or encoding the operators of CCS.
- Find a semantic foundation for the process language and explore its properties.
- Study abstracting equivalences and preorders which preferably have the property of being (pre-)congruences to ensure compositionality, enabling the use of partial specification as described in [Wal88, LarTho88].
- Exemplify the expressive power of the theories by application to examples.

The introduction of processes as communicable values suggests that we have a notion as powerful as higher order functions and the connection with the λ–Calculus should be investigated. In [MilParWal89] a framework for communicating systems with dynamically changing interconnection structure is presented. It is suggested that this may be used to simulate process passing; we should therefore investigate the relationship between link passing and process passing.

1.3 Overview

The main objective of this thesis is to study how processes as first class objects can be introduced in process calculus and to contribute with an investigation of the expressive power of doing so.

We stay firmly within the framework of process calculus and treat concurrency as nondeterministic interleaving.

We put forward two calculi, CHOCS and Plain CHOCS, with slightly different syntaxes, semantics and abstracting equivalences. Common to both is that they consider sending and receiving processes to be as fundamental as nondeterminism and parallel composition.

The thesis is divided into two parts. Part I consists of chapter 2 to 4 and concerns the CHOCS calculus. Part II consists of chapter 5 and describes the Plain CHOCS calculus.

The CHOCS calculus may be seen as an extension or an adaptation of CCS with value passing. In chapter 2 we present the syntax and operational semantics of CHOCS. Inspired by the developments of [Tho87] we propose a definition of bisimulation which takes processes passed in communication into account, we call this predicate higher order bisimulation and we investigate the algebraic theory of CHOCS. This theory turns out to contain the algebraic theory of CCS, the only addition being some natural laws concerning sending and receiving processes in communication. We show how process passing can be used to simulate recursion using a construction which resembles the Y-combinator of the untyped λ–Calculus. Clearly the set of port names which processes may use for communication through their life time plays an essential rôle especially for implementation considerations. We show how the concept of sort may be used when reasoning about higher order communicating systems. We present a sort inference system inspired by the type system of [Nie89]. The syntax and operational semantics of CHOCS may be formulated using the SMoLCS framework [AstReg87] and the definition of higher order bisimulation may be obtained as a specialization of the generalized bisimulation of [AstGioReg88] as demonstrated to me by Prof. E. Astesiano in [Ast89]. I shall not pursue this any further in this thesis since the framework of SMoLCS [AstReg87] does not seem to clarify the formulation of theories in this thesis. A bisimulation predicate similar to higher order bisimulation has been presented in [Bou89]. This predicate takes termination of processes into account to allow certain sequential behaviour which we shall not study in this thesis.

A new theory of communicating systems should be able to describe interesting systems which are hard or impossible to describe using existing theories. In chapter

3 we study three applications of CHOCS in the description of higher order communicating systems. First we study the λ–Calculus and we show how to encode function abstraction and function application in CHOCS. We show how to obtain different evaluation strategies such as lazy evaluation and call-by-value evaluation by varying the interpretation. Secondly we study the imperative toy language P from [Mil80] and we show how neatly process passing can cater for the problems of unbounded number of call-by-value procedure invocations recognized as a problem in [Mil80]. We also show how to describe call-by-reference procedure invocation in P following ideas from [EngNie86]. A third case study consists of a description of a fault tolerant editor inspired by systems investigated in [Pra88]. The editor system is described with an automatic reboot system which in the case of a fault occurring restarts the editor in the state just before the fault occurred, thus protecting the user from errors out of his/her control. This system can easily be generalized to an operating systems setting.

There have been a few attempts to give denotational descriptions of process languages with process passing [KenSle88, Chr88]. Both are formulated in a category theoretical setting and the main purpose of both papers is to establish functors describing the properties of process passing. A lot of effort is put into assuring that these functors can be used together with standard domain constructors and in recursive domain equations. I believe that it is not necessary to establish special functors for this purpose and that standard domain theory is sufficient to give denotational semantics for languages with processes as first class objects. In chapter 4 we follow the ideas of [Abr90a] and construct a denotational semantics for CHOCS which resides in a domain constructed using the standard constructions of separated sum, Cartesian product, the Plotkin power domain constructor and recursively defined domains. We show, under mild restrictions, that the denotational semantics and the operational semantics of CHOCS are fully abstract. We also use the denotational semantics to obtain a simpler proof of the simulation of recursion result.

Inspired by the ideas of [EngNie86, MilParWal89] of the restriction operator being a scope binder we put forward the Plain CHOCS calculus which includes this facility in chapter 5. It turns out that this calls for a different operational setting and we introduce the notion of higher order applicative transition systems and propose a notion of applicative higher order bisimulation inspired by the notion of applicative (bi)simulation from [Abr90] and the notion of strong ground bisimulation of [MilParWal89]. We investigate its algebraic properties which are similar to the algebraic properties of Mobile Processes [MilParWal89] with the addition of laws for sending and receiving processes. We make the connection between process

passing and link passing explicit by showing that they may be used to simulate one another. As an example of the use of Plain CHOCS as a metalanguage we show how one may give a formal semantics to an object oriented programming language called O. The language O is a prototype object oriented programming language which features most common concepts from object oriented programming such as class, inheritance and even object passing in method calls.

Finally in chapter 6 we summarize the contribution of this thesis and we give some directions for future work.

Chapter 2

Operational Theory of CHOCS

It has become almost a standard technique to define the semantics of process languages in process calculi in terms of labelled transition systems and thereby provide an operational semantics framework. In this chapter we present the CHOCS calculus, its syntax and its operational semantics. We also present abstracting equivalences and preorders built on the concept of bisimulation and we study the algebraic properties these enjoy. We start by reviewing the definition of labelled transition systems and the definitions and properties related to the notion of bisimulation.

2.1 Transition Systems and Bisimulation

When defining semantics of process languages in process calculi it has become almost standard to give the semantics in terms of a labelled transition system. This yields a method of defining processes, concurrent or nondeterministic, by the set of experiments they offer to an observer. Labelled transition systems are a simple model of nondeterminism based upon the primitive notion of state and transition.

Definition 2.1.1 *(Definition 1.4 in [Plo81])*
A labelled transition system *is a structure* $(St, Act, \longrightarrow)$, *where* St *is a set of* states *(or configurations),* Act *is a set of* actions *(or labels or operations) and* $\longrightarrow \subseteq St \times Act \times St$ *is the* transition relation.

For $(s, \Gamma, t) \in \longrightarrow$ we shall write $s \xrightarrow{\Gamma} t$ which may be interpreted as in state s the system may perform a Γ action and in doing so evolve to a state t. We use the usual abbreviations as e.g. $s \xrightarrow{\Gamma}$ for $\exists t \in St.s \xrightarrow{\Gamma} t$ and $s \not\xrightarrow{\Gamma}$ for $\neg\exists t \in St.s \xrightarrow{\Gamma} t$.

We shall identify the state of a process by the process, yielding a transition system $\mathcal{P} = (Pr, Act, \longrightarrow)$ modelling the operational semantics of a system of processes.

As an abstracting equivalence between processes defined in terms of labelled transition systems bisimulation [Par81, Mil83] is commonly accepted as the finest extensional or behavioural equivalence between processes one would impose:

Definition 2.1.2 *A* bisimulation *R is a binary relation on Pr such that whenever pRq and $\Gamma \in Act$ then:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, then $q \xrightarrow{\Gamma} q'$ for some q' with $p'Rq'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, then $p \xrightarrow{\Gamma} p'$ for some p' with $p'Rq'$

Two processes p and q are said to be bisimulation equivalent iff there exists a bisimulation R containing (p,q). In this case we write $p \sim q$.

Note how the processes have to match each others actions by syntactically equal actions. We shall relax this constraint in later parts of this thesis.

We may rephrase the above definition in terms of a functional $\mathcal{B}$ on the set of binary relations on Pr:

Definition 2.1.3 *$(p,q) \in \mathcal{B}(R)$ iff:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, then $q \xrightarrow{\Gamma} q'$ for some q' with $p'Rq'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, then $p \xrightarrow{\Gamma} p'$ for some p' with $p'Rq'$

It is easy to see that $\mathcal{B}$ is a monotone endofunction on the complete lattice of binary relations over Pr ordered by subset inclusion. Therefore by standard fixed point result, originally due to Tarski [Tar55], there exists a maximal fixed point for $\mathcal{B}$ and this fixed point equals $\bigcup\{R : R \subseteq \mathcal{B}(R)\}$. This fixed point equals $\sim$. It is easily verified that $\sim$ is itself a bisimulation. Moreover $\sim$ is an equivalence relation:

Proposition 2.1.4 *$\sim$ is an equivalence*

In addition $\sim$ enjoys the property of being a congruence relation with respect to the process construction in CCS [Mil80]. (The class of constructions for which bisimulation is a congruence has been studied in detail in [GroVaa89]).

In [Mil80] $\sim$ (now usually denoted $\sim_\omega$) was originally defined as the intersection of a decreasing sequence of equivalences on Pr:

Definition 2.1.5

- *$p \sim_0 q$ is always true (i.e. $\sim_0 = Pr \times Pr$)*

- $p \sim_{k+1} q$ *iff* $\forall \Gamma \in Act$:

  (i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \xrightarrow{\Gamma} q'$ *for some* q' *with* $p' \sim_k q'$

  (ii) Whenever $q \xrightarrow{\Gamma} q'$, *then* $p \xrightarrow{\Gamma} p'$ *for some* p' *with* $p' \sim_k q'$

(i.e. $\sim_{k+1} = \mathcal{B}(\sim_k)$*). Then* $\sim_\omega = \bigcap_{k\in\omega} \sim_k = \bigcap_{n\in\omega} \mathcal{B}^n(Pr^2)$

This decreasing sequence is bounded below by $\sim$ and we have $\sim \subseteq \sim_{k+1} \subseteq \sim_k$ for all k. The definition of $\sim_\omega$ is not in general a fixed point. However, under the condition that the transition systems is image finite it is.

Definition 2.1.6 *A transition system* $\mathcal{P} = (Pr, Act, \longrightarrow)$ *is said to be image finite iff for each process* p *of* Pr *and all actions* $\Gamma \in Act$ *the set* $\{p' : p \xrightarrow{\Gamma} p'\}$ *is finite.*

Proposition 2.1.7 *If* $\mathcal{P}$ *is image finite then* $\sim = \sim_\omega$ *on* $\mathcal{P}$.

$\sim$ and $\sim_\omega$ are defined relative to a transition system. Often we want to compare processes from different transition systems. This is done by taking their disjoint union and use $\sim$ resp. $\sim_\omega$ on this new transition system. We shall freely use this technique without further comment throughout this thesis.

2.2 Syntax and Semantics

In this section we introduce the syntax and operational semantics of a language for description of higher order communicating systems. CHOCS extends CCS as in [Mil80, Mil83, HenMil85] simply by allowing processes to be both sent and received and, equally important: to be used when received.

We presuppose an infinite set *Names* of channel names ranged over by $a, b, c, \ldots$ and an infinite set V of process variables ranged over by $x, y, z, \ldots$. A special symbol τ not in *Names* will be used to symbolize internal moves of processes. Let $p, q, r, \ldots$ (possible indexed and/or primed) range over process expressions with the following possible forms:

1. Inaction *nil*. This process may be thought of as the stopped process with no further communication capabilities.

2. Input prefix $a?x.p$. The prefix is a variable binder and x occurring free in p will be bound by this construct. The process has the capability of receiving any process on the a channel. The received process is put into use by substituting it for the bound variable.

3. Output prefix $a!p'.p$. This construct may be thought of as being able to send the process p' on the a channel and there after act as the process p.

4. Tau prefix $\tau.p$. This process performs the silent action τ and then behaves as p.

5. (Nondeterministic) choice $p+p'$. This process behaves as either p or p'. Which process is chosen depends on the communication capabilities and the choice may be nondeterministic.

6. Parallel composition $p \mid p'$. Processes composed in parallel act either asynchronously interleaved or by synchronized message passing producing τ-actions.

7. Restriction $p\backslash a$. This process acts like p except that communications on the a channel with components in its surrounding context are prohibited. Inside p communications along a can take place since they become silent τ-moves.

8. Renaming $p[S]$, where $S : \mathit{Names} \rightarrow \mathit{Names}$. This process acts as p but communication along channels are renamed according to S; e.g. if p can communicate via a then $p[S]$ can communicate via $S(a)$. We use the shorthand notation $p[a/b]$ for the renaming function which is the identity function on all $c \in \mathit{Names}$ except b where it returns a.

9. Process variables x are to be bound by input prefix. They act as place holders and do not occur free in programs.

The syntax of the expressions may be summarized as follows:

Definition 2.2.1

$$
\begin{array}{rcl}
p & ::= & \mathit{nil} \\
 & \mid & a?x.p \\
 & \mid & a!p'.p \\
 & \mid & \tau.p \\
 & \mid & p+p' \\
 & \mid & p \mid p' \\
 & \mid & p\backslash a \\
 & \mid & p[S] \\
 & \mid & x
\end{array}
$$

To avoid heavy use of brackets we adopt the following precedence of operators: restriction or renaming > prefix > parallel composition > choice.
We denote by Pr the set of processes built according to the above syntax. Readers familiar with CCS will notice that there is no recursion construct in CHOCS. We shall later see (Theorem 2.6.2) how recursive behaviours may be simulated using only process passing in communication.

We focus on the process passing and leave out details about other values. Pure synchronization may be obtained by ignoring the processes being sent and received. We shall use the sloppy notation $a?.p$ and $a!.p$ as action prefixing for pure synchronization. Other types of values may — with little theoretical overhead — be obtained simply by encoding the values in pure synchronization using the approach of [Mil83] by introducing a family of value indexed guards and generalizing the (nondeterministic) choice operator. We shall indeed use this technique in section 3.2 and we refer to this section for further discussion.

Input guards are variable binders. This implies a notion of free and bound variables.

Definition 2.2.2 *We define the set of free variables $FV(p)$ by induction:*

$$\begin{aligned}
FV(nil) &= \emptyset \\
FV(a?x.p) &= FV(p) \setminus \{x\} \\
FV(a!p'.p) &= FV(p) \cup FV(p') \\
FV(\tau.p) &= FV(p) \\
FV(p + p') &= FV(p) \cup FV(p') \\
FV(p \mid p') &= FV(p) \cup FV(p') \\
FV(p\backslash a) &= FV(p) \\
FV(p[S]) &= FV(p) \\
FV(x) &= \{x\}
\end{aligned}$$

A variable which is not free i.e. does not belong to $FV(p)$ is said to be bound in p.

The above definition may be rephrased as: x is free in p if x is not contained in any subexpression $a?x.p'$. An expression p is closed if $FV(p) = \emptyset$. Closed expressions are referred to as programs and we denote the set of programs by CPr.

To allow processes received in communication to be used we need a way of substituting the received processes for bound variables. Let $\overline{q} = (q_1, \ldots, q_n)$ be a vector of processes and $\overline{x} = (x_1, \ldots, x_n)$ be a vector of variables, then $p[\overline{q}/\overline{x}]$

describes the simultaneous substitution of expressions $\overline{q}$ for variables $\overline{x}$ in p. We always assume that $\overline{q}$ and $\overline{x}$ are compatible, i.e. have the same length and that $\overline{x}$ consists of distinct variables. We use the notation $FV(\overline{q}) = FV(q_1) \cup \ldots \cup FV(q_n)$ and in the case $\overline{x} = (x_1)$ we write $p[q/x_1]$. We also consider $\overline{x}$ as a set of variables and write $\overline{x} \cap FV(p) = \emptyset$ which means that $\overline{x}$ as a set does not have common elements with the set $FV(p)$.

Definition 2.2.3 *The substitution $p[\overline{q}/\overline{x}]$ is defined structurally on p:*

$$
\begin{aligned}
nil[\overline{q}/\overline{x}] &\equiv nil \\
(a?y.p)[\overline{q}/\overline{x}] &\equiv \begin{cases} a?y.(p[\overline{q}/\overline{x}]) & \text{if } y \notin \overline{x} \text{ and } y \notin FV(\overline{q}) \\ a?z.((p[z/y])[\overline{q}/\overline{x}]) & \text{otherwise} \\ & \text{for some } z \notin FV(p) \cup FV(\overline{q}) \cup \overline{x} \cup \{y\} \end{cases} \\
(a!p'.p)[\overline{q}/\overline{x}] &\equiv a!(p'[\overline{q}/\overline{x}]).p[\overline{q}/\overline{x}] \\
(\tau.p)[\overline{q}/\overline{x}] &\equiv \tau.(p[\overline{q}/\overline{x}]) \\
(p + p')[\overline{q}/\overline{x}] &\equiv (p[\overline{q}/\overline{x}]) + (p'[\overline{q}/\overline{x}]) \\
(p \mid p')[\overline{q}/\overline{x}] &\equiv (p[\overline{q}/\overline{x}]) \mid (p'[\overline{q}/\overline{x}]) \\
(p\backslash a)[\overline{q}/\overline{x}] &\equiv (p[\overline{q}/\overline{x}])\backslash a \\
(p[S])[\overline{q}/\overline{x}] &\equiv (p[\overline{q}/\overline{x}])[S] \\
(y)[\overline{q}/\overline{x}] &\equiv \begin{cases} q_i & \text{if } y = x_i \\ y & \text{otherwise} \end{cases}
\end{aligned}
$$

This definition extends the definition of substitution given in [Mil81] by allowing substitution in processes built using the parallel composition, restriction or renaming operators. To a certain extent this definition resembles the definition of substitution in the λ–Calculus as defined in [Bar84]. We shall pursue this further in a later section (see lemma 3.1.8). Note how substitution is straightforward only taking care of change of bound variables. In chapter 5 we modify this definition and study a version of CHOCS where the restriction operator acts as a binding operator on port names following the ideas of [EngNie86, MilParWal89].

Here are a few useful properties of substitution:

Proposition 2.2.4 *Let $\overline{p}[\overline{p}'/\overline{x}] = (p_1[\overline{p}'/\overline{x}], \ldots, p_n[\overline{p}'/\overline{x}])$.*

1. *If $\overline{x} \cap \overline{y} = \emptyset$ then $p[\overline{p}'/\overline{x}][\overline{p}''/\overline{y}] \equiv p[\overline{p}''/\overline{y}][\overline{p}'[\overline{p}''/\overline{y}]/\overline{x}]$.*
2. *$p[\overline{p}'/\overline{x}] \equiv p[\overline{p}''/\overline{x}'']$ where $\overline{x}'' = \overline{x} \cap FV(p)$ and $\overline{p}'' = (p_1, \ldots, p_m)$ with $p_j \in \{p'_j : x_j \in \overline{x}'' \ \&\ p'_j \in \overline{p}'\}$ i.e. $\overline{p}''$ is the projection of $\overline{p}'$ corresponding to $\overline{x}''$.*

3. $p[\overline{p}'/\overline{x}] \equiv p$ *if* $\overline{x} \cap FV(p) = \emptyset$.

4. *If* $\overline{x} \cap \overline{y} = \emptyset$ *and* $\overline{p}', \overline{p}''$ *are closed then* $p[\overline{p}'/\overline{x}][\overline{p}''/\overline{y}] \equiv p[\overline{p}''/\overline{y}][\overline{p}'/\overline{x}]$.

PROOF: The proofs of 1. and 2. are easily established by structural induction on p. The only case which needs special attention is when $p \equiv a?z.p_1$ and $z \in \overline{x} \cup \overline{y} \cup FV(\overline{p}') \cup FV(\overline{p}'')$. In this case we choose a "fresh" variable z_1 and proceed on $a?z_1.(p_1[z_1/z])$.
3. is a corollary of 2. Note the special case when p is closed.
4. follows as a corollary of 1. and 3. □

The operational semantics of CHOCS is given in terms of a labelled transition system. In [Mil80] a structure called communication trees, describing transitions of the form $p \xrightarrow{av} p'$, where v is some value, is used to give semantics to CCS with value passing in communication. We shall pursue this idea and from now on we shall consider labelled transition systems $\mathcal{P} = (Pr, Act, \longrightarrow)$, where Pr is the set of expressions (processes) built according to the syntax of definition 2.2.1 and where Act has the form $Names \times \{?,!\} \times Pr \cup \{\tau\}$ and $Names$ is an uninterpreted set referred to as a set of port names. We also study the subsystem of $\mathcal{P}$ where all expressions are closed: $\mathcal{CP} = (CPr, CAct, \longrightarrow)$, where $CAct$ has the form $Names \times \{?,!\} \times CPr \cup \{\tau\}$.

$p \xrightarrow{a?p'} p''$ may be read as "p can receive the process p' at port a and in doing so become the process p''".

$p \xrightarrow{a!p'} p''$ may be read as "p can send the process p' via port a and in doing so become the process p''".

$p \xrightarrow{\tau} p'$ may be interpreted as "the process p can do an internal or silent move and in doing so become the process p'".

Note that instead of insisting on an Abelian monoid structure on the set *Names* of port names as in [Mil83] we simply use the CSP–like notation of $?,!$ to indicate the input/output direction of communication. We call this structure higher order communication trees. The special symbol τ not in *Names* is used to symbolize internal moves of processes. We use r to stand for any action $a?p$, $a!p$ or τ. For actions of the form $a?p$ or $a!p$ let $\overline{a?p} = a!p$ and $\overline{a!p} = a?p$.

Definition 2.2.5 *Let* $\rightarrow$ *be the smallest subset of* $Pr \times Act \times Pr$, *where* $Act = Names \times \{?,!\} \times Pr \cup \{\tau\}$, *closed under the following rules:*

prefixing:	$a?x.p \xrightarrow{a?p'} p[p'/x]$	$a!p'.p \xrightarrow{a!p'} p$	$\tau.p \xrightarrow{\tau} p$
choice:	$\dfrac{p \xrightarrow{\Gamma} p'}{p+q \xrightarrow{\Gamma} p'}$	$\dfrac{p \xrightarrow{\Gamma} p'}{q+p \xrightarrow{\Gamma} p'}$	
parallel:	$\dfrac{p \xrightarrow{\Gamma} p'}{p \mid q \xrightarrow{\Gamma} p' \mid q}$	$\dfrac{p \xrightarrow{\Gamma} p'}{q \mid p \xrightarrow{\Gamma} q \mid p'}$	$\dfrac{p \xrightarrow{\Gamma} p'' \; q \xrightarrow{\overline{\Gamma}} q''}{p \mid q \xrightarrow{\tau} p'' \mid q''}$
restriction:	$\dfrac{p \xrightarrow{a?p'} p''}{p\backslash b \xrightarrow{a?p'} p''\backslash b}, a \neq b$	$\dfrac{p \xrightarrow{a!p'} p''}{p\backslash b \xrightarrow{a!p'} p''\backslash b}, a \neq b$	$\dfrac{p \xrightarrow{\tau} p''}{p\backslash b \xrightarrow{\tau} p''\backslash b}$
renaming:	$\dfrac{p \xrightarrow{a?p'} p''}{p[S] \xrightarrow{S(a)?p'} p''[S]}$	$\dfrac{p \xrightarrow{a!p'} p''}{p[S] \xrightarrow{S(a)!p'} p''[S]}$	$\dfrac{p \xrightarrow{\tau} p''}{p[S] \xrightarrow{\tau} p''[S]}$

Table 2.2.1: Operational semantics for CHOCS

A process guarded by input prefix has the capability of receiving any process. The received process is put into use by substituting it for the bound variable. Readers familiar with [Mil80] will recognize the similarities with the operational semantics for input guarding in CCS with value passing. Parallel composition acts either asynchronously interleaved or by synchronized message passing e.g. $a?x.p \mid a!p'.p''$ can perform $a?q$ or $a!p'$ as well as τ. The restriction and renaming operators have no effect on the processes sent or received. This is a matter of choice. The choice we have made for CHOCS in this chapter yields a very simple operational semantics for the two constructs.

2.3 Higher Order Bisimulation

To capture the observational behaviour of processes capable of sending and receiving processes we extend the notion of bisimulation. Bisimulation is commonly accepted

as the finest extensional or behavioural equivalence between processes that one would impose and the equivalence corresponds to a view where processes are black boxes only distinguishable by their interaction capabilities in different environments. As an excellent motivating example consider the following systems first presented in [AstGioReg88]:

Example 2.3.1
Let $p'_1 = p_3 + p_4$, $p'_2 = p_4 + p_3$, $p_1 = a!p'_1.nil$ *and* $p_2 = a!p'_2.nil$ *for some* p_3 *and* p_4. *If* $\sim$ *denotes bisimulation equivalence as defined in definition 2.1.2 then* $p_1 \not\sim p_2$ *since* $p_1 \xrightarrow{a!p'_1} nil$ *and* $p_2 \xrightarrow{a!p'_2} nil$ *but* $a!p'_1 \neq a!p'_2$. *The reason is that* p'_1 *and* p'_2 *are not syntactically equal, although we expect them to be equivalent.*

The extension of bisimulation should not distinguish between equivalent processes even when they are sent or received in communication. This is captured in the following definition:

Definition 2.3.2 *A* higher order bisimulation R *is a binary relation on* Pr *such that whenever* pRq *and* $\Gamma \in Act$ *then:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \xrightarrow{\Gamma'} q'$ *for some* q', Γ' *with* $\Gamma \widehat{R} \Gamma'$ *and* $p' R q'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, *then* $p \xrightarrow{\Gamma'} p'$ *for some* p', Γ' *with* $\Gamma' \widehat{R} \Gamma$ *and* $p' R q'$

Where $\widehat{R} = \{(\Gamma, \Gamma') : (\Gamma = a?p'' \ \& \ \Gamma' = a?q'' \ \& \ p''Rq'') \vee (\Gamma = a!p'' \ \& \ \Gamma' = a!q'' \ \& \ p''Rq'') \vee (\Gamma = \Gamma' = \tau)\}$.
Two processes p *and* q *are said to be higher order bisimulation equivalent iff there exists a higher order bisimulation* R *containing* (p, q). *In this case we write* $p \sim q$.

In the above definition $\widehat{R}$ takes care of extending R to the processes passed in communication. As we shall see later this has the effect that we do not distinguish between equivalent processes passed in communication.

We may rephrase the above definition in terms of a functional $\mathcal{HB}$ on the set of binary relations on Pr:

Definition 2.3.3 $(p, q) \in \mathcal{HB}(R)$ *iff:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \xrightarrow{\Gamma'} q'$ *for some* q', Γ' *with* $\Gamma \widehat{R} \Gamma'$ *and* $p' R q'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, *then* $p \xrightarrow{\Gamma'} p'$ *for some* p', Γ' *with* $\Gamma' \widehat{R} \Gamma$ *and* $p' R q'$

It is easy to see that $\mathcal{HB}$ is a monotone endofunction on the complete lattice of binary relations over Pr ordered by subset inclusion. Therefore by standard fixed point result, originally due to Tarski [Tar55], there exists a maximal fixed point for $\mathcal{HB}$ and this fixed point equals $\bigcup\{R \ : \ R \subseteq \mathcal{HB}(R)\}$. This fixed point equals $\sim$. It is easily verified that $\sim$ is itself a higher order bisimulation. Moreover $\sim$ is an equivalence relation:

Proposition 2.3.4 $\sim$ *is an equivalence*

PROOF: To see this observe that the relation: $Id = \{(p,p) \mid p \in Pr\}$ is a higher order bisimulation. This proves reflexivity.
Composition of higher order bisimulations yields a higher order bisimulation. This proves transitivity. Composition of relations $R, S \subseteq Pr^2$ is taken to be

$$R\,S = \{(p_1,p_3) \mid \exists p_2.(p_1,p_2) \in R \ \& \ (p_2,p_3) \in S\}.$$

Note how this is in the opposite order to function composition.
For all higher order bisimulations R the relation:

$$R^T = \{(p,q) \mid (q,p) \in R\}$$

is a higher order bisimulation. This proves symmetry. □

It is now easy to see that if a relation R is a bisimulation (definition 2.1.2) then we can extend R by Id and obtain a higher order bisimulation $R \cup Id$. To see that $R \cup Id$ is a higher order bisimulation observe that if $(p,q) \in R \cup Id$ then

either $(p,q) \in Id$ and whenever $p \xrightarrow{\Gamma} p'$ then $p = q \xrightarrow{\Gamma} q' = p'$ with $(\Gamma,\Gamma) \in \widehat{Id} \subseteq \widehat{R \cup Id}$ and $(p',q') \in Id \subseteq R \cup Id$.

or $(p,q) \in R$ and whenever $p \xrightarrow{\Gamma} p'$ then $q \xrightarrow{\Gamma} q'$ with $(\Gamma,\Gamma) \in \widehat{Id} \subseteq \widehat{R \cup Id}$ and $(p',q') \in R \subseteq R \cup Id$ since R is a bisimulation. A symmetric argument applies whenever $q \xrightarrow{\Gamma} q'$.

Note that R above is the set of objects which we want to prove properties about, i.e. bisimilar processes are also higher order bisimilar. The added Id relation acts as a kind of closure operation taking care of the processes sent and received. We shall later use this technique in proving algebraic laws for higher order bisimulation.

From now on we use the terms bisimulation and higher order bisimulation to mean higher order bisimulation. This justifies the ambiguous use of $\sim$.

We may now relate the process constructions of CHOCS to the underlying semantic equivalence $\sim$. As mentioned in section 2.1 the notion of bisimulation is relative to a particular transition system though we may relate processes from different transition systems by taking their disjoint union and usually we do this without explicit mention. To prove that $\sim$ is a congruence relation with respect to the operators of CHOCS we first study $\sim$ with respect to the transition system $\mathcal{CP}$ of closed expressions. Later we generalize this result to the transition system of all processes $\mathcal{P}$.

Let $\overline{q}_1 \sim \overline{q}_2$ mean $q_{1_j} \sim q_{2_j}$ for all $q_{i_j} \in \overline{q}_i$, $i \in 1,2$ and let $\overline{q}_i \in CPr$ mean $q_{i_j} \in CPr$ for all $q_{i_j} \in \overline{q}_i$.

Proposition 2.3.5 *$\sim$ is a congruence relation on programs (closed expressions).*

1. $p[\overline{q}_1/\overline{x}] \sim p[\overline{q}_2/\overline{x}]$ *if* $\overline{q}_1 \sim \overline{q}_2$ *and* $\overline{x} = FV(p)$
2. $a?x.p \sim a?x.q$ *if* $p[r/x] \sim q[r/x]$ *for all* r
3. $a!p'.p \sim a!q'.q$ *if* $p \sim q$ *and* $p' \sim q'$
4. $\tau.p \sim \tau.q$ *if* $p \sim q$
5. $p + p' \sim q + q'$ *if* $p \sim q$ *and* $p' \sim q'$
6. $p \mid p' \sim q \mid q'$ *if* $p \sim q$ *and* $p' \sim q'$
7. $p\backslash a \sim q\backslash a$ *if* $p \sim q$
8. $p[S] \sim q[S]$ *if* $p \sim q$

The proof of this proposition is quite involved and we need a few technical definitions and lemmas before presenting the proof. The reason for this is that we can not prove the congruence properties for CHOCS using the "standard" process calculus technique; i.e. prove that for each operator *op* in the process language the relation $R_{op} = \{op(\overline{p}_1), op(\overline{p}_2) \ : \ \overline{p}_1 \sim \overline{p}_2\}$ is a bisimulation and then prove the substitution property (i.e. that if $\overline{q}_1 \sim \overline{q}_2$ then $p[\overline{q}_1/\overline{x}] \sim p[\overline{q}_2/\overline{x}]$) by structural induction on p. This approach fails for CHOCS in the case of parallel composition since we need to know the substitution property to prove that the relation $R_|$ is a higher order bisimulation and we thus end up with a circular argument. This may at first seem surprising, but the "functional" nature of CHOCS may indicate that this property should be hard to prove: e.g. Abramsky has to give quite an argument to prove congruence properties of the Lazy–λ–Calculus in [Abr90].

Definition 2.3.6
Let $CR = \{(p[\overline{q}_1/\overline{x}], p[\overline{q}_2/\overline{x}]) \ : \ p \in Pr \ \& \ \overline{x} = FV(p) \ \& \ \overline{q}_1 \sim \overline{q}_2 \ \& \ \overline{q}_i \in CPr\}$ *and let* CR^* *be the reflexive and transitive closure of* CR *i.e.* $(p,q) \in CR^*$ *if there is a sequence* $p_1 \ldots p_n$ *such that* $(p,p_1) \in CR$, $(p_i, p_{i+1}) \in CR$ *for* $1 \leq i < n$ *and* $(p_n, q) \in CR$.

Note if $q_1 \sim q_2$ then $(x[q_1/x], x[q_2/x]) \in CR^*$ and we write $(q_1, q_2) \in CR^*$.

Lemma 2.3.7
If $p \in CPr$ *and* $p \xrightarrow{a?r} p'$ *and* $(r,r') \in CR$ *then* $p \xrightarrow{a?r'} p''$ *with* $(p',p'') \in CR$ *for some* p''.

PROOF: By induction on the length of the inference used to establish $p \xrightarrow{a?r} p'$ and cases of the structure of p. $p \equiv nil$, $p \equiv a!p_1.p_2$ and $p \equiv \tau.p_1$ are trivial since $p \not\xrightarrow{a?r}$.

$\underline{p \equiv a?y.p_1}$ By the operational semantics for input prefix $p \xrightarrow{a?r} p_1[r/y]$ for any r. Since p is closed either $FV(p_1) = \emptyset$ and $p_1[r/y] \equiv p_1$ and $p \xrightarrow{a?r'} p_1[r'/y] \equiv p_1$ and $(p_1,p_1) \in CR$ or $FV(p_1) = \{y\}$. If $(r,r') \in CR$ then $r \equiv r_1[\overline{q}_1/\overline{x}]$ and $r' \equiv r_1[\overline{q}_2/\overline{x}]$ for some r_1 with $FV(r_1) = \overline{x}$ and $\overline{q}_1 \sim \overline{q}_2$ for some closed $\overline{q}_i$. Then $p_1[r/y] \equiv p_1[r_1[\overline{q}_1/\overline{x}]/y] \equiv (p_1[r_1/y])[\overline{q}_1/\overline{x}]$ and $p_1[r'/y] \equiv p_1[r_1[\overline{q}_2/\overline{x}]/y] \equiv (p_1[r_1/y])[\overline{q}_2/\overline{x}]$ since $FV(p_1) = \{y\}$ and $FV(r_1) = \overline{x}$ we have $FV(p_1[r_1/y]) = \overline{x}$ and $((p_1[r_1/y])[\overline{q}_1/\overline{x}], (p_1[r_1/y])[\overline{q}_2/\overline{x}]) \in CR$.

$\underline{p \equiv p_1 + p_2}$ If $p \xrightarrow{a?r} p'$ then

either $p_1 \xrightarrow{a?r} p'$ by a shorter inference. By induction $p_1 \xrightarrow{a?r'} p''$ with $(p',p'') \in CR$. By the operational semantics for choice $p_1 + p_2 \xrightarrow{a?r'} p''$ with $(p',p'') \in CR$.

or $p_2 \xrightarrow{a?r} p'$ by a shorter inference and we may argue as above.

$\underline{p \equiv p_1 \mid p_2}$ If $p \xrightarrow{a?r} p'$ then

either $p_1 \xrightarrow{a?r} p_1'$ by a shorter inference and $p' \equiv p_1' \mid p_2$. By induction $p_1 \xrightarrow{a?r'} p_1''$ with $(p_1', p_1'') \in CR$. Thus for some p_3 with $FV(p_3) = \overline{x}$ and $\overline{q}_1 \sim \overline{q}_2$ for some closed $\overline{q}_i$ we have $p_1' \equiv p_3[\overline{q}_1/\overline{x}]$ and $p_1'' \equiv p_3[\overline{q}_2/\overline{x}]$. By the operational semantics for parallel $p_1 \mid p_2 \xrightarrow{a?r'} p_1'' \mid p_2$. Since p is closed p_2 is closed and $p_1' \mid p_2 \equiv (p_3 \mid p_2)[\overline{q}_1/\overline{x}]$ and $p_1'' \mid p_2 \equiv (p_3 \mid p_2)[\overline{q}_2/\overline{x}]$ and we have $(p_1' \mid p_2, p_1'' \mid p_2) \in CR$.

or $p_2 \xrightarrow{a?r} p'$ by a shorter inference and we may argue as above.

$\underline{p \equiv p_1\backslash b}$ If $p \xrightarrow{a?r} p'$ then $p_1 \xrightarrow{a?r} p_1'$ by a shorter inference and $p' \equiv p_1'\backslash b$ and $a \neq b$. By induction $p_1 \xrightarrow{a?r'} p_1''$ with $(p_1', p_1'') \in CR$. Thus for some p_3 with $FV(p_3) = \overline{x}$ and $\overline{q}_1 \sim \overline{q}_2$ for some closed $\overline{q}_i$ we have $p_1' \equiv p_3[\overline{q}_1/\overline{x}]$ and $p_1'' \equiv p_3[\overline{q}_2/\overline{x}]$. By the operational semantics for restriction $p_1\backslash b \xrightarrow{a?r'} p_1''\backslash b$. Thus we have $p_1'\backslash b \equiv (p_3\backslash b)[\overline{q}_1/\overline{x}]$ and $p_1''\backslash b \equiv (p_3\backslash b)[\overline{q}_2/\overline{x}]$ and $(p_1'\backslash b, p_1''\backslash b) \in CR$.

$\underline{p \equiv p_1[S]}$ Follows by an argument similar to the case $p \equiv p_1\backslash b$.

□

Lemma 2.3.8

If $p \in CPr$ and $p \xrightarrow{a?r} p'$ and $(r, r') \in CR^*$ then $p \xrightarrow{a?r'} p''$ with $(p', p'') \in CR^*$ for some p''.

PROOF: By induction on the length of the sequence $p_1 \ldots p_n$. The base case is covered by lemma 2.3.7 and the inductive step follows by the induction hypothesis and lemma 2.3.7. □

PROOF: (of proposition 2.3.5.1)
We show that the relation CR^* is a higher order bisimulation.
To see this we show that if $(p_1, p_2) \in CR$ then $p_1 \equiv p[\overline{q}_1/\overline{x}]$ and $p_2 \equiv p[\overline{q}_2/\overline{x}]$ for some p, $\overline{x}$, $\overline{q}_1$, $\overline{q}_2$ with $FV(p) = \overline{x}$ and $\overline{q}_1 \sim \overline{q}_2$. And whenever $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then $p[\overline{q}_2/\overline{x}] \xrightarrow{\Gamma'} p''$ with $(\Gamma, \Gamma') \in \widehat{CR^*}$ and $(p', p'') \in CR^*$. And whenever $p[\overline{q}_2/\overline{x}] \xrightarrow{\Gamma} p'$ then $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma'} p''$ with $(\Gamma, \Gamma') \in \widehat{CR^*}$ and $(p', p'') \in CR^*$. We only prove the first case, the second follows by a symmetric argument. The proposition then follows by induction on the length of the sequence $p_1 \ldots p_n$. To prove the first case above we proceed by induction on the length of the inference used to establish the transition $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ and cases of the structure of p.

$\underline{p \equiv nil}$ Trivial since $p[\overline{q}_i/\overline{x}] \not\rightarrow$.

$\underline{p \equiv a?y.p_1}$ Assume $y \notin \overline{x}$ (otherwise use α-conversion on y). Then $p[\overline{q}_i/\overline{x}] \equiv a?y.(p_1[\overline{q}_i/\overline{x}])$ and $p[\overline{q}_i/\overline{x}] \xrightarrow{a?r} (p_1[\overline{q}_i/\overline{x}])[r/y] \equiv (p_1[r/y])[\overline{q}_i/\overline{x}]$ since r, $\overline{q}_i$ are closed and $FV(p) = \overline{x}$ and $y \notin \overline{x}$. Since $FV(p_1[r/y]) = \overline{x}$ we have $((p_1[r/y])[\overline{q}_1/\overline{x}], (p_1[r/y])[\overline{q}_2/\overline{x}]) \in CR \subseteq CR^*$ and since $r \sim r$ we have $(a?(x[r/x]), a?(x[r/x])) \in \widehat{CR} \subseteq \widehat{CR^*}$.

$\underline{p \equiv a!p_1.p_2}$ From proposition 2.2.4 we have
$p[\overline{q}_i/\overline{x}] \equiv a!(p_1[\overline{q}_i/\overline{x}]).(p_2[\overline{q}_i/\overline{x}]) \equiv a!(p_1[\overline{q}_i^1/\overline{x}^1]).(p_2[\overline{q}_i^2/\overline{x}^2])$
where $\overline{x}^i = FV(p_i)$ and $\overline{q}_j^i$ is the respective projection of $\overline{q}_j$.
Then $p[\overline{q}_i/\overline{x}] \xrightarrow{a!(p_1[\overline{q}_i^1/\overline{x}^1])} (p_2[\overline{q}_i^2/\overline{x}^2])$
and $(p_2[\overline{q}_1^2/\overline{x}^2], p_2[\overline{q}_2^2/\overline{x}^2]) \in CR \subseteq CR^*$ and $(a!(p_1[\overline{q}_1^1/\overline{x}^1]), a!(p_1[\overline{q}_2^1/\overline{x}^1])) \in \widehat{CR} \subseteq \widehat{CR^*}$.

$\underline{p \equiv \tau.p_1}$ An argument similar to the argument given in the case above yields this case.

$\underline{p \equiv p_1 + p_2}$ If $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then

either $p_1[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ by a shorter inference. By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{\Gamma'} p''$ with $(\Gamma,\Gamma') \in \widehat{CR^*}$ and $(p',p'') \in CR^*$. By the operational semantics for choice we have $(p_1+p_2)[\overline{q}_2/\overline{x}] \xrightarrow{\Gamma'} p''$ which is a matching move.

or $p_2[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ and we may argue as above.

$\underline{p \equiv p_1 \mid p_2}$ If $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then

either $p_1[\overline{q}_1/\overline{x}] \equiv p_1[\overline{q}_1^1/\overline{x}^1] \xrightarrow{\Gamma} p_1'$ by a shorter inference and $p' \equiv p_1' \mid p_2[\overline{q}_1/\overline{x}] \equiv p_1' \mid p_2[\overline{q}_1^2/\overline{x}^2]$ where $\overline{x}^i = FV(p_i)$ and $\overline{q}_1^i$ is the respective projection of $\overline{q}_1$. By induction $p_1[\overline{q}_2/\overline{x}] \equiv p_1[\overline{q}_2^1/\overline{x}^1] \xrightarrow{\Gamma'} p_1''$ with $(\Gamma,\Gamma') \in \widehat{CR^*}$ and $(p_1',p_1'') \in CR^*$. By the operational semantics for parallel we have $(p_1 \mid p_2)[\overline{q}_2/\overline{x}] \equiv (p_1[\overline{q}_2/\overline{x}]) \mid (p_2[\overline{q}_2/\overline{x}]) \xrightarrow{\Gamma'} p_1'' \mid p_2[\overline{q}_2^2/\overline{x}^2]$. Since $(p_1',p_1'') \in CR^*$ there exist p_3, $\overline{q}_1^{1'}$, $\overline{q}_2^{1'}$ and $\overline{x}^{1'}$ such that $p_1' \equiv p_3[\overline{q}_1^{1'}/\overline{x}^{1'}]$ and $p_1'' \equiv p_3[\overline{q}_2^{1'}/\overline{x}^{1'}]$ with $FV(p_3) = \overline{x}^{1'}$ and $\overline{q}_1^{1'} \sim \overline{q}_2^{1'}$. We may assume $\overline{x}^{1'} \cap \overline{x}^2 = \emptyset$ since if $\overline{x}^{1'} \cap \overline{x}^2 \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup \overline{x}^{1'} \cup \overline{x}^2) = \emptyset$ and we have $p_3[\overline{q}_i^{1'}/\overline{x}^{1'}] \equiv (p_3[\overline{y}/\overline{x}^{1'}])[\overline{q}_i^{1'}/\overline{y}]$ by proposition 2.2.4. Thus $p_1' \mid p_2[\overline{q}_1^2/\overline{x}^2] \equiv (p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}^{1'} \cup \overline{x}^2]$ and $p_1'' \mid p_2[\overline{q}_2^2/\overline{x}^2] \equiv (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}^{1'} \cup \overline{x}^2]$ and $((p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}^{1'} \cup \overline{x}^2], (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}^{1'} \cup \overline{x}^2]) \in CR^*$.

or $p_2[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p_2'$ and we may argue as above.

or $\Gamma = \tau$ and w.l.o.g. $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a?r} p_1'$ and $p_2[\overline{q}_1/\overline{x}] \xrightarrow{a!r} p_2'$ by shorter inferences and $p' \equiv p_1' \mid p_2'$. By induction $p_2[\overline{q}_2/\overline{x}] \xrightarrow{a!r_2} p_2''$ with $(r,r_2) \in CR^*$ and $(p_2',p_2'') \in CR^*$ and $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?r_1} p_1''$ with $(r,r_1) \in CR^*$ and $(p_1',p_1'') \in CR^*$. By lemma 2.3.8 we have $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?r_2} p_1'''$ with $(r_1,r_2) \in CR^*$ and $(p_1'',p_1''') \in CR^*$. By the operational semantics for parallel $(p_1 \mid p_2)[\overline{q}_2/\overline{x}] \xrightarrow{\tau} p_1''' \mid p_2''$. Since $(p_1',p_1''') \in CR^*$ there exist p_3, $\overline{q}_1^1$, $\overline{q}_2^1$ and $\overline{x}^1$ such that $p_1' \equiv p_3[\overline{q}_1^1/\overline{x}^1]$ and $p_1''' \equiv p_3[\overline{q}_2^1/\overline{x}^1]$ with $FV(p_3) = \overline{x}^1$ and $\overline{q}_1^1 \sim \overline{q}_2^1$ and since $(p_2',p_2'') \in CR^*$ there exist p_4, $\overline{q}_1^2$, $\overline{q}_2^2$ and $\overline{x}^2$ such that $p_2' \equiv p_4[\overline{q}_1^2/\overline{x}^2]$ and $p_2'' \equiv p_4[\overline{q}_2^2/\overline{x}^2]$ with $FV(p_4) = \overline{x}^2$ and $\overline{q}_1^2 \sim \overline{q}_2^2$. We may assume $\overline{x}^1 \cap \overline{x}^2 = \emptyset$ since if $\overline{x}^1 \cap \overline{x}^2 \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup FV(p_4) \cup \overline{x}^1 \cup \overline{x}^2) = \emptyset$ and we have $p_3[\overline{q}_i^1/\overline{x}^1] \equiv (p_3[\overline{y}/\overline{x}^1])[\overline{q}_i^1/\overline{y}]$ by proposition 2.2.4. Therefore we have $p_1' \mid p_2' \equiv (p_3[\overline{q}_1^1/\overline{x}^1]) \mid (p_4[\overline{q}_1^2/\overline{x}^2]) \equiv (p_3 \mid p_4)[\overline{q}_1^1 \cup \overline{q}_1^2/\overline{x}^1 \cup \overline{x}^2]$ and $p_1''' \mid p_2'' \equiv (p_3[\overline{q}_2^1/\overline{x}^1]) \mid (p_4[\overline{q}_2^2/\overline{x}^2]) \equiv (p_3 \mid p_4)[\overline{q}_2^1 \cup \overline{q}_2^2/\overline{x}^1 \cup \overline{x}^2]$ and

$(p_1' \mid p_2', p_1''' \mid p_2'') \in CR^*$. (Note that if we have to introduce a "new" $\overline{y}$ it is because two or more occurrences of the same x_i refer to different q_i's after the transition.)

$\underline{p \equiv p_1\backslash b}$ If $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then

either $\Gamma = a?r$, then $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a?r} p_1'$ by a shorter inference and $p' \equiv p_1'\backslash b$ and $a \neq b$. By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?r'} p_2'$ with $(a?r, a?r') \in \widehat{CR^*}$ and $(p_1', p_2') \in CR^*$. By the operational semantics for restriction $(p_1\backslash b)[\overline{q}_2/\overline{x}] \xrightarrow{a?r'} p_2'\backslash b$. Since $(p_1', p_2') \in CR^*$ there exist p_3, $\overline{q}_1$, $\overline{q}_2$ and $\overline{x}$ such that $p_1' \equiv p_3[\overline{q}_1/\overline{x}]$ and $p_2' \equiv p_3[\overline{q}_2/\overline{x}]$ with $FV(p_3) = \overline{x}$ and $\overline{q}_1 \sim \overline{q}_2$ and $(p_1'\backslash b) \equiv (p_3\backslash b)[\overline{q}_1/\overline{x}]$ and $(p_2'\backslash b) \equiv (p_3\backslash b)[\overline{q}_2/\overline{x}]$ thus $(p_1'\backslash b, p_2'\backslash b) \in CR^*$.

or $\Gamma = a!r$ and we may argue as above.

or $\Gamma = \tau$ and we may argue as above.

$\underline{p \equiv p_1[S]}$ Similar to the case $p \equiv p_1\backslash b$.

$\underline{p \equiv y}$ By assumption $FV(p) = \overline{x}$ thus $\overline{x} = (y)$ and if $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then $q_1 \xrightarrow{\Gamma} q_1'$ and $p' = q_1'$. Since $q_1 \sim q_2$ we have $q_2 \xrightarrow{\Gamma'} q_2'$ with $(\Gamma, \Gamma') \in \approx \subseteq \widehat{CR^*}$ and $(q_1', q_2') \in \sim \subseteq CR^*$ and we have a matching move for $p[\overline{q}_2/\overline{x}]$.

□

Proposition 2.3.9
If $p \in CPr$ and $p \xrightarrow{a?r} p'$ and $r \sim r'$ then $p \xrightarrow{a?r'} p''$ with $p' \sim p''$ for some p''.

PROOF: If $r \sim r'$ then $(x[r/x], x[r'/x]) \in CR^*$ thus $(p', p'') \in CR^*$ by proposition 2.3.8 and by proposition 2.3.5 we have $p' \sim p''$. □

PROOF: (of proposition 2.3.5.2 to 2.3.5.8)

2. This is proved by showing that the following relation $R_1 = R \cup \sim$ where:

$$R = \{(a?x.p, a?x.q) \ : \ FV(p) = FV(q) \subseteq \{x\} \& \forall r.p[r/x] \sim q[r/x]\}$$

is a higher order bisimulation: Note that the above relation consists of two parts; one part covers the structure we are interested in and the second component is a kind of closure to cover the processes sent and received. The second component is necessary since the processes sent and received do not have to have the structure of the first part.
That the above relation is indeed a higher order bisimulation is easily established.

Assume $(p,q) \in R_1$. Then either $p \sim q$ and we are done since if $p \xrightarrow{\Gamma} p'$ then $q \xrightarrow{\Gamma'} q'$ for some q', Γ' with $(\Gamma,\Gamma') \in \approx \subseteq \widehat{R_1}$ and $(p',q') \sim \subseteq R_1$ or $p \equiv a?x.p'$ and $q \equiv a?x.q'$. If $a?x.p' \xrightarrow{\Gamma} p''$ then $\Gamma = a?r$ and $p'' \equiv p'[r/x]$ for some r. Then $a?x.q' \xrightarrow{a?r} q'[r/x]$ and by assumption $p'[r/x] \sim q'[r/x]$ which implies $(p'[r/x], q'[r/x]) \in R_1$. Also $(a?r, a?r) \in \widehat{Id} \subseteq \approx$ which implies $(\Gamma,\Gamma') \in \widehat{R_1}$. A symmetric argument applies if $a?x.q' \xrightarrow{\Gamma} q''$.

3. – 8. follow from

$$\begin{aligned}((a!x.y)[(p,p')/(x,y)], (a!x.y)[(q,q')/(x,y)]) &\in CR\\ ((\tau.x)[p/x], (\tau.x)[q/x]) &\in CR\\ ((x+y)[(p,p')/(x,y)], (x+y)[(q,q')/(x,y)]) &\in CR\\ ((x \mid y)[(p,p')/(x,y)], (x \mid y)[(q,q')/(x,y)]) &\in CR\\ ((x\backslash a)[p/x], (x\backslash a)[q/x]) &\in CR\\ ((x[S])[p/x], (x[S])[q/x]) &\in CR\end{aligned}$$

if $p \sim q$ and $p' \sim q'$ and $x \neq y$.

(Note that we could prove 3.-8. directly by construction of bisimulations, e.g. to prove

$$p \mid p' \sim q \mid q' \text{ if } p \sim q \text{ and } p' \sim q'$$

we show that the relation:

$$R = \{(p \mid p', q \mid q') \; : \; p \sim q \;\&\; p' \sim q'\} \cup \sim$$

is a higher order bisimulation. To see this we assume $(p,q) \in R$. Then either $p \sim q$ and if $p \xrightarrow{\Gamma} p'$ then $q \xrightarrow{\Gamma'} q'$ with $(\Gamma,\Gamma') \in \approx \subseteq \hat{R}$ and $(p',q') \in \sim \subseteq R$ and we have a matching move or $p \equiv p_1 \mid p_2$ and $q \equiv q_1 \mid q_2$. If $p_1 \mid p_2 \xrightarrow{\Gamma} p''$ then

either $p_1 \xrightarrow{\Gamma} p_1''$ and $p'' \equiv p_1'' \mid p_2$. Since $p_1 \sim q_1$ we know that $q_1 \xrightarrow{\Gamma'} q_1''$ and $(\Gamma,\Gamma') \in \approx \subseteq \hat{R}$ and $(p_1'', q_1'') \in \sim \subseteq R$. Then $q_1 \mid q_2 \xrightarrow{\Gamma'} q_1'' \mid q_2$ and $(p_1'' \mid p_2, q_1'' \mid q_2) \in R$.

or $p_2 \xrightarrow{\Gamma} p_2''$ and $p'' \equiv p_1 \mid p_2''$ and similar arguments as above apply.

or $\Gamma = \tau$ and w.l.o.g. $p_1 \xrightarrow{a?r} p_1''$ and $p_2 \xrightarrow{a!r} p_2''$ and $p'' \equiv p_1'' \mid p_2''$. Then $q_1 \xrightarrow{a?r_1} q_1''$ with $r \sim r_1$ and $p_1'' \sim q_1''$ since $p_1 \sim q_1$ and $q_2 \xrightarrow{a!r_2} q_2''$ with $r \sim r_2$ and $p_2'' \sim q_2''$ since $p_2 \sim q_2$. By proposition 2.3.9 we have $q_1 \xrightarrow{a?r_2} q_1'''$ and $q_1'' \sim q_1'''$. Then $q_1 \mid q_2 \xrightarrow{\tau} q_1''' \mid q_2''$ and $(\tau,\tau) \in \hat{R}$ and $(p_1'' \mid p_2'', q_1''' \mid q_2'') \in R$.

If $q_1 \mid q_2 \xrightarrow{\Gamma} q''$ then a symmetric argument applies.) □

The relation $\sim$ is an equivalence relation on Pr but not a congruence relation since e.g. $x \sim y$ but $a?x.x \not\sim a?x.y$. Instead of using $\sim$ directly on Pr we define $\sim$ in terms of $\sim$ on CPr and abuse the notation:

Definition 2.3.10

$$p \sim q \ \textit{iff} \ \forall \overline{r}.p[\overline{r}/\overline{x}] \sim q[\overline{r}/\overline{x}]$$

where $\overline{x} = FV(p) = FV(q)$ *and* $\overline{r}$ *is closed.*

This is equivalent to the following definition:

$$p \sim q \text{ iff } a?x_1 \ldots a?x_n.p \sim a?x_1 \ldots a?x_n.q$$

Proposition 2.3.11 $\sim$ *is a congruence relation.*

PROOF: We only need to check that if $\overline{q}_1 \sim \overline{q}_2$ then $p[\overline{q_1}/\overline{x}] \sim p[\overline{q_2}/\overline{x}]$. This is done by structural induction on p using the above definition of $\sim$ on open terms and proposition 2.3.5. We show two cases for illustration:

$\underline{p \equiv a?y.p_1}$ Assume $y \notin \overline{x}$ and $y \notin FV(\overline{q}_1) \cup FV(\overline{q}_2)$ (otherwise use α-conversion on y). By induction $p_1[\overline{q}_1/\overline{x}] \sim p_1[\overline{q}_2/\overline{x}] \equiv \forall \overline{r}.p_1[\overline{q}_1/\overline{x}][\overline{r}/\overline{y}] \sim p_1[\overline{q}_2/\overline{x}][\overline{r}/\overline{y}]$ where $\overline{y} = FV(p_1[\overline{q}_1/\overline{x})] = FV(p_1[\overline{q}_2/\overline{x})]$ and $\overline{r}$ is closed. If $y \in FV(p_1)$ then $y \in \overline{y}$, w.l.o.g. assume $y = y_n$ and let $\overline{r}' = (r_1, \ldots, r_{n-1})$ and let $\overline{y}' = (y_1, \ldots, y_{n-1})$. Thus: $\forall r.\forall \overline{r}'.p_1[\overline{q}_1/\overline{x}][\overline{r}'/\overline{y}'][r/y] \sim p_1[\overline{q}_2/\overline{x}][\overline{r}'/\overline{y}'][r/y]$ and by proposition 2.3.5.2 we have $\forall \overline{r}'.a?y.p_1[\overline{q}_1/\overline{x}][\overline{r}'/\overline{y}'] \sim a?y.p_1[\overline{q}_2/\overline{x}][\overline{r}'/\overline{y}']$ and therefore $a?y.p_1[\overline{q}_1/\overline{x}] \sim a?y.p_1[\overline{q}_2/\overline{x}]$. If $y \notin \overline{y}$ then $a?y.p_1[\overline{q}_1/\overline{x}] \sim a?y.p_1[\overline{q}_2/\overline{x}]$ follows immediately.

$\underline{p \equiv p_1 \mid p_2}$ By induction $p_i[\overline{q}_1/\overline{x}] \sim p_i[\overline{q}_2/\overline{x}] \equiv \forall \overline{r}.p_i[\overline{q}_1/\overline{x}][\overline{r}/\overline{y}_i] \sim p_i[\overline{q}_2/\overline{x}][\overline{r}/\overline{y}_i]$ where $\overline{y}_i = FV(p_i[\overline{q}_1/\overline{x}]) = FV(p_i[\overline{q}_2/\overline{x}])$ and $\overline{r}$ is closed. By proposition 2.3.5.6

$\forall \overline{r}_1.\forall \overline{r}_2.p_1[\overline{q}_1/\overline{x}][\overline{r}_1/\overline{y}_1] \mid p_2[\overline{q}_1/\overline{x}][\overline{r}_2/\overline{y}_2] \sim p_1[\overline{q}_2/\overline{x}][\overline{r}_1/\overline{y}_1] \mid p_2[\overline{q}_2/\overline{x}][\overline{r}_2/\overline{y}_2]$
$\equiv \forall \overline{r}.(p_1 \mid p_2)[\overline{q}_1/\overline{x}][\overline{r}/\overline{y}] \sim (p_1 \mid p_2)[\overline{q}_2/\overline{x}][\overline{r}/\overline{y}]$
where $\overline{y} = \overline{y}_1 \cup \overline{y}_2 = FV((p_1 \mid p_2)[\overline{q}_1/\overline{x}]) = FV((p_1 \mid p_2)[\overline{q}_2/\overline{x}])$ and $\overline{r}$ is closed.

□

Having established that $\sim$ is a congruence with respect to the process constructions in CHOCS, it is natural to consider $\sim$ as an equational theory; containing equations like: $p \mid p' \sim p' \mid p$. Of course the left hand side of this equation is a different program from the one on the right hand side, but we would expect to find their behaviour equivalent, and this is in fact what the equation expresses.

We only need to establish bisimulation proof for closed expressions since $\sim$ for open expressions is expressed in terms of closed expressions. The equational properties of $\sim$ may yield a better understanding of the underlying semantics of CHOCS and for the unexperienced user of the language it may turn out to be a helpful way of understanding the language and the interplay of its constructs. In the process algebraic framework the semantics of the ACP–language [BerKlo84] is given entirely as an equational theory in an algebraic setting. We shall not do so, but in fact $Pr/\sim$ may be considered as an algebra: e.g. $(Pr/\sim, +, nil)$ is an Abelian monoid as justified by the first three of the following equations:

Proposition 2.3.12

$$\begin{aligned} p + p' &\sim p' + p \\ p + (p' + p'') &\sim (p + p') + p'' \\ p + nil &\sim p \\ p + p &\sim p \end{aligned}$$

PROOF: This follows from showing that the following relations are higher order bisimulations:

$$\begin{aligned} R_1 &= \{(p + p', p' + p)\} \cup Id \\ R_2 &= \{(p + (p' + p''), (p + p') + p'')\} \cup Id \\ R_3 &= \{(p + nil, p)\} \cup Id \\ R_4 &= \{(p + p, p)\} \cup Id \end{aligned}$$

To see this observe that for $(r, q) \in R_i, i \in \{1, 2, 3, 4\}$ we have either $(r, q) \in Id$ and if $r \xrightarrow{\Gamma} r'$ then $r = q \xrightarrow{\Gamma} q' = r'$ with $(\Gamma, \Gamma) \in \widehat{Id} \subseteq \widehat{R_i}$ and $(r', q') \in Id \subseteq R_i$ and we have a matching move or (r, q) belongs to the first part of R_i and if $r \xrightarrow{\Gamma} r'$ then this must have been inferred by the rules for choice. Then also $q \xrightarrow{\Gamma} r'$ which is a matching move since $(\Gamma, \Gamma) \in \widehat{Id} \subseteq \widehat{R_i}$ and $(r', q') \in Id \subseteq R_i$. □

The following are some expected properties of parallel composition:

Proposition 2.3.13

$$\begin{aligned} p \mid p' &\sim p' \mid p \\ p \mid (p' \mid p'') &\sim (p \mid p') \mid p'' \\ p \mid nil &\sim p \end{aligned}$$

PROOF: This follows from showing that the following relations are higher order bisimulations:

$$\begin{aligned} R_1 &= \{(p \mid p', p' \mid p)\} \cup Id \\ R_2 &= \{(p \mid (p' \mid p''), (p \mid p') \mid p'')\} \cup Id \\ R_3 &= \{(p \mid nil, p)\} \cup Id \end{aligned}$$

$\underline{R_1}$ Assume $(p,q) \in R_1$. Then either $(p,q) \in Id \subseteq R_1$ and if $p \xrightarrow{\Gamma} p'$ we can establish a matching move for q since $p = q \xrightarrow{\Gamma} q' = p'$ with $(\Gamma,\Gamma) \in \widehat{Id} \subseteq \widehat{R_2}$ and $(p',q') \in Id \subseteq R_2$ and we are done or $p \equiv p_1 \mid p_2$ and $q \equiv p_2 \mid p_1$. If $p_1 \mid p_2 \xrightarrow{\Gamma} p'$ then this must have been inferred by the rules for parallel composition. There are four cases:

either $p_1 \xrightarrow{\Gamma} p_1'$ and $p' \equiv p_1' \mid p_2$. Then $p_2 \mid p_1 \xrightarrow{\Gamma} p_2 \mid p_1'$ which is a matching move, since $(\Gamma,\Gamma) \in \widehat{Id} \subseteq \widehat{R_1}$ and $(p_1' \mid p_2, p_2 \mid p_1') \in R_1$.

or $p_2 \xrightarrow{\Gamma} p_2'$ and $p' \equiv p_1 \mid p_2'$. Then similar arguments as above apply.

or $\Gamma = \tau$ and $p_1 \xrightarrow{a?p_1'} p_1''$ and $p_2 \xrightarrow{a!p_1'} p_2''$ and $p' \equiv p_1'' \mid p_2''$. Then $p_2 \mid p_1 \xrightarrow{\tau} p_2'' \mid p_1''$ which is a matching move, since $(\tau,\tau) \in \widehat{Id} \subseteq \widehat{R_1}$ and $(p_1'' \mid p_2'', p_2'' \mid p_1'') \in R_1$.

or $\Gamma = \tau$ and $p_1 \xrightarrow{a!p_1'} p_1''$ and $p_2 \xrightarrow{a?p_1'} p_2''$ and $p' \equiv p_1'' \mid p_2''$. Then similar arguments as above apply.

If $p_2 \mid p_1 \xrightarrow{\Gamma} p'$ then symmetric arguments apply.

$\underline{R_2}$ The argument for this case is similar to the argument for R_1 though it is necessary to apply the argument twice.

$\underline{R_3}$ Assume $(p,q) \in R_3$. Then either $(p,q) \in Id \subseteq R_3$ and if $p \xrightarrow{\Gamma} p'$ we can establish a matching move for q since $p = q \xrightarrow{\Gamma} q' = p'$ with $(\Gamma,\Gamma) \in \widehat{Id} \subseteq \widehat{R_3}$ and $(p',q') \in Id \subseteq R_3$ and we are done or $p \equiv p_1 \mid nil$ and $q \equiv p_1$. If

$p_1 \mid nil \xrightarrow{\Gamma} p'$ then this must have been inferred by the rules for parallel composition. Since $nil \not\rightarrow$ the transition must have been inferred from a transition of p_1, i.e. $p_1 \xrightarrow{\Gamma} p_1'$ and $p' \equiv p_1' \mid nil$. Clearly p_1 has a matching move and $(\Gamma, \Gamma) \in \widehat{Id} \subseteq \widehat{R_3}$ and $(p_1' \mid nil, p_1') \in R_3$. Also if $p_1 \xrightarrow{\Gamma} p_1'$ then $p_1 \mid nil \xrightarrow{\Gamma} p_1' \mid nil$ by the rules for parallel composition. Clearly this is a matching move and $(\Gamma, \Gamma) \in \widehat{Id} \subseteq \widehat{R_3}$ and $(p_1' \mid nil, p_1') \in R_3$.

□

The following proposition gives a range of properties satisfied by restriction. The order of restrictions does not matter and restriction distributes over choice. Restriction does not in general distribute over parallel composition, but we shall later show an interplay under certain conditions. The last clause shows that the processes sent are not affected by the restriction on the sending processes.

Proposition 2.3.14

$$
\begin{aligned}
p\backslash a\backslash b &\sim p\backslash b\backslash a \\
(p+p')\backslash b &\sim p\backslash b + p'\backslash b \\
(a?x.p)\backslash b &\sim \begin{cases} a?x.p\backslash b & \text{if } b \neq a \\ nil & \text{otherwise} \end{cases} \\
(a!p'.p)\backslash b &\sim \begin{cases} a!p'.p\backslash b & \text{if } b \neq a \\ nil & \text{otherwise} \end{cases}
\end{aligned}
$$

PROOF: This follows from showing that the following relations are higher order bisimulations:

$$
\begin{aligned}
R_1 &= \{(p\backslash a\backslash b, p\backslash b\backslash a)\} \cup Id \\
R_2 &= \{((p+p')\backslash b, p\backslash b + p'\backslash b)\} \cup Id \\
R_3 &= \{((a?x.p)\backslash b, a?x.(p\backslash b)) \ : \ a \neq b\} \cup \{((a?x.p)\backslash b, nil) \ : \ a = b\} \cup Id \\
R_4 &= \{((a!p'.p)\backslash b, a!p'.(p\backslash b)) \ : \ a \neq b\} \cup \{((a!p'.p)\backslash b, nil) \ : \ a = b\} \cup Id
\end{aligned}
$$

□

As for restriction renaming only affects the sending processes, not the processes sent. Renaming distributes over both choice and parallel composition. Two renamings after one another act as the renaming using their function composition, and renaming by the identity function does not affect the process.

Proposition 2.3.15

$$\begin{aligned}
(a?x.p)[S] &\sim S(a)?x.p[S] \\
(a!p'.p)[S] &\sim S(a)!p'.p[S] \\
(p+p')[S] &\sim p[S]+p'[S] \\
(p \mid p')[S] &\sim p[S] \mid p'[S] \\
p[S][S'] &\sim p[S' \circ S] \\
p[Id] &\sim p
\end{aligned}$$

PROOF: This follows from showing that the following relations are higher order bisimulations:

$$\begin{aligned}
R_1 &= \{((a?x.p)[S], S(a)?x.(p[S]))\} \cup Id \\
R_2 &= \{((a!p'.p)[S], S(a)!p'.(p[S]))\} \cup Id \\
R_3 &= \{((p+p')[S], p[S]+p'[S])\} \cup Id \\
R_4 &= \{((p \mid p')[S], p[S] \mid p'[S])\} \cup Id \\
R_5 &= \{(p[S][S'], p[S' \circ S])\} \cup Id \\
R_6 &= \{(p[Id], p)\} \cup Id
\end{aligned}$$

□

The interplay between renaming and restriction may be formulated as follows:

Proposition 2.3.16

$$(p[S])\backslash b \sim (p\backslash a_1 \ldots \backslash a_n)[S]$$

where $\{a_1 \ldots a_n\} = \{a : S(a) = b\}$ *assuming this set is finite.*

PROOF: This follows from showing that the following relation is a higher order bisimulation assuming the premisses of the proposition:

$$R = \{((p[S])\backslash b, (p\backslash a_1 \ldots \backslash a_n)[S])\} \cup Id$$

□

If S is a $1-1$ function we know that its inverse S^{-1} exists and the proposition can be rephrased as:

$$p[S]\backslash b \sim p\backslash S^{-1}(b)[S]$$

This law was incorrectly stated as a general law in [Tho89]. The mistake was pointed out to me by Sanjiva Prasad in a private communication.

We have not listed any immediate interplay between (nondeterministic) choice and parallel composition above. This is due to the fact that the two operators in general do not commute, but there is a restricted interplay between them. The following proposition is a version of the expansion theorem found in most process algebras. It states that parallel composition may be expressed as the nondeterministic choice of the sequential interleaving of the actions of the components of the parallel composition. But note that contrary to most process algebras we can not in general eliminate the parallel operator using this proposition. This is because we may introduce copying of processes in communication, we may even copy the process we are trying to eliminate. This will become clear in theorem 2.6.2.

Proposition 2.3.17 *Let* $\overline{x} = \{x_1 \ldots x_n\}$, $\overline{y} = \{y_1 \ldots y_n\}$ *and* $\overline{x} \cap \overline{y} \neq \emptyset$ *and* $\overline{x} \cap FV(q) = \emptyset$ *and* $\overline{y} \cap FV(p) = \emptyset$ *then*

$$\begin{array}{rrcl} \textit{if} & p & = & \Sigma_i a_i?x_i.p_i + \Sigma_j a_j!p'_j.p_j \\ \textit{and} & q & = & \Sigma_k b_k?y_k.q_k + \Sigma_l b_l!q'_l.q_l \\ \textit{then} & p \mid q & \sim & \Sigma_i a_i?x_i.(p_i \mid q) + \Sigma_j a_j!p'_j.(p_j \mid q) + \\ & & & \Sigma_k b_k?y_k.(p \mid q_k) + \Sigma_l b_l!q'_l.(p \mid q_l) + \\ & & & \Sigma_{(i,l)\in\{(i,l)\,:\,a_i=b_l\}}\tau.(p_i[q'_l/x_i] \mid q_l) + \\ & & & \Sigma_{(j,k)\in\{(j,k)\,:\,a_j=b_k\}}\tau.(p_j \mid q_k[p'_j/y_k]) \end{array}$$

where $\Sigma_i \Gamma_i.p_i$ *describes the sum* $\Gamma_1.p_1 + \ldots + \Gamma_n.p_n$ *when* $n > 0$ *and nil if* $n = 0$, *knowing this notation is unambiguous because of proposition 2.3.12.*

PROOF: Assume the premisses of the proposition. Let rhs denote the right hand side of the above equation. Let

$$R = \{(p \mid q, rhs)\} \cup Id$$

Then R is a higher order bisimulation. For each transition of $p \mid q$ we may find a matching transition of rhs and vice versa.
If $p \mid q \xrightarrow{\Gamma} r$ then

either $p \xrightarrow{\Gamma} p'$ and $r \equiv p' \mid q$. If $\Gamma = a_i?p''$ then $p' \equiv p_i[p''/x_i]$ for some i and $rhs \xrightarrow{\Gamma} (p_i \mid q)[p'/x_i] \equiv p_1[p''/x_i] \mid q$ which is a matching move since $x_i \notin FV(q)$.

If $\Gamma = a_j!p'_j$ then $p' \equiv p_j$ for some j and $rhs \xrightarrow{\Gamma} p_j \mid q$ which is a matching move.

or $q \xrightarrow{\Gamma} q'$ and $r \equiv p \mid q'$. Then similar arguments as above apply.

or $\Gamma = \tau$. Then

either $p \xrightarrow{a_i?q_l'} p_i[q_l'/x_i]$ and $q \xrightarrow{b_l!q_l'} q_l$ and $r \equiv p_i[q_l'/x_i] \mid q_l$ and $a_i = b_l$. Then $rhs \xrightarrow{\tau} r$ which is a matching move.

or $q \xrightarrow{b_k?p_j'} q_k[p_j'/y_k]$ and $p \xrightarrow{a_j!p_j'} p_j$ and $r \equiv p_j \mid q_k[p_j'/y_k] \mid q_k$ and $a_j = b_k$. Again $rhs \xrightarrow{\tau} r$ which is a matching move.

If $rhs \xrightarrow{\Gamma} r$ then a similar case analysis as above will yield matching moves for $p \mid q$. □

Using these laws we may prove properties about processes without directly constructing a bisimulation. This approach is often much more manageable and the two methods may be combined when convenient. In [HenMil85] and [Mil81] equations like those given above are used to prove soundness for sets of sound and complete proof systems for the finite respectively regular sublanguages of CCS. We shall not do so in this thesis since we cannot hope for a complete axiomatization of the properties of CHOCS; the reason for this will become clear in the following sections.

2.4 Sorts and CHOCS

We have deliberately chosen to refer to the set *Names* as a set of port names emphasizing that process values are to be thought of as communicated via ports. In any implementation of a system described in CHOCS it would be of great importance to know certain facts about these names as e.g. the number of different names, substitutivity of names etc. We may ascribe a sort (a set of port names) to each program. To formally define the sort of a program we need a bit of notation.

Definition 2.4.1 *q is a derivative of p if $p \longrightarrow^* q$, where $p \longrightarrow q \equiv \exists \Gamma \in Act.p \xrightarrow{\Gamma} q$ and $\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$.*

Definition 2.4.2 *For each $L \subseteq Names$, let Pr_L be the set of processes p such that for any derivative q of p, if $q \xrightarrow{a?q'} q''$ or $q \xrightarrow{a!q'} q''$ then $a \in L$. If $p \in Pr_L$ we say p has sort L (written $p :: L$).*

We may see how the process constructions of CHOCS act on sorts:

Proposition 2.4.3 *Assume p, p' are closed expressions, then*

1. *If $p :: L$ and $L \subseteq M$ then $p :: M$.*

2. *If* $a \in L$ *and* $p[r/x] :: L$ *for all* r *then* $a?x.p :: L$.

3. *If* $a \in L$ *and* $p :: L$ *then* $a!p'.p :: L$ *for any* p'.

4. *If* $p :: L$ *then* $\tau.p :: L$.

5. *If* $p :: L$ *and* $p' :: L$ *then* $p + p' :: L$.

6. *If* $p :: L$ *and* $p' :: L$ *then* $p \mid p' :: L$.

7. *If* $p :: L$ *then* $p\backslash a :: L \setminus \{a\}$.

8. *If* $p :: L$ *then* $p[S] :: \{S(a) : a \in L\}$.

PROOF:

1. Follows directly from definition 2.4.2.

2. Any derivative of $a?x.p$ is either $p[r/x]$ for some r due to $a?x.p \xrightarrow{a?r} p[r/x]$ or it is a derivative of $p[r/x]$.

3. Any derivative of $a!p'.p$ is either p due to $a!p'.p \xrightarrow{a!p'} p$ or it is a derivative of p.

4. Any derivative of $\tau.p$ is either p due to $\tau.p \xrightarrow{\tau} p$ or it is a derivative of p.

5. Any derivative of $p + p'$ is either a derivative of p or a derivative of p'.

6. Any derivative of $p \mid p'$ has the form $q \mid q'$ where q is a derivative of p and q' is a derivative of p'.

7. Any derivative p' of $p\backslash a$ is a derivative p'' of p such that if $p'' \xrightarrow{\Gamma} p'''$ then if $\Gamma = b?q$ or $\Gamma = b!q$ then $b \neq a$ and $p' \equiv p''\backslash a$.

8. Any derivative p' of $p[S]$ is a derivative p'' of p such that if $p'' \xrightarrow{\Gamma} p'''$ then $p' \equiv p''[S]$ and $p''[S] \xrightarrow{S(\Gamma)} p'''[S]$.

□

The following semantic function may be used to compute the sort of a process:

Definition 2.4.4 *dynamicsort* : *CHOCS* → *Names*

$$dynamicsort(p) = \{a \in Names : \exists q, q', q''.p \longrightarrow^* q \xrightarrow{a?q'} q'' \text{ or } p \longrightarrow^* q \xrightarrow{a!q'} q''\}$$

This set is the minimal sort for an agent. The dynamic sort is often inconvenient. We are often satisfied by coarser — but easier to compute — information which may be extracted from the program text.

Definition 2.4.5 *We define staticsort : CHOCS → Names structurally on p:*

$$\begin{aligned}
staticsort(nil) &= \emptyset \\
staticsort(a?x.p) &= \{a\} \cup staticsort(p) \\
staticsort(a!p'.p) &= \{a\} \cup staticsort(p) \\
staticsort(\tau.p) &= staticsort(p) \\
staticsort(p + p') &= staticsort(p) \cup staticsort(p') \\
staticsort(p \mid p') &= staticsort(p) \cup staticsort(p') \\
staticsort(p\backslash a) &= staticsort(p) \setminus \{a\} \\
staticsort(p[S]) &= \{S(a) \,:\, a \in staticsort(p)\} \\
staticsort(x) &= Names
\end{aligned}$$

Note how we need to "assume the worst" when encountering a variable. This is because we do not know the sort of the processes which may be substituted for the variable. In fact $a?x.x \xrightarrow{a?p} p$ for any p, and p may have any sort which is reflected both in the dynamic sort and the static sort of $a?x.x$. This "assuming the worst" resembles how static approximations of dynamic properties of sequential programming languages are made in the framework of abstract interpretation [CouCou79]. "Assuming the worst" in case of a variable implies that it is not necessary to calculate the static sort of process values as may be seen from the clause for output prefix, the sort of any process received in communication will be covered by the static assumption on variables.

The dynamic sort and the static sort are of course related:

Proposition 2.4.6 $dynamicsort \subseteq staticsort$

In general we cannot hope to show that $dynamicsort = staticsort$ since this is undecidable, even without process passing, as a consequence of [AusBou84]. But both *dynamicsort* and *staticsort* are sound with respect to definition 2.4.2 of a sort for p.

We may now give some equational properties which only hold under certain constraints on the sort.

Proposition 2.4.7

1. $p\backslash b \sim p$ *if* $p :: L$ *and* $b \notin L$
2. $(p \mid p')\backslash b \sim p\backslash b \mid p'\backslash b$ *if* $p :: L$, $p' :: M$ *and* $b \notin L \cap M$
3. $p\backslash b \sim p[c/b]\backslash c$ *if* $p :: L$ *and* $c \notin L$

1. shows that restriction has no effect if the restricted port does not belong to the sort of the agent. 2. shows that restriction only distributes over communication if the restriction does not involve the ports which the processes are able to communicate via. 3. shows that the name of a restricted port is not essential up to renaming. This property corresponds to the notion of α–convertibility in [EngNie86].

PROOF: The proposition is proved by showing that the following relations are higher order bisimulations:

$$\begin{aligned} R_1 &= \{(p\backslash b, p) : p :: L \;\&\; b \notin L\} \cup Id \\ R_2 &= \{((p_1 \mid p_2)\backslash b, p_1\backslash b \mid p_2\backslash b) : p_1 :: L \;\&\; p_2 :: M \;\&\; b \notin L \cap M\} \cup Id \\ R_3 &= \{(p\backslash b, p[c/b]\backslash c) : p :: L \;\&\; c \notin L\} \cup Id \end{aligned}$$

That the above relations are higher order bisimulations is easily established by the following arguments:

$\underline{R_1}$ Assume $(p,q) \in R_1$. Then either $p = q$ and since $Id \subseteq R_1$ we are done or $p \equiv p_1\backslash b$ and $q \equiv p_1$. If $p_1\backslash b \xrightarrow{\Gamma} p''$ then this has been inferred by the rules for restriction and $p_1 \xrightarrow{\Gamma} p_1''$ and $p'' \equiv p_1''\backslash b$ and if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq b$. We have thus established a matching move. If $p_1 \xrightarrow{\Gamma} p_1''$ then if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq b$ since $b \notin L$ and $p_1 :: L$. By the rules for restriction we have $p_1 \xrightarrow{\Gamma} p_1''\backslash b$ and we have established a matching move.

$\underline{R_2}$ Assume $(p,q) \in R_2$. Then either $p = q$ and since $Id \subseteq R_2$ we are done or $p \equiv (p_1 \mid p_2)\backslash b$ and $q \equiv p_1\backslash b \mid p_2\backslash b$. If $(p_1 \mid p_2)\backslash b \xrightarrow{\Gamma} p'$ then this has been inferred by the rules for restriction and $p_1 \mid p_2 \xrightarrow{\Gamma} p''$ with $p' \equiv p''\backslash b$ and if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq b$. The transition $p_1 \mid p_2 \xrightarrow{\Gamma} p''$ must have been inferred by the rules for parallel composition and

either $p_1 \xrightarrow{\Gamma} p_1''$ and $p'' \equiv p_1'' \mid p_2$. Then $p_1\backslash b \mid p_2\backslash b \xrightarrow{\Gamma} p_1''\backslash b \mid p_2\backslash b$ and we have a matching move.

or $p_2 \xrightarrow{\Gamma} p_2''$ and $p'' \equiv p_1 \mid p_2''$ and we may argue as above.

or $\Gamma = \tau$ and $p_1 \xrightarrow{\Gamma'} p_1''$ and $p_2 \xrightarrow{\overline{\Gamma'}} p_2''$ and $p'' \equiv p_1'' \mid p_2''$. Assume w.l.o.g. $\Gamma' = a?r$, then $a \neq b$ since $a \in L \cap M$. Then $p_1\backslash b \xrightarrow{\Gamma'} p_1''\backslash b$ and $p_2\backslash b \xrightarrow{\overline{\Gamma'}} p_2''\backslash b$. From the rule for parallel composition we have $p_1\backslash b \mid p_2\backslash b \xrightarrow{\tau} p_1''\backslash b \mid p_2''\backslash b$ and we have a matching move.

If $p_1\backslash b \mid p_2\backslash b \xrightarrow{\Gamma} p''$ then

either $p_1\backslash b \xrightarrow{\Gamma} p_1''$ and $p'' \equiv p_1'' \mid p_2$ and this is because $p_1 \xrightarrow{\Gamma} p_1'$ and if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq b$ and $p_1'' \equiv p_1'\backslash b$. Then $(p_1 \mid p_2)\backslash b \xrightarrow{\Gamma} (p_1'' \mid p_2)\backslash b$ which is a matching move.

or $p_2\backslash b \xrightarrow{\Gamma} p_2''$ and we may argue as above.

or $\Gamma = \tau$ and $p_1\backslash b \xrightarrow{\Gamma'} p_1''$ and $p_2\backslash b \xrightarrow{\overline{\Gamma'}} p_2''$ and $p'' \equiv p_1'' \mid p_2''$ and this is because $p_1 \xrightarrow{\Gamma'} p_1'$ and if $\Gamma' = a?r$ or $\Gamma' = a!r$ then $a \neq b$ and $p_1'' \equiv p_1'\backslash b$ and $p_2 \xrightarrow{\overline{\Gamma'}} p_2'$ and if $\Gamma' = a?r$ or $\Gamma' = a!r$ then $a \neq b$ and $p_2'' \equiv p_2'\backslash b$. Then by the rules for parallel composition $(p_1 \mid p_2)\backslash b \xrightarrow{\Gamma} (p_1'' \mid p_2'')\backslash b$ which is a matching move.

$\underline{R_3}$ Assume $(p,q) \in R_3$. Then either $p = q$ and since $Id \subseteq R_3$ we are done or $p \equiv p_1\backslash b$ and $q \equiv p_1[c/b]\backslash c$. If $p_1\backslash b \xrightarrow{\Gamma} p''$ then this has been inferred by the rules for restriction and $p_1 \xrightarrow{\Gamma} p_1''$ and $p'' \equiv p_1''\backslash b$ and if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq b$. By the rules for renaming $p_1[c/b] \xrightarrow{\Gamma} p_1''[c/b]$ and by the rules for restriction $p_1[c/b]\backslash c \xrightarrow{\Gamma} p_1''[c/b]\backslash c$ and we have established a matching move. If $p_1[c/b]\backslash c \xrightarrow{\Gamma} p_1''[c/b]\backslash c$ then $p_1[c/b] \xrightarrow{\Gamma} p_1''[c/b]$ and if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq c$. Then $p_1 \xrightarrow{\Gamma} p_1''$ and if $\Gamma = a?r$ or $\Gamma = a!r$ then $a \neq b$ since $c \notin L$ and $p_1 :: L$. By the rules for restriction we have $p_1\backslash b \xrightarrow{\Gamma} p_1''\backslash b$ and we have established a matching move.

□

Sorted CHOCS

The information given by the static sort of definition 2.4.5 is often too coarse as in $p = (a?x.x \mid a!(b!nil.nil).nil)\backslash a$ since $staticsort(p) = Names \setminus \{a\}$, whereas $dynamicsort(p) = \{b\}$. As a solution to this problem one could introduce a sort declaration on each binding of variables and limit communication to processes of the prescribed sort. This would correspond to type declarations in typed programming languages like PASCAL. This is indeed the approach of [Nie89] where a type system including sorts of processes is presented for the language TPL. In this system the sort of processes sent and received contributes to the calculation of the sort of processes. The language TPL is a merge between the typed λ–Calculus and a CHOCS-like process language with processes passing. If we leave out the non-process expressions we may degenerate TPL to a version of CHOCS (Sorted CHOCS) with the following abstract syntax:

$$s ::= a?s \mid a!s \mid s \cup s' \mid \emptyset$$

$$p ::= nil \mid a?x \in s.p \mid a!p'.p \mid \tau.p \mid p + p' \mid p \mid p' \mid p\backslash a \mid p[S] \mid x$$

where $a \in Names$, $S : Names \to Names$ and $x \in V$.

We use the term sort instead of type for the degenerated types described by the first syntactic category of the above syntax. The sort expresses the communication possibilities of processes but not their causality. We may think of $\emptyset$ as the empty sort and it plays the double rôle of acting as $\perp$ and NIL of [Nie89]. The process part of Sorted CHOCS given by the second syntactic category of the above syntax is given an operational semantics defined by the rules for (unsorted) CHOCS from definition 2.2.5. The process language differs slightly from the process part of TPL. In TPL communication takes place using a sort dependent parallel operator in the style of [Hoa85], but $p \mid p'$ can be interpreted as $p \mid Names \mid p'$. The choice operator $+$ differs from the `or` operator of [Nie89] since the first of the following operational rules applies to $p + p'$ and the second applies to p `or` p':

$$\frac{p \xrightarrow{\tau} p''}{p + p' \xrightarrow{\tau} p''} \qquad \frac{p \xrightarrow{\tau} p''}{p \texttt{ or } p' \xrightarrow{\tau} p'' \texttt{ or } p'}$$

We now present an inference system for inferring when a process p has sort s. We have to express sorts of open expressions and in general we have statements like $\rho \vdash p : s$ which can be read as process p has sort s in the sort environment ρ. Here $\rho : V \to Sort$ is a function from variables to sorts. We write $\rho[s/x]$ for the environment which acts like ρ except on x where it returns s. Let $\rho_\emptyset$ describe the environment such that $\rho_\emptyset(x) = \emptyset$ for all x.

---

subsort $\vdash \emptyset \subseteq s$ $\quad \vdash s \subseteq s$ $\quad \vdash s_1 \subseteq s_1 \cup s_2$ $\quad \vdash s_2 \subseteq s_1 \cup s_2$

$$\frac{\vdash s_1 \subseteq s_2 \quad \vdash s_2 \subseteq s_3}{\vdash s_1 \subseteq s_3} \qquad \frac{\vdash s_1 \subseteq s \quad \vdash s_2 \subseteq s}{\vdash s_1 \cup s_2 \subseteq s} \qquad \frac{\vdash s \subseteq s' \quad \vdash s' \subseteq s}{\vdash s = s'}$$

$$\frac{\vdash s \subseteq s'}{\vdash a!s \subseteq a!s'} \qquad \frac{\vdash s \subseteq s'}{\vdash a?s' \subseteq a?s}$$

$$\text{non-structural} \quad \frac{\rho \vdash p : s \quad \vdash s \subseteq s'}{\rho \vdash p : s'}$$

$$\text{variables} \quad \rho \vdash x : s \text{ if } \rho(x) = s$$

$$\text{nil} \quad \rho \vdash nil : \emptyset$$

$$\text{input} \quad \frac{\rho[s/x] \vdash p : s'}{\rho \vdash a?x \in s.p : a?s \cup s'}$$

$$\text{output} \quad \frac{\rho \vdash p' : s' \quad \rho \vdash p : s}{\rho \vdash a!p'.p : a!s' \cup s}$$

$$\text{tau} \quad \frac{\rho \vdash p : s}{\rho \vdash \tau.p : s}$$

$$\text{choice} \quad \frac{\rho \vdash p : s \quad \rho \vdash p' : s'}{\rho \vdash p + p' : s \cup s'}$$

$$\text{restriction} \quad \frac{\rho \vdash p : s}{\rho \vdash p\backslash a : s\backslash a} \quad \text{where } s\backslash a = \begin{cases} b?s' & \text{if } s = b?s' \text{ and } a \neq b \\ b!s' & \text{if } s = b!s' \text{ and } a \neq b \\ \emptyset & \text{if } s = b?s' \text{ and } a = b \\ \emptyset & \text{if } s = b!s' \text{ and } a = b \\ \emptyset & \text{if } s = \emptyset \\ s'\backslash a \cup s''\backslash a & \text{if } s = s' \cup s'' \end{cases}$$

$$\text{renaming} \quad \frac{\rho \vdash p : s}{\rho \vdash p[S] : s[S]} \quad \text{where } s[S] = \begin{cases} S(a)?s' & \text{if } s = a?s' \\ S(a)!s' & \text{if } s = a!s' \\ \emptyset & \text{if } s = \emptyset \\ s'[S] \cup s''[S] & \text{if } s = s' \cup s'' \end{cases}$$

$$\text{parallel} \quad \frac{\rho \vdash p : s \quad \rho \vdash p' : s'}{\rho \vdash p \mid p' : s \cup s'} \quad , \; IOmatch\,(s, s')$$

Table 2.4.1: Sort system for CHOCS

The relation $\vdash s \subseteq s'$ facilitates reasoning about subsorts. Intuitively $\vdash s \subseteq s'$ says that s' allows more communication possibilities than s. Note that $\subseteq$ is a quasi ordering and $\cup$ is the least upper bound, with $\emptyset$ as the least sort. In the rule for $a?s$ the ordering between types is switched. Intuitively this says that a process allows more communications the less we assume about its input (see [Nie89] for further discussion).

In [Nie89] the process *nil* is given its own sort (type) NIL. Using the sort inference rules we may establish that $\rho \vdash nil : s$ for any s such that $\vdash NIL \subseteq s$. To a certain extent this follows ideas from [Hoa85] where each *nil* (or $STOP$) process

has to have its sort declared, i.e. $STOP_A$. We have ascribed nil with the sort $\emptyset$ instead of giving nil its own sort. The reason for this is that we want to keep in line with ideas about sorts from [Mil80, Mil83, Mil89] where nil can have any sort. We obtain this by $\vdash \emptyset \subseteq s$ for any s and $\rho \vdash nil : \emptyset$ and the non-structural rule.

The predicate $IOmatch\ (s_1, s_2)$ is intended to express that the output types of p are compatible with the input types of p' and vice versa. To formalize this we use the following definition of [Nie89]:

Definition 2.4.8 *Let* $d =!$ *or* $d =?$ *then*

$$P_{ad} : Sort \rightarrow P(Sort)$$

$$\begin{aligned} P_{ad}(a'd's) &= \begin{cases} \{s\} & \text{if } ad = a'd' \\ \emptyset & \text{otherwise} \end{cases} \\ P_{ad}(s \cup s') &= P_{ad}(s) \cup P_{ad}(s') \\ P_{ad}(\emptyset) &= \emptyset \end{aligned}$$

Intuitively $P_{ad}(s)$ is the set of communication possibilities that s allows over channel a in direction d.

Using this definition we may define $IOmatch\ (s_1, s_2)$ as:

Definition 2.4.9

$$\begin{aligned} IOmatch\ (s_1, s_2) \iff & \forall j \in 1,2. \forall a \in\ Chan\ (s_1) \cap\ Chan\ (s_2). \\ & \forall s'_j \in P_{a?}(s_j). \forall s'_{3-j} \in P_{a!}(s_{3-j}). \vdash s'_1 = s'_2 \end{aligned}$$

$$\text{where } Chan\ (s) = \begin{cases} \{a\} & \text{if } s = a?s' \text{ or } s = a!s' \\ \emptyset & \text{if } s = \emptyset \\ Chan\ (s') \cup\ Chan\ (s'') & \text{if } s = s' \cup s'' \end{cases}$$

This definition differs slightly from the definition of $IOmatch$ in [Nie89] which would become

$$\begin{aligned} IOmatch\ (s_1, s_2) \iff & \forall j \in 1,2. \forall a \in Names. \forall s'_j \in P_{a?}(s_j). \\ & \forall s'_{3-j} \in P_{a!}(s_{3-j}). \vdash s'_1 = s'_2 \end{aligned}$$

If $Names$ is infinite an infinite number of conditions is introduced in the inference system whereas $IOmatch$ from definition 2.4.9 only needs to check a finite number of conditions.

We may use the sort information provided by the sort inference system to infer properties of processes dependent on the sort information. To do this we need to relate the sort information to the underlying (un-sorted) operational semantics:

Theorem 2.4.10 *(Theorem 5.1 of [Nie89]).*
Let $\rho_\emptyset \vdash p : s$ *and* $p \xrightarrow{\Gamma} p''$

(1) *if* $\Gamma = \tau$ *then* $\rho_\emptyset \vdash p'' : s$

(2) *if* $\Gamma = a!p'$ *then* $a \in Chan\,(s)$ *and* $(\rho_\emptyset \vdash p'' : s)$ *and* $(\exists s' \in P_{a!}(s).\rho_\emptyset \vdash p' : s')$

(3) *if* $\Gamma = a?p'$ *then* $a \in Chan\,(s)$ *and* $P_{a?}(s) \neq \emptyset$ *and*
$(\forall s' \in P_{a?}(s).\rho_\emptyset \vdash p' : s') \implies (\rho_\emptyset \vdash p'' : s)$

Corollary 2.4.11 *If* $\rho_\emptyset \vdash p : s$ *and* $\forall s' \in P_{a?}(s).\rho_\emptyset \vdash q' : s'$ *for all derivatives* q *of* p *such that* $q \xrightarrow{a?q'} q''$ *then* $p :: Chan\,(s)$.

PROOF: By definition 2.4.9 and theorem 2.4.10 □

Intuitively corollary 2.4.11 states that if we restrict ourselves to supplying processes which have sort declared in s when p needs an input then p's computations will only use the channels in s.

The proof of Theorem 2.4.10 follows the proof of Theorem 5.1 of [Nie89] quite closely except for a few details due to the difference between operators in TPL and in CHOCS and the fact that we use functions ρ for sort environments whereas [Nie89] uses ordered lists.

Before presenting a proof of theorem 2.4.10 we need the following lemmas:

Lemma 2.4.12 *(Lemma 5.2 of [Nie89]).*
In a deduction of $\rho \vdash p : s$ *we may assume that the non-structural rule is used after every structural rule and axiom and nowhere else.*

PROOF: If we look at the proof tree for $\rho \vdash p : s$ we can transform it into a new proof tree where we use the non-structural rule after each structural rule since $\vdash s \subseteq s$ holds for all s. In this new proof tree there might be applications of the non-structural rule followed by applications of the non-structural rule but we can eliminate these using the transitivity rule $\dfrac{\vdash s_1 \subseteq s_2 \quad \vdash s_2 \subseteq s_3}{\vdash s_1 \subseteq s_3}$ by replacing each double application of the non-structural rule: $\dfrac{\rho \vdash p : s_1 \quad \vdash s_1 \subseteq s_2}{\rho \vdash p : s_2}$ followed by $\dfrac{\rho \vdash p : s_2 \quad \vdash s_2 \subseteq s_3}{\rho \vdash p : s_3}$ with the single application of the non-structural rule:

$$\frac{\rho \vdash p : s_1 \quad \vdash s_1 \subseteq s_3}{\rho \vdash p : s_3}$$ □

Lemma 2.4.13 *(Lemma 5.3 of [Nie89]).*
if $\vdash s_1 \subseteq s_2$ *then*

$$\forall s_1' \in P_{a!}(s_1).\exists s_2' \in P_{a!}(s_2).\ \vdash s_1' \subseteq s_2'$$
$$\forall s_1' \in P_{a?}(s_1).\exists s_2' \in P_{a?}(s_2).\ \vdash s_2' \subseteq s_1'$$

PROOF: By induction on the structure of the inference $\vdash s_1' \subseteq s_2'$. For the axioms $\vdash \emptyset \subseteq s$, $\vdash s \subseteq s$, $\vdash s_1 \subseteq s_1 \cup s_2$ and $\vdash s_2 \subseteq s_1 \cup s_2$ the result follows from $P_{ad}(\emptyset) = \emptyset$ and $P_{ad}(s_1 \cup s_2) = P_{ad}(s_1) \cup P_{ad}(s_2)$. For the rules $\dfrac{\vdash s \subseteq s'}{\vdash a!s \subseteq a!s'}$ and $\dfrac{\vdash s \subseteq s'}{\vdash a?s' \subseteq a?s}$ the result follows from the definition of $P_{ad}(a'd's')$. □

Lemma 2.4.14 *if* $\rho[s'/x] \vdash p : s$ *and* $\rho \vdash p' : s'$ *then* $\rho \vdash p[p'/x] : s$

PROOF: By induction on the length of the inference $\rho[s'/x] \vdash p : s$ and cases of the structure of p with the use of lemma 2.4.12.

We shall use the fact if $x \notin FV(p)$ then $\rho[s'/x] \vdash p : s$ iff $\rho \vdash p : s$ which is easily established by an argument by induction.

Assume $\rho[s'/x] \vdash p : s$ and $\rho \vdash p' : s'$

$\underline{p \equiv nil}$ Trivial since $x \notin FV(nil)$ and $nil[p'/x] \equiv nil$ we have $\rho \vdash nil[p'/x] : s$.

$\underline{p \equiv y}$ If $x \neq y$ then $y[p'/x] \equiv y$ and $x \notin FV(y)$ thus $\rho \vdash y[p'/x] : s$.
If $x = y$ then $y[p'/x] \equiv p'$. Since $\rho[s'/x] \vdash y : s$ we must have $\vdash s' \subseteq s$. By the non-structural rule and $\rho \vdash p' : s'$ we have $\rho \vdash y[p'/x] : s$

$\underline{p \equiv a?y \in s_1.p_1}$ If $x = y$ then $x \notin FV(a?y \in s_1.p_1)$ and $(a?y \in s_1.p_1)[p'/x] \equiv a?y \in s_1.p_1$ and we have $\rho \vdash (a?y \in s_1.p_1)[p'/x] : s$.
If $x \neq y$ we may assume $y \notin FV(p')$ otherwise $(a?y \in s_1.p_1)[p'/x] \equiv a?z \in s_1.((p_1[z/x])[p'/x])$ where $z \neq x$ and $z \notin FV(p_1) \cup FV(p')$ and we may show $\rho \vdash a?y \in s_1.p_1 : s$ iff $\rho \vdash a?z \in s_1.p_1[z/y] : s$ and we will have to argue on $\rho \vdash a?z \in s_1.p_1[z/y] : s$.
By lemma 2.4.12 there exists a sort s_2 such that $(\rho[s'/x])[s_1/y] \vdash p_1 : s_2$ and $\vdash a?s_1 \cup s_2 \subseteq s$. Since $(\rho[s'/x])[s_1/y] = (\rho[s_1/y])[s'/x]$ if $x \neq y$ we have $(\rho[s_1/y])[s'/x] \vdash p_1 : s_2$. By induction (on $(\rho[s_1/y]) \vdash p : s$) we have

$(\rho[s_1/y]) \vdash p_1[p'/x] : s_2$. Since $x \neq y$ and $y \notin FV(p')$ we have $(a?y \in s_1.p_1)[p'/x] \equiv a?y \in s_1.(p_1[p'/x])$ and we have $\rho \vdash (a?y \in s_1.p_1)[p'/x] : a!s_1 \cup s_2$. By $\vdash a?s_1 \cup s_2 \subseteq s$ and the non-structural rule we have $\rho \vdash (a?y \in s_1.p_1)[p'/x] : s$.

The rest of the cases follow straightforwardly by induction and use of lemma 2.4.12. We give one case for illustration:

$\underline{p \equiv p_1 \mid p_2}$ By lemma 2.4.12 there exist sorts s_1 and s_2 such that $\rho[s'/x] \vdash p_1 : s_1$ and $\rho[s'/x] \vdash p_2 : s_2$ and $IOmatch\ (s_1, s_2)$ and $\vdash s_1 \cup s_2 \subseteq s$. By induction $\rho \vdash p_1[p'/x] : s_1$ and $\rho \vdash p_2[p'/x] : s_2$. Since $p_1[p'/x] \mid p_2[p'/x] \equiv (p_1 \mid p_2)[p'/x] : s_2$ we have $\rho \vdash (p_1 \mid p_2)[p'/x] : s_1 \cup s_2$ and by the non-structural rule $\rho \vdash (p_1 \mid p_2)[p'/x] : s$.

□

With this machinery in hand we now prove theorem 2.4.10.

PROOF: Assume $\rho \vdash p : s$ and $p \xrightarrow{\Gamma} p'$. We proceed by induction on the length of the inference used to establish $p \xrightarrow{\Gamma} p'$. Consider the possible forms of p:

$\underline{p \equiv nil}$ Trivial since $nil \not\rightarrow$.

$\underline{p \equiv a?x \in s_1.p_1}$ Then $a?x \in s_1.p_1 \xrightarrow{a?p'} p_1[p'/x]$. By lemma 2.4.12 there exists a sort s_2 such that $\rho_\emptyset\,[s_1/x] \vdash p_1 : s_2$ and $\vdash (a?s_1) \cup s_2 \subseteq s$. Thus we have $a \in Chan\ (s)$. Assume $\rho_\emptyset \vdash p' : s'$ for all $s' \in P_{a?}(s)$ then by lemma 2.4.13 we know $\rho_\emptyset \vdash p' : s_1$ holds. Using lemma 2.4.14 we get $\rho_\emptyset \vdash p_1[p'/x] : s_2$ and by the non-structural rule we have $\rho_\emptyset \vdash p_1[p'/x] : s$. Also $P_{a?}(s) \neq \emptyset$ holds.

$\underline{p \equiv a!p_1.p_2}$ Then $a!p_1.p_2 \xrightarrow{a!p_1} p_2$. By lemma 2.4.12 there exist sorts s_1 and s_2 such that $\rho_\emptyset \vdash p_1 : s_1$ and $\rho_\emptyset \vdash p_2 : s_2$ and $\vdash (a!s_1) \cup s_2 \subseteq s$. Thus we have $a \in Chan\ (s)$ and by the non-structural rule we have $\rho_\emptyset \vdash p_2 : s$. By lemma 2.4.13 we have $\exists s' \in P_{a!}(s).\rho_\emptyset \vdash p_1 : s'$ namely $s' = s_1$.

$\underline{p \equiv \tau.p_1}$ Then $\tau.p_1 \xrightarrow{\tau} p_1$. By lemma 2.4.12 there exists a sort s_1 such that $\rho_\emptyset \vdash p_1 : s_1$ and $\vdash s_1 \subseteq s$. By the non-structural rule we have $\rho_\emptyset \vdash p_1 : s$.

$\underline{p \equiv p_1 + p_2}$ If $p_1 + p_2 \xrightarrow{\Gamma} p'$ then either $p_1 \xrightarrow{\Gamma} p'$ or $p_2 \xrightarrow{\Gamma} p'$ by a shorter inference. We consider the case where $p_1 \xrightarrow{\Gamma} p'$, the other case is similar. By lemma 2.4.12 there exist sorts s_1 and s_2 such that $\rho_\emptyset \vdash p_1 : s_1$ and $\rho_\emptyset \vdash p_2 : s_2$ and $\vdash s_1 \cup s_2 \subseteq s$. We have the following possible forms of Γ:

$\underline{\Gamma \equiv \tau}$ By induction $\rho_\emptyset \vdash p' : s_1$ and by the non-structural rule we have $\rho_\emptyset \vdash p' : s$.

$\underline{\Gamma = a!p''}$ By induction $a \in Chan\,(s_1)$ and $\rho_\emptyset \vdash p' : s_1$ and $\rho_\emptyset \vdash p'' : s''$ for some $s'' \in P_{a!}(s_1)$. By the non-structural rule we have $\rho_\emptyset \vdash p' : s$ and by lemma 2.4.13 we have $\rho_\emptyset \vdash p'' : s''$ for some $s'' \in P_{a!}(s)$ and $a \in Chan\,(s)$.

$\underline{\Gamma = a?p''}$ Assume $\rho_\emptyset \vdash p'' : s''$ for all $s'' \in P_{a?}(s_1)$. By lemma 2.4.13 we have $\rho_\emptyset \vdash p'' : s''$ for all $s'' \in P_{a?}(s)$. By induction $a \in Chan\,(s_1)$ and $\rho_\emptyset \vdash p' : s_1$ and $P_{a?}(s_1) \neq \emptyset$. By the non-structural rule we have $\rho_\emptyset \vdash p' : s$ and by lemma 2.4.13 we have $P_{a?}(s) \neq \emptyset$ and $a \in Chan\,(s)$.

$\underline{p \equiv p_1 \mid p_2}$ By lemma 2.4.12 there exist sorts s_1 and s_2 such that $\rho_\emptyset \vdash p_1 : s_1$ and $\rho_\emptyset \vdash p_2 : s_2$ and $IOmatch\,(s_1, s_2)$ and $\vdash s_1 \cup s_2 \subseteq s$. If $p_1 \mid p_2 \xrightarrow{\Gamma} p_1' \mid p_2$ then

either $p_1 \xrightarrow{\Gamma} p_1'$ by a shorter inference and $p' \equiv p_1' \mid p_2$. An argument similar to the one given for $p \equiv p_1 + p_2$ applies.

or $p_2 \xrightarrow{\Gamma} p_2'$ by a shorter inference and $p' \equiv p_1 \mid p_2'$. An argument similar to the one given for $p \equiv p_1 + p_2$ applies.

or $p_1 \xrightarrow{\Gamma} p_1'$ and $p_2 \xrightarrow{\overline{\Gamma}} p_2'$ by shorter inferences and $p' \equiv p_1' \mid p_2'$. Assume w.l.o.g. that $\Gamma = a?r$. By induction $a \in Chan\,(s_2)$ and $\rho_\emptyset \vdash p_2' : s_2$ and $\rho_\emptyset \vdash r : s_2'$ for some $s_2' \in P_{a!}(s_2)$. From $IOmatch\,(s_1, s_2)$ we have whenever $s_1' \in P_{a?}(s_1)$ then $\rho_\emptyset \vdash r : s_1'$. By induction $a \in Chan\,(s_1)$ and $\rho_\emptyset \vdash p_1' : s_1$ and $P_{a?}(s_1) \neq \emptyset$. It follows that $\rho_\emptyset \vdash p_1' \mid p_2' : s_1 \cup s_2$. By the non-structural rule we have $\rho_\emptyset \vdash p_1' \mid p_2' : s$.

$\underline{p \equiv p_1 \backslash b}$ By lemma 2.4.12 there exists a sort s_1 such that $\rho_\emptyset \vdash p_1 : s_1$ and $\vdash s_1 \backslash b \subseteq s$. If $p_1 \backslash b \xrightarrow{\Gamma} p'$ then $p_1 \xrightarrow{\Gamma} p_1'$ by a shorter inference and $p' \equiv p_1 \backslash b$. We have the following possible forms of Γ:

$\underline{\Gamma \equiv \tau}$ By induction $\rho_\emptyset \vdash p_1' : s_1$ and $\rho_\emptyset \vdash p_1' \backslash b : s_1 \backslash b$. By the non-structural rule we have $\rho_\emptyset \vdash p_1' \backslash b : s$.

$\underline{\Gamma = a!p''}$ Then $a \neq b$. By induction $a \in Chan\,(s_1)$ and $\rho_\emptyset \vdash p_1' : s_1$ and $\rho_\emptyset \vdash p'' : s''$ for some $s'' \in P_{a!}(s_1)$. Since $a \neq b$ we have $a \in Chan\,(s_1 \backslash b)$ and $P_{a!}(s_1) = P_{a!}(s_1 \backslash b)$. From $\rho_\emptyset \vdash p_1' \backslash b : s_1 \backslash b$ and the non-structural rule we have $\rho_\emptyset \vdash p_1' : s$ and by lemma 2.4.13 we have $\rho_\emptyset \vdash p'' : s''$ for some $s'' \in P_{a!}(s)$ and $a \in Chan\,(s)$.

$\underline{\Gamma = a?p''}$ Assume $\rho_\emptyset \vdash p'' : s''$ for all $s'' \in P_{a?}(s_1)$. By lemma 2.4.13 and if $a \neq b$ we have $P_{a?}(s_1) = P_{a?}(s_1 \backslash b)$ we get $\rho_\emptyset \vdash p'' : s''$ for all $s'' \in P_{a?}(s)$. By induction $a \in Chan\,(s_1)$ and $\rho_\emptyset \vdash p_1' : s_1$ and $P_{a?}(s_1) \neq \emptyset$. By the non-structural rule we have $\rho_\emptyset \vdash p_1' : s$ and by lemma 2.4.13 we have

$P_{a?}(s) \neq \emptyset$ since $P_{a?}(s_1) = P_{a?}(s_1\backslash b)$ if $a \neq b$. We also get $a \in Chan\,(s)$ since $a \neq b$.

$\underline{p \equiv p_1[S]}$ An argument similar to the argument for the case when $p \equiv p_1\backslash b$ applies.

$\underline{p \equiv x}$ Trivial since $x \not\rightarrow$.

□

2.5 Observational Equivalence

When τ–actions are interpreted as unobservable internal actions the bisimulation equivalence between processes is too distinctive. To refine the bisimulation equivalence we need the following derived transition relations based on observable actions:

Definition 2.5.1

$$p \stackrel{\Gamma}{\Longrightarrow} p'' \equiv p(\stackrel{\tau}{\longrightarrow}^*) \stackrel{\Gamma}{\longrightarrow} p''$$
$$p \stackrel{\varepsilon}{\Longrightarrow} p' \equiv p \stackrel{\tau}{\longrightarrow}^* p'$$

where $p \stackrel{\Gamma}{\longrightarrow}\stackrel{\Gamma'}{\longrightarrow} p''$ *means* $\exists p'.p \stackrel{\Gamma}{\longrightarrow} p'$ & $p' \stackrel{\Gamma'}{\longrightarrow} p''$ *and* $\stackrel{\tau}{\longrightarrow}^*$ *is the reflexive and transitive closure of* $\stackrel{\tau}{\longrightarrow}$.

Intuitively we may read $p \stackrel{a?p'}{\Longrightarrow} p''$ as "after a finite number of internal actions p is in a state where it can receive a process p' on a and in doing so end up in a state p''". If $p \stackrel{\tau}{\Longrightarrow} p''$ we know that p has at least one τ-transition and $\stackrel{\tau}{\Longrightarrow} \equiv \stackrel{\tau}{\longrightarrow}^+$. The above definition of derived transition relations follows [Mil81b] and [Abr87] as opposed to the more common definition: $p \stackrel{a?p'}{\Longrightarrow} p'' \equiv p \stackrel{\tau}{\longrightarrow}^* \stackrel{a?p'}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* p''$. The definition we use facilitates the proofs of congruence properties and we have been unable to prove these with the usual definition of derived transition relations.

Weak higher order bisimulation equivalence or observational equivalence may now be defined:

Definition 2.5.2 *A* weak higher order bisimulation R *is a binary relation on* Pr *such that whenever* pRq *and* $\Phi \in (Act \setminus \{\tau\}) \cup \{\varepsilon\}$ *then:*

(i) Whenever $p \stackrel{\Phi}{\Longrightarrow} p'$, *then* $q \stackrel{\Phi'}{\Longrightarrow} q'$ *for some* q', Φ' *with* $\Phi \hat{\hat{R}} \Phi'$ *and* $p'Rq'$

(ii) Whenever $q \stackrel{\Phi}{\Longrightarrow} q'$, *then* $p \stackrel{\Phi'}{\Longrightarrow} p'$ *for some* p', Φ' *with* $\Phi' \hat{\hat{R}} \Phi$ *and* $p'Rq'$

Where $\hat{\hat{R}} = \{(\Phi, \Phi') : (\Phi = a?p'' \,\&\, \Phi' = a?q'' \,\&\, p''Rq'') \vee (\Phi = a!p'' \,\&\, \Phi' = a!q'' \,\&\, p''Rq'') \vee (\Phi = \Phi' = \varepsilon)\}$.
Two processes p and q are said to be weak higher order bisimulation equivalent iff there exists a weak higher order bisimulation R containing (p, q). *In this case we write* $p \approx q$.

We may define $\mathcal{WHB}(R)$ for $R \subseteq Pr^2$ as the set of pairs (p, q) satisfying clause (i) and (ii) above. It is easy to see that $\mathcal{WHB}$ is a monotone endofunction and that there exists a maximal fixed point for $\mathcal{WHB}$. This equals $\approx$.

Proposition 2.5.3 $\approx$ *is an equivalence*

PROOF: As proposition 2.3.4. □

Bisimulation equivalence is more discriminating than observational equivalence which is a direct consequence of the following proposition.

Proposition 2.5.4 $p \sim p' \Rightarrow p \approx p'$

PROOF: The relation $R = \{(p, q) : p \sim q\}$ is a weak higher order bisimulation which follows from $p \xrightarrow{\Gamma} p'$ implies $p \overset{\Gamma}{\Longrightarrow} p'$. □

As a consequence of proposition 2.5.4 we know that $\approx$ satisfies the equations of propositions 2.3.12 to 2.3.17. Moreover $\approx$ satisfies the following:

Proposition 2.5.5 $p \approx \tau.p$.

PROOF: The relation $R = \{(p, \tau.p)\} \cup Id$ is a weak higher order bisimulation. To see this observe that
if $p \overset{\Phi}{\Longrightarrow} p'$ then $\tau.p \xrightarrow{\tau} p \overset{\Phi}{\Longrightarrow} p'$ thus $\tau.p \overset{\Phi}{\Longrightarrow} p'$ and $(\Phi, \Phi) \in \widehat{\widehat{Id}} \subseteq \hat{\hat{R}}$ and $(p', p') \in Id \subseteq R$.
Also if $\tau.p \overset{\Phi}{\Longrightarrow} p'$ then

either $\Phi = \varepsilon$ thus $\tau.p \xrightarrow{\tau}{}^{*} p'$ and

- **either** $p' = \tau.p$ in which case $p \overset{\varepsilon}{\Longrightarrow} p$ which is a matching move since $(\varepsilon, \varepsilon) \in \hat{\hat{R}}$ and $(p, \tau.p) \in R$.
- **or** $\tau.p \xrightarrow{\tau} p \xrightarrow{\tau}{}^{*} p'$ in which case $p \xrightarrow{\tau}{}^{*} p'$ and since $(\varepsilon, \varepsilon) \in \hat{\hat{R}}$ and $(p', p') \in Id \subseteq R$ we have a matching move.

or $\Phi = \Gamma$ and $\Gamma \neq \tau$ thus $\tau.p \xrightarrow{\tau} p \overset{\Gamma}{\Longrightarrow} p'$ and then $p \overset{\Gamma}{\Longrightarrow} p'$ which is a matching move since $(\Gamma, \Gamma) \in \widehat{\widehat{Id}} \subseteq \hat{\hat{R}}$ and $(p', p') \in Id \subseteq R$.

□

The observational equivalence $\approx$ does not enjoy the property of being a congruence with respect to the operators of CHOCS. As for CCS it is the (nondeterministic) choice operator which presents problems as may be seen from the following counter example first presented in [Mil80]:

Example 2.5.6 *$\tau.nil \approx nil$ but $a!.nil + \tau.nil \not\approx a!.nil + nil$ since $a!.nil + \tau.nil \stackrel{\varepsilon}{\Longrightarrow} nil$ but $a!.nil + nil \stackrel{\varepsilon}{\not\Longrightarrow} nil$.*

More surprisingly perhaps is that $\approx$ is not in general preserved by parallel composition which may be seen from the following example:

Example 2.5.7 *Let $p_1 = b?x.(a!.nil + x)$ and $q_1 = b!(\tau.nil).nil$ and $q_2 = b!(nil).nil$. Then $p_1 \approx p_1$ and $q_1 \approx q_2$ but $p_1 \mid q_1 \not\approx p_1 \mid q_2$ since $p_1 \mid q_1 \stackrel{\tau}{\longrightarrow} (a!.nil + \tau.nil) \mid nil$ whereas $p_1 \mid q_2 \stackrel{\tau}{\longrightarrow} (a!.nil + nil) \mid nil$ and as we saw above these two states are incomparable. In this example the states distinguishing $p_1 \mid q_1$ from $p_1 \mid q_2$ occur after just one transition, but it is easy to generalize the example to any depth of transition.*

Let Pr^- be the set of processes constructed according to the syntax of definition 2.2.1 but without the use of the +-operator. Let $\mathcal{P}^- = (Pr^-, Act^-, \rightarrow)$ where $Act^- = Names \times \{?, !\} \times Pr^- \cup \{\tau\}$ and let CPr^- and $\mathcal{CP}^-$ be defined in the obvious way.

Proposition 2.5.8 *$\approx$ is a congruence relation on Pr^-.*

1. $p[\overline{q}_1/\overline{x}] \approx p[\overline{q}_2/\overline{x}]$ *if* $\overline{q}_1 \approx \overline{q}_2$
2. $a?x.p \approx a?x.q$ *if* $p[r/x] \approx q[r/x]$ *for all* r
3. $a!p'.p \approx a!q'.q$ *if* $p \approx q$ *and* $p' \approx q'$
4. $\tau.p \approx \tau.q$ *if* $p \approx q$
5. $p \mid p' \approx q \mid q'$ *if* $p \approx q$ *and* $p' \approx q'$
6. $p\backslash a \approx q\backslash a$ *if* $p \approx q$
7. $p[S] \approx q[S]$ *if* $p \approx q$

We prove this proposition by showing that $\approx$ is a congruence relation on CPr^- and then lift the definition of $\approx$ to open expressions in the standard way:

Definition 2.5.9

$$p \approx q \text{ iff } \forall \overline{r}.p[\overline{r}/\overline{x}] \approx q[\overline{r}/\overline{x}]$$

where $\overline{x} = FV(p) = FV(q)$ and $\overline{r}$ is closed.

The proof of the congruence property of $\approx$ on CPr^- closely follows the proof of the congruence property of $\sim$. It is useful to have the following alternative definition of weak higher order bisimulation:

Definition 2.5.10 *An* alternative weak higher order bisimulation R *is a binary relation on* Pr *such that whenever* pRq *and* $\Gamma \in Act$ *then:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} q'$ *for some* q', Γ' *with* $\widetilde{\Gamma}\widehat{\widehat{R}}\widetilde{\Gamma'}$ *and* $p'Rq'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, *then* $p \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} p'$ *for some* p', Γ' *with* $\widetilde{\Gamma'}\widehat{\widehat{R}}\widetilde{\Gamma}$ *and* $p'Rq'$

Where $\widetilde{\Gamma} = \begin{cases} \varepsilon & \text{if } \Gamma = \tau \\ a?p & \text{if } \Gamma = a?p \\ a!p & \text{if } \Gamma = a!p \end{cases}$

If there exists an alternative weak higher order bisimulation R *containing* (p,q) *we write* $p \approx' q$.

We may define $\mathcal{AWHB}(R)$ for $R \subseteq Pr^2$ as the set of pairs (p,q) satisfying clause (i) and (ii) above. It is easy to see that $\mathcal{AWHB}$ is a monotone endofunction and that there exists a maximal fixed point for $\mathcal{AWHB}$. This equals $\approx'$.

Proposition 2.5.11 $\approx = \approx'$

PROOF:

$\underline{\approx \Rightarrow \approx'}$ To see this we show that $R_1 = \{(p,q) : p \approx q\}$ is an alternative weak higher order bisimulation.
This is easily established since if $p \xrightarrow{\Gamma} p'$ then $p \stackrel{\Gamma}{\Longrightarrow} p'$. Thus $q \stackrel{\Phi}{\Longrightarrow} q'$ with $\widetilde{\Gamma}\widehat{\widehat{\approx}}\Phi$ and $p' \approx q'$ since $p \approx q$. Define $\Gamma' = \tau$ if $\Phi = \varepsilon$, $\Gamma' = a?p''$ if $\Phi = a?p''$ and $\Gamma' = a!p''$ if $\Phi = a!p''$. Then $q \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} q'$ which is a matching move.
A symmetric argument applies if $q \xrightarrow{\Gamma} q'$.

$\underline{\approx' \Rightarrow \approx}$ To see this we show that $R_2 = \{(p,q) : p \approx' q\}$ is a weak higher order bisimulation.
This is easily established since if $p \stackrel{\Phi}{\Longrightarrow} p'$ then

either $\Phi = \varepsilon$ and $p = p'$. In this case $q \stackrel{\Phi}{\Longrightarrow} q'$ and since $\widetilde{\tau} = \varepsilon$ we have $\varepsilon\widehat{\widehat{\approx}}'\varepsilon$ and $p' \approx' q$ and we have a matching move.

or $\Phi = \varepsilon$ and $p \xrightarrow{\tau}{}^{+} p'$. Then $q \stackrel{\widetilde{\tau}}{\Longrightarrow} q'$ with $\widetilde{\tau}\widehat{\widehat{\approx}}'\widetilde{\tau}$ and $p' \approx' q'$ since $p \approx' q$. This establishes a matching move.

or $\Phi = a?p''$ and $p \stackrel{a?p''}{\Longrightarrow} p'$. Then $q \stackrel{a?p''}{\Longrightarrow} q' \equiv q \stackrel{\widetilde{a?p''}}{\Longrightarrow} q'$ with $\widetilde{a?p''}\widehat{\approx}'\widetilde{a?p''}$ and $p' \approx' q'$ since $p \approx' q$. This establishes a matching move.

or $\Phi = a!p''$ and we may argue as in the previous case.

If $q \stackrel{\Phi}{\Longrightarrow} q'$ a symmetric argument applies.

□

Definition 2.5.12
Let $WCR^- = \{(p[\overline{q}_1/\overline{x}], p[\overline{q}_2/\overline{x}]) : p \in Pr^- \ \& \ \overline{x} = FV(p) \ \& \ \overline{q}_1 \approx \overline{q}_2 \ \& \ \overline{q}_i \in CPr^-\}$ *and let* WCR^{-*} *be the transitive closure of* WCR^-.

Note if $q_1 \approx q_2$ then $(x[q_1/x], x[q_2/x]) \in WCR^-$ and we write $(q_1, q_2) \in WCR^-$.

Lemma 2.5.13
If $p \in CPr^-$ *and* $p \stackrel{a?r}{\longrightarrow} p'$ *and* $(r, r') \in WCR^{-*}$ *for some* r' *then* $p \stackrel{a?r'}{\longrightarrow} p''$ *and* $(p', p'') \in WCR^{-*}$.

PROOF: As lemma 2.3.8. □

With these preliminaries in hand we may now prove the congruence property of $\approx$ on Pr^-.

PROOF: (of proposition 2.5.8) To see 1. we show that the relation WCR^{-*} is an alternative weak higher order bisimulation. First for $(p_1, p_2) \in WCR^-$ we show that: Whenever $p_1 \equiv p[\overline{q}_1/\overline{x}] \stackrel{\Gamma}{\longrightarrow} p'$ then $p_2 \equiv p[\overline{q}_2/\overline{x}] \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} p''$ with $(\widetilde{\Gamma}, \widetilde{\Gamma}') \in \widehat{\widehat{WCR}}^{-*}$ and $(p', p'') \in WCR^{-*}$. We proceed by induction on the length of the inference used to establish the transition $p[\overline{q}_1/\overline{x}] \stackrel{\Gamma}{\longrightarrow} p'$ and cases of the structure of p. The only case which differs slightly from the proof of proposition 2.3.5 is the following:

$\underline{p \equiv p_1 \mid p_2}$ If $p[\overline{q}_1/\overline{x}] \stackrel{\Gamma}{\longrightarrow} p'$ then

either $p_1[\overline{q}_1/\overline{x}] \equiv p_1[\overline{q}_1^1/\overline{x}^1] \stackrel{\Gamma}{\longrightarrow} p_1'$ by a shorter inference and $p' \equiv p_1' \mid p_2[\overline{q}_1/\overline{x}] \equiv p_1' \mid p_2[\overline{q}_1^2/\overline{x}^2]$ where $\overline{x}^i = FV(p_i)$ and $\overline{q}_1^i$ is the respective projection of $\overline{q}_1$. By induction $p_1[\overline{q}_2/\overline{x}] \equiv p_1[\overline{q}_2^1/\overline{x}^1] \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} p_1''$ with $(\widetilde{\Gamma}, \widetilde{\Gamma}') \in \widehat{\widehat{WCR}}^{-*}$ and $(p_1', p_1'') \in WCR^{-*}$. Since $(p_1', p_1'') \in WCR^{-*}$ there exist p_3, $\overline{q}_1^{1'}$, $\overline{q}_2^{1'}$ and $\overline{x}^{1'}$ such that $p_1' \equiv p_3[\overline{q}_1^{1'}/\overline{x}^{1'}]$ and $p_1'' \equiv p_3[\overline{q}_2^{1'}/\overline{x}^{1'}]$ with $FV(p_3) = \overline{x}^{1'}$ and $\overline{q}_1^{1'} \approx \overline{q}_2^{1'}$. We may assume $\overline{x}^{1'} \cap \overline{x}^2 = \emptyset$ since if $\overline{x}^{1'} \cap \overline{x}^2 \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup \overline{x}^{1'} \cup \overline{x}^2) = \emptyset$ and we have $p_3[\overline{q}_i^{1'}/\overline{x}^{1'}] \equiv (p_3[\overline{y}/\overline{x}^{1'}])[\overline{q}_i^{1'}/\overline{y}]$ by proposition 2.2.4. If $\widetilde{\Gamma}' = \varepsilon$ and $p_1'' = p_1[\overline{q}_2^1/\overline{x}^1]$ then $p_1[\overline{q}_2^1/\overline{x}^1] \equiv p_3[\overline{q}_2^{1'}/\overline{x}^{1'}]$ otherwise we use the operational semantics for parallel and we have

$(p_1 \mid p_2)[\overline{q}_2/\overline{x}] \equiv (p_1[\overline{q}_2/\overline{x}]) \mid (p_2[\overline{q}_2/\overline{x}]) \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} p_1'' \mid p_2[\overline{q}_2^2/\overline{x}^2]$. Thus $p_1' \mid p_2[\overline{q}_1^2/\overline{x}^2] \equiv (p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}^{1'} \cup \overline{x}^2]$ and $p_1'' \mid p_2[\overline{q}_2^2/\overline{x}^2] \equiv (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}^{1'} \cup \overline{x}^2]$ and $((p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}^{1'} \cup \overline{x}^2], (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}^{1'} \cup \overline{x}^2]) \in WCR^{-*}$.

or $p_2[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p_2'$ and we may argue as above.

or $\Gamma = \tau$ and w.l.o.g $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a?r} p_1'$ and $p_2[\overline{q}_1/\overline{x}] \xrightarrow{a!r} p_2'$ by shorter inferences and $p' \equiv p_1' \mid p_2'$. By induction $p_2[\overline{q}_2/\overline{x}] \stackrel{a!r_2}{\Longrightarrow} p_2''$ with $(r, r_2) \in WCR^{-*}$ and $(p_2', p_2'') \in WCR^{-*}$ and $p_1[\overline{q}_2/\overline{x}] \stackrel{a?r_1}{\Longrightarrow} p_1''$ with $(r, r_1) \in WCR^{-*}$ and $(p_1', p_1'') \in WCR^{-*}$. Thus $p_1[\overline{q}_2/\overline{x}](\xrightarrow{\tau}{}^*)p_5 \xrightarrow{a?r_1} p_1''$ for some p_5. By lemma 2.5.13 we have $p_5 \xrightarrow{a?r_2} p_1'''$ with $(r_1, r_2) \in WCR^{-*}$ and $(p_1'', p_1''') \in WCR^{-*}$. Therefore $p_1[\overline{q}_2/\overline{x}](\xrightarrow{\tau}{}^*)p_5 \xrightarrow{a?r_2} p_1''' \equiv p_1[\overline{q}_2/\overline{x}] \stackrel{a?r_2}{\Longrightarrow} p_1'''$. (Note that we need to know that p_1'' is the state immediately after the input-transition in order to apply lemma 2.5.13 as we did above. It is an open question if the proposition can be proved if p_1'' occurs after a sequence of internal moves i.e. if we had used the usual definition of $\stackrel{\Gamma}{\Longrightarrow}$). By the operational semantics for parallel $(p_1 \mid p_2)[\overline{q}_2/\overline{x}] \stackrel{\varepsilon}{\Longrightarrow} p_1''' \mid p_2''$. Since $(p_1', p_1''') \in WCR^{-*}$ there exist p_3, $\overline{q}_1^1$, $\overline{q}_2^1$ and $\overline{x}^1$ such that $p_1' \equiv p_3[\overline{q}_1^1/\overline{x}^1]$ and $p_1''' \equiv p_3[\overline{q}_2^1/\overline{x}^1]$ with $FV(p_3) = \overline{x}^1$ and $\overline{q}_1^1 \approx \overline{q}_2^1$ and since $(p_2', p_2'') \in WCR^{-*}$ there exist p_4, $\overline{q}_1^2$, $\overline{q}_2^2$ and $\overline{x}^2$ such that $p_2' \equiv p_4[\overline{q}_1^2/\overline{x}^2]$ and $p_2'' \equiv p_4[\overline{q}_2^2/\overline{x}^2]$ with $FV(p_4) = \overline{x}^2$ and $\overline{q}_1^2 \approx \overline{q}_2^2$. We may assume $\overline{x}^1 \cap \overline{x}^2 = \emptyset$ since if $\overline{x}^1 \cap \overline{x}^2 \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup FV(p_4) \cup \overline{x}^1 \cup \overline{x}^2) = \emptyset$ and we have $p_3[\overline{q}_i^1/\overline{x}^1] \equiv (p_3[\overline{y}/\overline{x}^1])[\overline{q}_i^1/\overline{y}]$ by proposition 2.2.4. Therefore we have $p_1' \mid p_2' \equiv (p_3[\overline{q}_1^1/\overline{x}^1]) \mid (p_4[\overline{q}_1^2/\overline{x}^2]) \equiv (p_3 \mid p_4)[\overline{q}_1^1 \cup \overline{q}_1^2/\overline{x}^1 \cup \overline{x}^2]$ and $p_1''' \mid p_2'' \equiv (p_3[\overline{q}_2^1/\overline{x}^1]) \mid (p_4[\overline{q}_2^2/\overline{x}^2]) \equiv (p_3 \mid p_4)[\overline{q}_2^1 \cup \overline{q}_2^2/\overline{x}^1 \cup \overline{x}^2]$ and $(p_1' \mid p_2', p_1''' \mid p_2'') \in WCR^{-*}$.

Next we show that if $(p_1, p_2) \in WCR^{-*}$ and $p_1 \stackrel{\widetilde{\Gamma}}{\Longrightarrow} p_1''$ then $p_2 \stackrel{\widetilde{\Gamma}'}{\Longrightarrow} p_2''$ and $(\widetilde{\Gamma}, \widetilde{\Gamma}') \in \widehat{\widehat{WCR}}^{-*}$ and $(p_1'', p_2'') \in WCR^{-*}$. This follows by induction on (m, n) in the lexicographic order on $\omega \times \omega$ where m is the number of $\longrightarrow$-transitions used in establishing $p_1 \stackrel{\widetilde{\Gamma}}{\Longrightarrow} p_1''$ and n is the length of the transitive sequence used to establish $(p_1, p_2) \in WCR^{-*}$. The base case $(0,0)$ is trivial since $p_i \stackrel{\varepsilon}{\Longrightarrow} p_i$ for $i \in \{1,2\}$ and $(p_1, p_2) \in WCR^{-}$. The case $(0, n+1)$ follows by induction and the base case. To prove the inductive step it is useful to prove the case $(1, n)$ for all n. The case $(1,0)$ follows from the first step above. The inductive step $(1, n+1)$ follows by applying the first step above to the first pair in the transitive sequence and induction on the remaining n pairs. To see the case $(m+1, n)$ for all n, we use induction on (m, n) for all n and apply the result for $(1, n)$ for all n for the last transition in the sequence.

The overall result then follows by induction on the length of the transitive sequence. The base case $n = 0$ follows from the first step above and the inductive step follows by applying the first step above to the first pair $(p_1, p_2) \in WCR^-$ of the sequence $p_1 \ldots p_{n+1}$ reducing its length by 1, then applying the induction hypothesis and the second step above to the rest of the sequence $p_2 \ldots p_{n+1}$.

2. to 7. follow by constructions similar to those given in the proof of 2. to 8. in proposition 2.3.5. □

The above result only applied to processes in Pr^-. We may obtain a congruence relation on Pr containing $\approx$ using techniques presented in [Mil80, Mil89].

Definition 2.5.14

$$p \approx^c q \text{ iff } \forall C.C[p] \approx C[q]$$

where C is a context.

Generally a context is an expression with zero or more "holes" to be filled by an expression. We write $C[p]$ for $C[\,]$ with p exchanged for $[\,]$. We deliberately use the word exchange instead of substitute since according to the definition of substitution (definition 2.2.3) change of bound variables is taken care of, whereas free variables in p may become bound in $C[p]$.

Proposition 2.5.15 1. $\approx^c$ *is a congruence*

2. *if* $\bowtie$ *is a congruence and* $p \bowtie q \Rightarrow p \approx q$ *then* $p \bowtie q \Rightarrow p \approx^c q$.

Corollary 2.5.16 $p \sim q \Rightarrow p \approx^c q$.

Note that we do not have to define $\approx^c$ on closed expressions first and then lift it to open expressions since $a?x_1 \ldots a?x_n.[\,]$ is just a special context.

The definition of observational congruence yields that $\approx^c$ is the largest congruence containing $\approx$. The definition is rather awkward to work with and it is useful to find alternative descriptions. A "standard" alternative characterization of observational congruence first presented in [Mil80] is in terms of +-contexts:

Definition 2.5.17 $p \approx^+ q$ *iff* $\forall r.p + r \approx q + r$

It is surprising to observe that this definition is not in general a congruence relation on Pr as may be seen from the following example:

Example 2.5.18 *Let* $p_1 = b?x.(a!.nil + x)$ *and* $q_1 = b!(\tau.nil).nil$ *and* $q_2 = b!(nil).nil$. *Then* $p_1 \approx^+ p_1$ *and* $q_1 \approx^+ q_2$ *but* $p_1 \mid q_1 \not\approx p_1 \mid q_2$ *since for all* r *we have* $(p_1 \mid q_1) + r \xrightarrow{\tau} (a!.nil + \tau.nil) \mid nil$ *whereas* $(p_1 \mid q_2) + r \xrightarrow{\tau} (a!.nil + nil) \mid nil$ *and as we saw in example 2.5.6 these two states are incomparable.*

At first this may seem quite surprising, but the power of sending and receiving processes in communication might suggest that the congruence property should always hold for the communicated processes since these might turn up at any stage and that we therefore should look for a recursive formulation of observational congruence.

As an attempt to define a bisimulation-like predicate characterizing observational congruence we propose the following definition inspired by [Abr90b].

Definition 2.5.19 *An* irreflexive weak higher order bisimulation R *is a binary relation on* Pr *such that whenever* pRq *and* $\Gamma \in Act$ *then:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \stackrel{\Gamma'}{\Longrightarrow} q'$ *for some* q', Γ' *with* $\tilde{\Gamma}\hat{\hat{R}}\tilde{\Gamma'}$ *and* $p'Rq'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, *then* $p \stackrel{\Gamma'}{\Longrightarrow} p'$ *for some* p', Γ' *with* $\tilde{\Gamma'}\hat{\hat{R}}\tilde{\Gamma}$ *and* $p'Rq'$

If there exists an irreflexive weak higher order bisimulation R *containing* (p,q) *we write* $p \approx^i q$.

Note that this definition only differs from definition 2.5.10 by insisting that whenever $p \xrightarrow{\Gamma} p'$, then $q \stackrel{\Gamma'}{\Longrightarrow} q'$ and not $q \stackrel{\tilde{\Gamma}'}{\Longrightarrow} q'$. This mean that any τ-transition of p must be matched by at least one τ-transition of q and vice versa.

We may define $\mathcal{IWHB}(R)$ for $R \subseteq Pr^2$ as the set of pairs (p,q) satisfying clause (i) and (ii) above. It is easy to see that $\mathcal{IWHB}$ is a monotone endofunction and that there exists a maximal fixed point for $\mathcal{IWHB}$. This equals $\approx^i$.

Proposition 2.5.20 $\approx^i$ *is a congruence relation on* Pr.

1. $\approx^i$ *is an equivalence.*
2. $p[\overline{q}_1/\overline{x}] \approx^i p[\overline{q}_2/\overline{x}]$ *if* $\overline{q}_1 \approx^i \overline{q}_2$
3. $a?x.p \approx^i a?x.q$ *if* $p[r/x] \approx^i q[r/x]$ *for all* r
4. $a!p'.p \approx^i a!q'.q$ *if* $p \approx^i q$ *and* $p' \approx^i q'$
5. $\tau.p \approx^i \tau.q$ *if* $p \approx^i q$
6. $p + p' \approx^i q + q'$ *if* $p \approx^i q$ *and* $p' \approx^i q'$
7. $p \mid p' \approx^i q \mid q'$ *if* $p \approx^i q$ *and* $p' \approx^i q'$
8. $p\backslash a \approx^i q\backslash a$ *if* $p \approx^i q$

9. $p[S] \approx^i q[S]$ *if* $p \approx^i q$

The proof that $\approx^i$ is an equivalence follows from the same kind of arguments given for proposition 2.3.5 and proposition 2.5.8. We prove $\approx^i$ is a congruence relation on CPr and then lift the definition of $\approx^i$ to open expressions in the standard way. The proof of the congruence property of $\approx^i$ on CPr follows the proof of congruence property of $\approx$ on CPr^-.

Definition 2.5.21
Let $IWCR = \{(p[\overline{q}_1/\overline{x}], p[\overline{q}_2/\overline{x}]) : p \in \Pr \ \& \ \overline{x} = FV(p) \ \& \ \overline{q}_1 \approx^i \overline{q}_2 \ \& \ \overline{q}_i \in CPr\}$ *and let* $IWCR^*$ *be the transitive closure of* $IWCR$.

Note if $q_1 \approx^i q_2$ then $(x[q_1/x], x[q_2/x]) \in IWCR$ and we write $(q_1, q_2) \in IWCR$.

Lemma 2.5.22
If $p \in CPr$ *and* $p \xrightarrow{a?r} p'$ *and* $(r, r') \in IWCR^*$ *for some* r' *then* $p \xrightarrow{a?r'} p''$ *and* $(p', p'') \in IWCR^*$.

PROOF: As lemma 2.3.8 □

PROOF: (of proposition 2.5.20) The proof of 1. follows the proof of proposition 2.5.8.1 i.e. we show that the relation $IWCR^*$ is an irreflexive weak higher order bisimulation. First for $(p_1, p_2) \in IWCR$ we show: Whenever $p_1 \equiv p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then $p_2 \equiv p[\overline{q}_2/\overline{x}] \stackrel{\Gamma'}{\Longrightarrow} p''$ with $(\tilde{\Gamma}, \tilde{\Gamma}') \in \widehat{IWCR^*}$ and $(p', p'') \in IWCR^*$. To do this we proceed by induction on the number of inferences used to establish the transition $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ and cases of the structure of p. The only case which did not occur in the proof of proposition 2.5.8 is the following:

$\underline{p \equiv p_1 + p_2}$ If $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then

either $p_1[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ by a shorter inference. By induction $p_1[\overline{q}_2/\overline{x}] \stackrel{\Gamma'}{\Longrightarrow} p''$ with $(\tilde{\Gamma}, \tilde{\Gamma}') \in \widehat{IWCR^*}$ and $(p', p'') \in IWCR^*$. Since $p_1[\overline{q}_2/\overline{x}] \stackrel{\Gamma'}{\Longrightarrow} p''$ implies that at least one transition takes place we can apply the operational semantics for choice and we have $(p_1 + p_2)[\overline{q}_2/\overline{x}] \stackrel{\Gamma'}{\Longrightarrow} p''$ which is a matching move.

or $p_2[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ and we may argue as above.

As in the proof of proposition 2.5.8.1 we establish that if $(p_1, p_2) \in IWCR^*$ and $p_1 \stackrel{\Gamma}{\Longrightarrow} p_1''$ then $p_2 \stackrel{\Gamma'}{\Longrightarrow} p_2''$ and $(\tilde{\Gamma}, \tilde{\Gamma}') \in \widehat{IWCR^*}$ and $(p_1'', p_2'') \in IWCR^*$. This follows by induction on (m, n) in the lexicographic order on $\omega \times \omega$ where m is the number of $\xrightarrow{\tau}$-transitions used in establishing $p_1 \stackrel{\Gamma}{\Longrightarrow} p_1''$ and n is the length of the transitive sequence used to establish $(p_1, p_2) \in IWCR^*$.

The overall result then follows by an argument similar to the argument given for 2.5.8.1 i.e. by induction on the length of the transitive sequence.

2. to 8. follow by constructions similar to those given in the proof of 2. to 8. in proposition 2.3.5. □

Corollary 2.5.23 $p \approx^i q \Rightarrow p \approx^c q$

PROOF: It is easy to establish that $p \approx^i q \Rightarrow p \approx q$. The proposition then follows from the congruence property of $\approx^i$ and proposition 2.5.15 □

We may now turn to the algebraic properties of $\approx^i$. Since $p \sim q \Rightarrow p \approx^i q$ the algebraic laws for $\sim$ also apply for $\approx^i$. In addition $\approx^i$ satisfies the following τ-law:

Proposition 2.5.24

$$p + \tau.p \approx^i \tau.p$$

PROOF: We show that the relation:

$$R = \{(p + \tau.p, \tau.p) \; : \; p \in CPr\} \cup Id$$

is an irreflexive weak higher order bisimulation.
To see this observe that if $p + \tau.p \xrightarrow{\Gamma} p'$ then

either $p \xrightarrow{\Gamma} p'$ and $\tau.p \stackrel{\Gamma}{\Longrightarrow} p'$ with $(\tilde{\Gamma}, \tilde{\Gamma}) \in \widehat{\widehat{Id}} \subseteq \hat{\hat{R}}$ and $(p', p') \in Id \subseteq R$.

or $\tau.p \xrightarrow{\Gamma} p'$ and $\Gamma = \tau$ and $\tau.p \xrightarrow{\tau} p'$ with $(\tilde{\tau}, \tilde{\tau}) \in \hat{\hat{R}}$ and $(p', p') \in Id \subseteq R$.

Also if $\tau.p \xrightarrow{\Gamma} p'$ then $p' = p$ and $\Gamma = \tau$ and $p + \tau.p \xrightarrow{\Gamma} p'$ with $(\tilde{\tau}, \tilde{\tau}) \in \hat{\hat{R}}$ and $(p', p') \in Id \subseteq R$. □

However, the following τ-laws are not valid for $\approx^i$:

1. $a?x.\tau.p \approx^i a?x.p$
2. $a!p'.\tau.p \approx^i a!p'.p$
3. $\tau.\tau.p \approx^i \tau.p$
4. $a?x.(p + \tau.q) + a?x.q \approx^i a?x.(p + \tau.q)$
5. $a!p'.(p + \tau.q) + a!p'.q \approx^i a!p'.(p + \tau.q)$
6. $\tau.(p + \tau.q) + \tau.q \approx^i \tau.(p + \tau.q)$

The first three laws correspond to the τ-law: $a.\tau.p \approx^c a.p$ of CCS [Mil89]. In fact this law is not valid for $\approx^i$ even without process passing since if $p = nil$ then $a!.\tau.p \xrightarrow{a!} \tau.p$ and $a!.nil \stackrel{a!}{\Longrightarrow} nil$ with $a! \widehat{\approx^i} a!$ but $\tau.nil \not\approx^i nil$ since $\tau.nil \xrightarrow{\tau} nil$ whereas $nil \stackrel{\tau}{\not\Longrightarrow}$. The last three laws are not valid for $\approx^i$ because of the definition of $\stackrel{\Gamma}{\Longrightarrow}$. This fact was noticed by Walker for CCS in [Wal88].

It is an open question if the following τ-laws are valid for $\approx^c$:

1. $a?x.\tau.p \approx^c a?x.p$
2. $a!p'.\tau.p \approx^c a!p'.p$
3. $\tau.\tau.p \approx^c \tau.p$

So far I have been unsuccessful in either validating or refuting these. If they were invalid we should be able to find a context C such that $C[pre.\tau.p] \not\approx C[pre.p]$ where pre is either $a?x$, $a!p'$ or τ. Example 2.5.18 suggests that we look for a context which "strips off" the initial action pre and sends $\tau.p$ respectively p into the troublesome context $a!.nil+[\,]$. However, it does not seem to be possible to define such a context in CHOCS since the processes we send are inactive until received and instantiated for a free variable. It is worth noting that if we had the following operational rule from [Nie89]: $\dfrac{p' \xrightarrow{\tau} p''}{a!p'.p \xrightarrow{\tau} a!p''.p}$ we could refute the above τ-laws.

2.6 Recursion

We have seen that almost all properties of CCS carry over to CHOCS but since CHOCS includes higher order constructs one would expect to find it more powerful and indeed it is. In CCS the recursion operator $\mathtt{rec}\, x.p$ is the only operator capable of introducing infinite behaviours. $\mathtt{rec}\, x.$ is a variable binder and FV and $[\,/\,]$ have to be extended according to this:

$$FV(\mathtt{rec}\, x.p) = FV(p) \setminus \{x\}$$

and

$$(\mathtt{rec}\, y.p)[q/x] \equiv \begin{cases} \mathtt{rec}\, y.(p[q/x]) & \text{if } y \neq x \text{ and } y \notin FV(q) \\ \mathtt{rec}\, z.((p[z/y])[q/x]) & \text{for some } z \neq y \text{ and } z \neq x \text{ and not free in } q \text{ nor } p \text{ otherwise} \end{cases}$$

In CCS recursive processes have the following operational semantics:

$$\frac{p[\mathtt{rec}\, x.p/x] \xrightarrow{\Gamma} p'}{\mathtt{rec}\, x.p \xrightarrow{\Gamma} p'}$$

This inference rule basically says that a recursive process has the same derivations as its unfoldings. In CHOCS we can "program" a recursion construct to obtain infinite behaviours. To a certain extent this construct resembles the Curry paradoxical combinator $Y[\,] = (\lambda x.[\,](xx))(\lambda x.[\,](xx))$ which is often referred to as the Y combinator in the λ–Calculus.

Definition 2.6.1 *Let* $W_x[\,]$ *be the context:*

$$a?x.([\,][(x \mid a!x.nil)\backslash a/x])$$

and let $Y_x[\,]$ *be the context:*

$$(W_x[\,] \mid a!(W_x[\,]).nil)\backslash a$$

Note that if $FV(p) \subseteq \{x\}$ then

$$\begin{aligned} Y_x[p] &\xrightarrow{\tau} (p[(x \mid a!x.nil)\backslash a/x][W_x[p]/x] \mid nil)\backslash a \equiv (p[Y_x[p]/x] \mid nil)\backslash a \\ &\sim (p[Y_x[p]/x])\backslash a \equiv (p\backslash a)[Y_x[p]/x] \end{aligned}$$

By proposition 2.4.7 we have $(p\backslash a)[Y_x[p]/x] \sim p[Y_x[p]/x]$ if $p :: L$ and $a \notin L$.

Note how $Y_x[\,]$ needs a τ–transition to unwind the "recursion". This resembles the unwinding of recursion in the inference rule of recursion in TCCS [HenNic87]: $\mathtt{rec}\, x.p \rightsquigarrow p[\mathtt{rec}\, x.p/x]$, where $\rightsquigarrow$ may be read as $\xrightarrow{\tau}$.

Theorem 2.6.2 $Y_x[p] \sim \mathtt{rec}\, x.\tau.(p\backslash a)$

Corollary 2.6.3 *If* $p :: L$ *and* $a \notin L$ *then* $Y_x[p] \approx \mathtt{rec}\, x.p$

PROOF: if $p :: L$ and $a \notin L$ then $p\backslash a \sim p$ by proposition 2.4.7 and $\tau.p \approx p$ by proposition 2.5.5. We need to prove that $\mathtt{rec}\, x.p \approx \mathtt{rec}\, x.q$ if $p \approx q$. We may rely on the proof in [Mil89] for CCS which only needs minor changes to take the process passing into account. □

In [Tho89] the following alternative Y-context was presented:

$$Y_x[\,] = (a?x.([\,] \mid a!x.nil) \mid a!(a?x.([\,] \mid a!x.nil)).nil)\backslash a$$

This context is limited to processes where x does not occur free in a sending position (i.e. does not occur free in any subsubexpression p' of the form $q = b!p'.p''$ where q is a subexpression of p). With the Y-context of definition 2.6.1 we may program systems which recursively send out copies of themselves.

Example 2.6.4 *Let* $p \equiv b!x.x$ *then according to the inference rules of definition 2.2.5* $Y_x[p]$ *has the following derivations:*

$$\begin{array}{c} Y_x[p] \\ \downarrow\tau \\ (b!x.x[Y_x[p]/x] \mid nil)\backslash a \end{array}$$

$$\begin{array}{c} \downarrow_{b!Y_x[p]} \\ (Y_x[p] \mid nil)\backslash a \\ \downarrow_{\tau} \\ ((b!x.x[Y_x[p]/x] \mid nil)\backslash a \mid nil)\backslash a \\ \downarrow_{b!Y_x[p]} \\ \vdots \end{array}$$

This is almost a specification of a computer virus. Think of the behaviour of $Y_x[p]$ *where* $p = ethernet!x.(x \mid delete\_all\_files!.nil)$ *and the consequences such a program could have in a network of computers connected via an ethernet.*

To prove theorem 2.6.2 we need a bit of technical machinery and we extend the result about bisimulation up to $\sim$ from [Mil83, Mil89] to take the process passing into account.

Definition 2.6.5 *A binary relation* R *on* Pr *is a* higher order bisimulation *up to* $\sim$ *if whenever* pRq *and* $\Gamma \in Act$ *then:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \xrightarrow{\Gamma'} q'$ *for some* q', Γ' *with* $\Gamma \sim \widehat{R} \sim \Gamma'$ *and* $p' \sim R \sim q'$

(ii) Whenever $q \xrightarrow{\Gamma} q'$, *then* $p \xrightarrow{\Gamma'} p'$ *for some* p', Γ' *with* $\Gamma' \sim \widehat{R} \sim \Gamma$ *and* $p' \sim R \sim q'$

Where $\sim \widehat{R} \sim = \{(\Gamma, \Gamma') : (\Gamma = a?p'' \ \& \ \Gamma' = a?q'' \ \& \ p'' \sim R \sim q'') \vee (\Gamma = a!p'' \ \& \ \Gamma' = a!q'' \ \& \ p'' \sim R \sim q'') \vee (\Gamma = \Gamma' = \tau)\}$.
Note that $\sim R \sim$ *is relation composition.*

Proposition 2.6.6 $\sim \widehat{R} \sim = \hat{\sim} \hat{R} \hat{\sim}$

PROOF: If $\Gamma \sim \widehat{R} \sim \Gamma'$ there are three cases

$\Gamma = a?p$ then $\Gamma' = a?p'$ and $p \sim R \sim p'$. Thus there exist p'_1, p''_1 such that $p \sim p'_1$, $p'_1 R p''_1$ and $p''_1 \sim p'$. Clearly $a?p \hat{\sim} a?p'_1$, $a?p'_1 \hat{R} a?p''_1$ and $a?p''_1 \hat{\sim} a?p'$ and we have $\Gamma \hat{\sim} \hat{R} \hat{\sim} \Gamma'$.

$\Gamma = a!p$ and we may argue as above.

$\Gamma = \tau$ then $\Gamma' = \tau$ and $\tau \hat{\sim} \tau$ and $\tau \hat{R} \tau$ and $\tau \hat{\sim} \hat{R} \hat{\sim} \tau$.

If $\Gamma \hat{\sim} \hat{R} \hat{\sim} \Gamma'$ then there are three cases

$\Gamma = a?p$ then $\Gamma' = a?p'$ and there exists Γ_1 and Γ_1' such that $\Gamma \hat{\sim} \Gamma_1$, $\Gamma_1 \hat{R} \Gamma_1'$ and $\Gamma_1'' \hat{\sim} \Gamma'$.
We must have $\Gamma_1 = a?p_1$ and $\Gamma_1' = a?p_1'$ and $p \sim p_1$, $p_1 R p_1'$ and $p_1' \sim p'$ thus $p \sim R \sim p'$ and we have $\Gamma \sim \widehat{R} \sim \Gamma'$.

$\Gamma = a!p$ and we may argue as above.

$\Gamma = \tau$ then $\Gamma' = \tau$ and $\Gamma \sim \widehat{R} \sim \Gamma'$. □

Lemma 2.6.7 *If R is a bisimulation up to $\sim$, then $\sim R \sim$ is a bisimulation.*

PROOF: Assume $(p, q) \in \sim R \sim$. This means that for some p_1, q_1 we have $p \sim p_1 R q_1 \sim q$. Thus if $p \xrightarrow{\Gamma} p'$ then $p_1 \xrightarrow{\Gamma_1} p_1'$ with $\Gamma \hat{\sim} \Gamma_1$ and $p' \sim p_1'$. Since R is a bisimulation up to $\sim$ we know that $q_1 \xrightarrow{\Gamma_1'} q_1'$ with $\Gamma_1 \sim \widehat{R} \sim \Gamma_1'$ and $p_1' \sim R \sim q_1'$ and since $q_1 \sim q$ we have $q \xrightarrow{\Gamma'} q'$ with $\Gamma_1' \hat{\sim} \Gamma'$ and $q_1' \sim p'$. By transitivity of $\sim$ we have $p' \sim \sim R \sim \sim p$, which implies $p \sim R \sim p'$ and $\Gamma \hat{\sim} \sim \widehat{R} \sim \hat{\sim} \Gamma'$ which implies $\Gamma \sim \widehat{R} \sim \Gamma'$ by proposition 2.6.6 and transitivity of $\hat{\sim}$ (which is easily established as a corollary of proposition 2.6.6). Thus we have established a matching move for q.

If $q \xrightarrow{\Gamma} q'$ a symmetric argument to the above applies. □

Proposition 2.6.8 *If R is a bisimulation up to $\sim$ then $R \subseteq \sim$.*

PROOF: Since $\sim R \sim$ is a bisimulation we know that $\sim R \sim \subseteq \sim$. $Id \subseteq \sim$ so $R \subseteq \sim R \sim$ which proves the proposition. □

With this machinery in hand we may now prove theorem 2.6.2

PROOF: For this proof we need the following property of substitution:

$$\text{if } x \neq y \text{ then } p[p'/x][p''/y] \equiv p[p''/y][p'[p''/y]/x]$$

and a simple corollary:

$$\text{if } x \neq y \text{ and } p', p'' \text{ are closed then } p[p'/x][p''/y] \equiv p[p''/y][p'/x]$$

which is easily established by structural induction on p. (They are not corollaries of proposition 2.2.4 since we have to take recursive processes into account.)
Then the relation:

$$R = \{(q[\mathtt{rec}\, x.\tau.(p \backslash a)/x], q[Y_x[p]/x]) \;:\; FV(q) \subseteq \{x\}\}$$

is a bisimulation up to $\sim$. To prove this we show that
if $q[\mathtt{rec}\, x.\tau.(p \backslash a)/x] \xrightarrow{\Gamma'} q'$ then $q[Y_x[p]/x] \xrightarrow{\Gamma''} q''$
with $(\Gamma', \Gamma'') \in \sim \widehat{R} \sim$ and $(q', q'') \in \sim R \sim$
(i.e. we show $(\Gamma', \Gamma_1') \in \hat{\sim}$, $(\Gamma_1', \Gamma_1'') \in \hat{R}$, $(\Gamma_1'', \Gamma'') \in \hat{\sim}$, $(q', q_1') \in \sim$, $(q_1', q_1'') \in R$ and $(q_1'', q'') \in \sim$ for some Γ_1', Γ_1'', q_1' and q_1'').

We prove this by induction on the length of the inference used to establish the transition $q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$ and cases of the structure of q. In the case where q has the form $a?y.q'$ or $\mathbf{rec}\, y.q'$ we need the above properties of substitution. The theorem then follows by choosing $q \equiv x$. (The proof follows the pattern of the proof of proposition 4.6 of [Mil83]). q may have the following structure:

$\underline{q \equiv nil}$ Trivial since both $nil[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \not\to$ and $nil[Y_x[p]/x] \not\to$.

$\underline{q \equiv b?y.q_1}$ Assume $y \neq x$ (otherwise use α-conversion on y).

If $q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q''$ then $\Gamma' = b?r$ for some r
and $q' \equiv q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x][r/y] \equiv q_1[r/y][\mathbf{rec}\, x.\tau.(p\backslash a)/x]$ since
$q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \equiv (b?y.q_1)[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \equiv b?y.(q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x])$
and $\mathbf{rec}\, x.\tau.(p\backslash a)$ is closed.
Note that since r is closed $FV(q_1[r/y]) \subseteq \{x\}$.
Also $q[Y_x[p]/x] \xrightarrow{\Gamma'} q'' = (q_1[Y_x[p]/x])[r/y] \equiv (q_1[r/y])[Y_x[p]/x]$
since $Y_x[p]$ is closed and $y \neq x$.
This is a matching move since $(\Gamma', \Gamma') \subseteq \hat{Id} \subseteq \hat{\sim} \subseteq \sim\widehat{R}\sim$
and $(q', q'') \in R \subseteq \sim R \sim$.

$\underline{q \equiv b!q_1.q_2}$ If $q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$ then
$\Gamma' = b!(q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x])$ and $q' = (q_2[\mathbf{rec}\, x.\tau.(p\backslash a)/x])$ since
$q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \equiv (b!q_1.q_2)[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \equiv$
$b!(q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x]).(q_2[\mathbf{rec}\, x.\tau.(p\backslash a)/x])$.
Also $q[Y_x[p]/x] \xrightarrow{\Gamma''} q''$ where $\Gamma'' = b!(q_1[Y_x[p]/x])$ and $q'' = q_2[Y_x[p]/x]$.
This is a matching move and $(\Gamma', \Gamma'') \in \hat{R} \subseteq \sim\widehat{R}\sim$ and $(q', q'') \in R \subseteq \sim R \sim$.

$\underline{q \equiv q_1 + q_2}$ If $q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \equiv q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x] + q_2[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$
then

either $q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$ by a shorter inference.
By induction $q_1[Y_x[p]/x] \xrightarrow{\Gamma''} q''$ with $(\Gamma', \Gamma'') \in \sim\widehat{R}\sim$ and $(q', q'') \in \sim R \sim$.
By the inference rules for choice
$q[Y_x[p]/x] \equiv q_1[Y_x[p]/x] + q_2[Y_x[p]/x] \xrightarrow{\Gamma''} q''$
which is a matching move.

or $q_2[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$ and we may argue as above.

$\underline{q \equiv q_1 \mid q_2}$ If $q[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \equiv q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \mid q_2[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$
then

either $q_1[\mathbf{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q_1'$ by a shorter inference
and $q' \equiv q_1' \mid (q_2[\mathbf{rec}\, x.\tau.(p\backslash a)/x])$.

By induction $q_1[Y_x[p]/x] \xrightarrow{\Gamma''} q_1''$ with $(\Gamma', \Gamma'') \in \sim\widehat{R}\sim$ and $(q_1', q_1'') \in \sim R\sim$.
Thus there exists r_1 such that $q_1' \sim r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$ and
$r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] R r_1[Y_x[p]/x]$ and $r_1[Y_x[p]/x] \sim q_1''$.
By the inference rules for parallel composition
$q[Y_x[p]/x] \equiv q_1[Y_x[p]/x] \mid q_2[Y_x[p]/x] \xrightarrow{\Gamma''} q_1'' \mid (q_2[Y_x[p]/x])$.
By definition 2.2.1 we have
$q_1 \mid (q_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x]) \sim r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \mid q_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
$\equiv (r_1 \mid q_2)[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
and $q_1 \mid (q_2[Y_x[p]/x]) \sim r_1[Y_x[p]/x] \mid q_2[Y_x[p]/x] \equiv (r_1 \mid q_2)[Y_x[p]/x]$.
Clearly $((r_1 \mid q_2)[\mathtt{rec}\, x.\tau.(p\backslash a)/x], (r_1 \mid q_2)[Y_x[p]/x]) \in R$
and we have established a matching move.

or $q_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q_2'$ and we may argue as above.

or $\Gamma' = \tau$ and $q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma} q_1'$ and $q_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\overline{\Gamma}} q_2'$
by shorter inferences and $q' \equiv q_1' \mid q_2'$.
By induction $q_1[Y_x[p]/x] \xrightarrow{\Gamma''} q_1''$ with $(\Gamma, \Gamma'') \in \sim\widehat{R}\sim$ and $(q_1', q_1'') \in \sim R\sim$.
Thus there exists r_1 such that $q_1' \sim r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
and $r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] R r_1[Y_x[p]/x]$ and $r_1[Y_x[p]/x] \sim q_1''$.
Also $q_2[Y_x[p]/x] \xrightarrow{\overline{\Gamma''}} q_2''$ with $(\overline{\Gamma}, \overline{\Gamma''}) \in \sim\widehat{R}\sim$ and $(q_2', q_2'') \in \sim R\sim$.
Thus there exists r_2 such that $q_2' \sim r_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$ and
$r_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x] R r_2[Y_x[p]/x]$ and $r_2[Y_x[p]/x] \sim q_2''$.
Then by the inference rules for parallel composition
$q[Y_x[p]/x] \equiv q_1[Y_x[p]/x] \mid q_2[Y_x[p]/x] \xrightarrow{\tau} q_1'' \mid q_2''$.
By definition 2.2.1 we have
$q_1' \mid q_2' \sim r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \mid r_2[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
$\equiv (r_1 \mid r_2)[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$ and $q_1'' \mid q_2'' \sim r_1[Y_x[p]/x] \mid r_2[Y_x[p]/x] \equiv (r_1 \mid r_2)[Y_x[p]/x]$.
Clearly $((r_1 \mid r_2)[\mathtt{rec}\, x.\tau.(p\backslash a)/x], (r_1 \mid r_2)[Y_x[p]/x]) \in R$
and we have established a matching move.

$\underline{q \equiv q_1\backslash b}$ If $q[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv q_1\backslash b[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv$
$(q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x])\backslash b \xrightarrow{\Gamma'} q'$
then $q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q_1'$ by a shorter inference and $q' \equiv q_1'\backslash b$.
If $\Gamma' = c?r$ or $\Gamma' = c!r$ then $c \neq b$.
By induction $q_1[Y_x[p]/x] \xrightarrow{\Gamma''} q_1''$ with $(\Gamma', \Gamma'') \in \sim\widehat{R}\sim$ and $(q_1', q_1'') \in \sim R\sim$.
Thus there exists r_1 such that
$q_1' \sim r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$ and $r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] R r_1[Y_x[p]/x]$ and $r_1[Y_x[p]/x] \sim q_1''$.
By the inference rules for restriction

$q[Y_x[p]/x] \equiv q_1\backslash b[Y_x[p]/x] \equiv (q_1[Y_x[p]/x])\backslash b \xrightarrow{\Gamma''} q'' = q_1''\backslash b.$
By definition 2.2.1 we have
$q_1'\backslash b \sim (r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x]\backslash b) \equiv (r_1\backslash b)[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
and $q_1'\backslash b \sim (r_1[Y_x[p]/x])\backslash b \equiv (r_1\backslash b)[Y_x[p]/x]$.
Clearly $(q', q'') \in \sim R \sim$ which is a matching move.

$\underline{q \equiv q_1[S]}$ If $q[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv q_1[S][\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv$
$(q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x])[S] \xrightarrow{\Gamma'} q'$
then $q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma_1'} q_1'$ by a shorter inference and $q' \equiv q_1'[S]$
and $\Gamma_1' = S(\Gamma')$ where $S(a?p) = S(a)?p$, $S(a!p) = S(a)!p$ and $S(\tau) = \tau$.
By induction $q_1[Y_x[p]/x] \xrightarrow{\Gamma_1''} q_1''$ with $(\Gamma_1', \Gamma_1'') \in \widehat{\sim R \sim}$ and $(q_1', q_1'') \in \sim R \sim$.
Thus there exists r_1 such that $q_1' \sim r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
and $r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x] R r_1[Y_x[p]/x]$ and $r_1[Y_x[p]/x] \sim q_1''$.
By the inference rules for renaming
$q[Y_x[p]/x] \equiv q_1[S][Y_x[p]/x] \equiv (q_1[Y_x[p]/x])[S] \xrightarrow{\Gamma''} q'' = q_1''[S]$ where $\Gamma'' = S(\Gamma_1'')$.
By definition 2.2.1 we have
$q_1'[S] \sim (r_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x][S]) \equiv (r_1[S])[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$
and $q_1'[S] \sim (r_1[Y_x[p]/x])[S] \equiv (r_1[S])[Y_x[p]/x]$.
Clearly $(q', q'') \in \sim R \sim$ and $(\Gamma', \Gamma'') \in \widehat{\sim R \sim}$
since $(\Gamma_1', \Gamma_1'') \in \widehat{\sim R \sim}$ implies $(S(\Gamma_1'), S(\Gamma_1'')) \in \widehat{\sim R \sim}$ which is easily established.

$\underline{q \equiv x}$ If $q[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv x[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv \mathtt{rec}\, x.\tau.(p\backslash a) \xrightarrow{\Gamma'} q'$
then $\Gamma' = \tau$ and $q' \equiv (p\backslash a)[\mathtt{rec}\, x.\tau.(p\backslash a)/x]$.
Also
$q[Y_x[p]/x] \equiv Y_x[p] \xrightarrow{\tau} q'' = (p[Y_x[p]/x] \mid nil)\backslash a$
$\sim (p[Y_x[p]/x])\backslash a \equiv (p\backslash a)[Y_x[p]/x]$.
Clearly $((p\backslash a)[\mathtt{rec}\, x.\tau.(p\backslash a)/x], (p\backslash a)[Y_x[p]/x]) \in R$ thus $(q', q'') \in \sim R \sim$.

$\underline{q \equiv \mathtt{rec}\, y.q_1}$ Assume $y \neq x$ (otherwise use α-conversion on y).
Then $q[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv (\mathtt{rec}\, y.q_1)[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \equiv$
$\mathtt{rec}\, y.(q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x]) \xrightarrow{\Gamma'} q'$
since $\mathtt{rec}\, x.\tau.(p\backslash a)$ is closed.
Then $q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x][\mathtt{rec}\, y.(q_1[\mathtt{rec}\, x.\tau.(p\backslash a)/x])/x]$
$\equiv q_1[\mathtt{rec}\, y.q_1/y][\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma'} q'$
by a shorter inference.
By induction (on $q_1[\mathtt{rec}\, y.q_1/y]$) we have
$q_1[\mathtt{rec}\, y.q_1/y][Y_x[p]/x] \equiv q_1[Y_x[p]/x][\mathtt{rec}\, y.(q_1[Y_x[p]/x])/x] \xrightarrow{\Gamma''} q''$

with $(\Gamma', \Gamma'') \in \sim\widehat{R}\sim$ and $(q', q'') \in \sim R \sim$.
By the inference rule for recursion:
$\mathtt{rec}\, y.(q_1[Y_x[p]/x]) \equiv (\mathtt{rec}\, y.q_1)[Y_x[p]/x] \xrightarrow{\Gamma''} q''$
which is a matching move.

We also have to prove that if $q[Y_x[p]/x] \xrightarrow{\Gamma'} q'$ then $q[\mathtt{rec}\, x.\tau.(p\backslash a)/x] \xrightarrow{\Gamma''} q''$ with $(\Gamma', \Gamma'') \in \sim\widehat{R}\sim$ and $(q', q'') \in \sim R \sim$. This is straightforward and follows the pattern of the above argument. □

This proof is limited to the case where at most x is free in q. The extension to the case where there are other free variables is now routine, using the definition of higher order bisimulation for open terms from definition 2.3.10.

2.7 Transition Systems with Divergence

In previous sections we have only studied transition systems of the form $\mathcal{P} = (Pr, Act, \rightarrow)$ and the notion of higher order bisimulation. In this section we add a fourth component; a divergence predicate.

In the study of concurrent systems divergence plays an essential rôle since divergent processes may indefinitely do internal actions and thus prevent any external communications and should therefore be distinguished from stopped processes. Although we shall not study the prospects of using divergent processes as unspecified parts in a partial specifications technique it is worth noting that the formalisms introduced in this section provide the necessary machinery to enable the use of the partial specification techniques presented in [Wal88, LarTho88].

To formalize the notion of divergence we adopt the technique presented in [HenPlo80, Mil81b, Abr87, Wal88] and extend the semantic model of labelled transition systems with a unary basic divergence predicate on processes: $\uparrow$.

Transition systems now take the form $\mathcal{P} = (Pr, Act, \rightarrow, \uparrow)$. The notion of convergence $\downarrow$ is defined as the negation of divergence i.e. $p\downarrow \equiv \neg p\uparrow$.

We may now define the notion of a higher order prebisimulation. This predicate on labelled transition systems with divergence is the extension of bisimulation to take the additional structure of divergence into account.

Definition 2.7.1 *A* higher order prebisimulation R *is a binary relation on* Pr *such that whenever* pRq *and* $\Gamma \in Act$ *then:*

(i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \xrightarrow{\Gamma'} q'$ *for some* q', Γ' *with* $\Gamma \widehat{R} \Gamma'$ *and* $p' R q'$

(ii) Whenever $p \downarrow$ *then* $q \downarrow$ *and if* $q \xrightarrow{\Gamma} q'$, *then* $p \xrightarrow{\Gamma'} p'$ *for some* p', Γ' *with* $\Gamma' \hat{R} \Gamma$ *and* $p' R q'$

Where $\hat{R} = \{(\Gamma, \Gamma') : (\Gamma = a?p'' \ \& \ \Gamma' = a?q'' \ \& \ p''Rq'') \vee (\Gamma = a!p'' \ \& \ \Gamma' = a!q'' \ \& \ p''Rq'') \vee (\Gamma = \Gamma' = \tau)\}$.
If there exists a higher order prebisimulation R *containing* (p, q) *we write* $p \precsim^B q$.

As for higher order bisimulation we may define higher order prebisimulation as the maximal fixed point of a functional on Pr^2. We define $\mathcal{HPB}(R)$ for $R \subseteq Pr^2$ as the set of pairs (p, q) satisfying clause (i) and (ii) above. It is easy to see that $\mathcal{HPB}$ is a monotone endofunction and that there exists a maximal fixed point for $\mathcal{HPB}$. This equals $\precsim^B$.

Proposition 2.7.2 $\precsim^B$ *is a preorder*

PROOF: Reflexivity follows from the fact that: $Id = \{(p, p) \mid p \in Pr\}$ is a higher order prebisimulation.
Transitivity follows from the fact that composition of higher order prebisimulations yields a higher order prebisimulation. □

We let $\sim^B$ denote the equivalence generated by $\precsim^B \cap \precsim^{B^{-1}}$. Clearly $p \sim^B q \Rightarrow p \sim q$.

In the coming sections we shall make use of an alternative (and more explicit) characterization of higher order prebisimulation. This is done by giving a decreasing sequence of relations on Pr^2 given by:

Definition 2.7.3

- $p \precsim_0 q$ *is always true (i.e.* $\precsim_0 = Pr \times Pr$*)*
- $p \precsim_{k+1} q$ *iff* $\forall \Gamma \in Act$*:*

  (i) Whenever $p \xrightarrow{\Gamma} p'$, *then* $q \xrightarrow{\Gamma'} q'$ *for some* q', Γ' *with* $\Gamma \widehat{\precsim_k} \Gamma'$ *and* $p' \precsim_k q'$

  (ii) Whenever $p \downarrow$ *then* $q \downarrow$ *and if* $q \xrightarrow{\Gamma} q'$, *then* $p \xrightarrow{\Gamma'} p'$ *for some* p', Γ' *with* $\Gamma' \widehat{\precsim_k} \Gamma$ *and* $p' \precsim_k q'$

  Where $\widehat{\precsim_k} = \{(\Gamma, \Gamma') : (\Gamma = a?p'' \ \& \ \Gamma' = a?q'' \ \& \ p'' \precsim_k q'') \vee (\Gamma = a!p'' \ \& \ \Gamma' = a!q'' \ \& \ p'' \precsim_k q'') \vee (\Gamma = \Gamma' = \tau)\}$.

(i.e. $\precsim_{k+1} = \mathcal{HPB}(\precsim_k)$*). Let* $\precsim_\omega = \bigcap_k \precsim_k$ *and* $\sim_\omega = \precsim_\omega \cap \precsim_\omega^{-1}$.

This decreasing sequence is bounded below by $\lesssim^B$ and we have $\lesssim_k \supseteq \lesssim_{k+1} \supseteq \lesssim^B$ for all k.

Definition 2.7.4 *A transition system $\mathcal{P} = (Pr, Act, \rightarrow, \uparrow)$ is said to be image finite iff:*

$$\forall p \in Pr.\{(\Gamma, p'') \ : \ p \xrightarrow{\Gamma} p''\} \text{ finite}$$

Note that this is equivalent to defining image finiteness as:

$$\forall p \in Pr.\forall a \in Names.\{(p', p'') : p \xrightarrow{a?p'} p''\} \cup \{(p', p'') : p \xrightarrow{a!p'} p''\} \cup \{p'' : p \xrightarrow{\tau} p''\} \text{ finite}$$

since the set $\{(\Gamma, p'') \ : \ p \xrightarrow{\Gamma} p''\}$ has the same cardinality as the set $\{(p', p'') : \exists a \in Names.p \xrightarrow{a?p'} p''\} \cup \{(p', p'') : \exists a \in Names.p \xrightarrow{a!p'} p''\} \cup \{p'' : p \xrightarrow{\tau} p''\}$. The above definition of image finiteness is stronger than the usual definition of image finiteness given by definition 2.1.6. The stronger version is necessary to facilitate the proof of the next proposition.

Proposition 2.7.5 *If $\mathcal{P} = (Pr, Act, \rightarrow, \uparrow)$ is image finite then $\lesssim_\omega = \lesssim^B$*

PROOF: We prove this proposition by showing that if $\mathcal{P} = (Pr, Act, \rightarrow, \uparrow)$ is image finite then $\mathcal{HPB}$ is anticontinuous. It then follows from classic fix point theory [Tar55] that it has got a maximal fixed point on a complete lattice. Pr^2 is a complete lattice ordered by subset inclusion and we have $\sqcap_k \mathcal{HPB}^k(Pr^2) = \bigcap_k \mathcal{HPB}^k(Pr^2)$, where $\mathcal{HPB}^0 = Id$ and $\mathcal{HPB}^{k+1} = \mathcal{HPB}^k \circ \mathcal{HPB}$. Since $\lesssim^B$ is defined as the maximal fixed point of $\mathcal{HPB}$ on Pr^2 we have $\lesssim^B = \lesssim_\omega$.
To see that $\mathcal{HPB}$ is anticontinuous we must prove $\mathcal{HPB}(\bigcap_k R_k) = \bigcap_k \mathcal{HPB}(R_k)$ where $R_1 \supseteq R_2 \supseteq R_3 \supseteq \ldots R_n \supseteq \ldots$ is a decreasing chain of binary relations over Pr.
The "$\subseteq$"-direction follows directly from monotonicity of $\mathcal{HPB}$ and $\bigcap_k R_k \subseteq R_i$ for all $i \in \omega$. For the "$\supseteq$"-direction, let $(p, q) \in \bigcap_k \mathcal{HPB}(R_k)$. If $p \xrightarrow{\Gamma} p''$ we must find a matching move for q, i.e. Γ' and q'' such that $q \xrightarrow{\Gamma'} q''$ with $(\Gamma, \Gamma') \in \widehat{\bigcap_k R_k}$ and $(p'', q'') \in \bigcap_k R_k$.
Thus for all k there exist Γ'_k and q''_k such that $q \xrightarrow{\Gamma'_k} q''_k$ with $(\Gamma, \Gamma'_k) \in \widehat{R_k}$ and $(p'', q''_k) \in R_k$.
By the image finiteness condition on Pr there is only finitely many pairs (Γ'_k, q''_k). This means that there exists a pair (Γ', q'') such that $(\Gamma, \Gamma') \in \widehat{R_k}$ and $(p'', q'') \in R_k$ for infinitely many $k \in \omega$. Since R_k is decreasing in k we have $(\Gamma, \Gamma') \in \widehat{R_k}$ and $(p'', q'') \in R_k$ for all $k \in \omega$ and thus $(p'', q'') \in \bigcap_k R_k$.
If $(p, q) \in \bigcap_k \mathcal{HPB}(R_k)$ then if $p \downarrow$ also $q \downarrow$ and if $q \xrightarrow{\Gamma} q''$ we may find a matching move for p by an argument as above. □

We briefly turn our attention to CHOCS and see how the new structure of divergence may be used. First we make an extension of the syntax:

$$p ::= \ldots \mid \Omega$$

where Ω is a new constant. This new process should be thought of as the always divergent process with no actions. The operational semantics of CHOCS is then defined by the transition relation defined in definition 2.2.5 and the divergence predicate defined below. Note that since Ω has no actions we do not need to alter the transition relation. The divergence predicate is defined syntax directed as the maximal relation satisfying the following axioms and rules:

Definition 2.7.6

$$\frac{}{\Omega\uparrow} \quad \frac{p\uparrow}{p+p'\uparrow} \quad \frac{p'\uparrow}{p+p'\uparrow} \quad \frac{p\uparrow}{p\mid p'\uparrow} \quad \frac{p'\uparrow}{p\mid p'\uparrow} \quad \frac{p\uparrow}{p\backslash a\uparrow} \quad \frac{p\uparrow}{p[S]\uparrow}$$

Note that this definition yields that only CHOCS processes with unguarded Ω's are divergent.

Proposition 2.7.7 $\precsim^B$ *is a precongruence.*

PROOF: We may prove this for closed expressions as for the congruence properties of $\sim$. We may then lift this result to open expressions. □

Since the equivalence $\sim^B$ implies $\sim$ this equivalence satisfies the laws of section 2.3. In addition $\precsim^B$ satisfies the following law:

Proposition 2.7.8 $\Omega \precsim^B p$

PROOF: The relation $R = \{(\Omega, p) : p \in Pr\}$ is a higher order prebisimulation. To see this observe that $\Omega \not\rightarrow$ and $\Omega\uparrow$. Thus clause (i) and (ii) of definition 2.7.1 are trivially satisfied. □

Except for Ω we do not have any other basic divergent processes in CHOCS. This is opposed to CCS where recursive processes with unguarded recursion variables may be basic divergent as well. This makes the study of basic divergence in the context of CHOCS rather trivial, though we shall use the basic divergence in the formulation of a denotational theory for CHOCS in chapter 4.

When τ-transitions are interpreted as internal/unobservable moves, as in the theory of observational equivalence, we may interpret a process with the potential of evolving into a process possessing the capability of an infinite sequence of τ-transitions as divergent. This is e.g. the case when simulating unguarded recursion: $Y_x[x] \sim \mathtt{rec}\ x.\tau.x$.

To formalize this we define the following derived divergence predicate:

Definition 2.7.9 *Let the relation $\Uparrow$ on Pr be the largest relation satisfying:*

$$p\Uparrow \equiv (p \stackrel{\varepsilon}{\Longrightarrow} p' \ \& \ p'\uparrow) \ \text{or} \ p \stackrel{\tau}{\longrightarrow}^{\omega}$$

where $p \stackrel{\tau}{\longrightarrow}^{\omega} \equiv \exists\{p_n\}.p = p_0 \ \& \ \forall n.p_n \stackrel{\tau}{\longrightarrow} p_{n+1}$.

We interpret $p\Uparrow$ as p may diverge. The notion of convergence is defined as the negation of divergence i.e. $p\Downarrow \equiv \neg(p\Uparrow)$.

We may use this predicate to formulate a notion of weak higher order prebisimulation:

Definition 2.7.10 *A* weak higher order prebisimulation *R is a binary relation on Pr such that whenever pRq and $\Phi \in (Act \setminus \{\tau\}) \cup \{\varepsilon\}$ then:*

(i) Whenever $p \stackrel{\Phi}{\Longrightarrow} p'$, then $q \stackrel{\Phi'}{\Longrightarrow} q'$ for some q', Φ' with $\Phi \hat{\hat{R}} \Phi'$ and $p'Rq'$

(ii) Whenever $p\Downarrow$ then $q\Downarrow$ and if $q \stackrel{\Phi}{\Longrightarrow} q'$, then $p \stackrel{\Phi'}{\Longrightarrow} p'$ for some p', Φ' with $\Phi' \hat{\hat{R}} \Phi$ and $p'Rq'$

Where $\hat{\hat{R}} = \{(\Phi, \Phi') \ : \ (\Phi = a?p'' \,\&\, \Phi' = a?q'' \,\&\, p''Rq'') \vee (\Phi = a!p'' \,\&\, \Phi' = a!q'' \,\&\, p''Rq'') \vee (\Phi = \Phi' = \varepsilon)\}$.
If there exists a weak higher order bisimulation R containing (p, q) we write $p \stackrel{\sqsubseteq}{\approx} q$.

We may define $\mathcal{WHFB}(R)$ for $R \subseteq Pr^2$ as the set of pairs (p, q) satisfying clause (i) and (ii) above. It is easy to see that $\mathcal{WHFB}$ is a monotone endofunction and that there exists a maximal fixed point for $\mathcal{WHFB}$. This equals $\stackrel{\sqsubseteq}{\approx}$.

Proposition 2.7.11 $\stackrel{\sqsubseteq}{\approx}$ *is a preorder*

PROOF: As proposition 2.7.2. □

As for the theory of observational equivalence weak higher order prebisimualtion is not a congruence with respect to the + operator of CHOCS. We could carry through a program as in section 2.5 and prove precongruence properties of the +-free processes and also generate a congruence along the lines of section 2.5 by defining $p \stackrel{\sqsubseteq^c}{\approx} q$ iff $\forall C.C[p] \stackrel{\sqsubseteq}{\approx} C[q]$. We shall not pursue this since the techniques are tedious, but straightforward generalizations of the results from section 2.5.

Proposition 2.7.12 $\stackrel{\sqsubseteq}{\approx} \cap \stackrel{\sqsubseteq}{\approx}^{-1} \Rightarrow \approx$

Thus the equational properties of $\approx$ are also satisfied by the equivalence generated by $\stackrel{\sqsubseteq}{\approx}$. In addition it satisfies the following law:

Proposition 2.7.13 $\tau.(p+\Omega) \stackrel{\Gamma}{\approx} p+\Omega$

PROOF: We show that the relation

$$R = \{(\tau.(p+\Omega), p+\Omega)\} \cup Id$$

is a weak higher order prebisimulation.
To see this observe that if $p+\Omega \stackrel{\Gamma}{\Longrightarrow} p'$ then this is because $p \stackrel{\Gamma}{\Longrightarrow} p'$ and then $\tau.(p+\Omega) \stackrel{\Gamma}{\Longrightarrow} p'$ which is a matching move.
Also $\tau.(p+\Omega)\Uparrow$ and $p+\Omega\Uparrow$ thus clause (ii) of definition 2.7.10 is trivially satisfied.

□

2.8 Finite CHOCS

In this section we define a finite version of CHOCS. We introduce a new operator $a?^F p_1.p_2$ called finite input prefix. Informally we use this construct to approximate the input prefix $a?x.p_1$ by $\Sigma_{p\in P_I} a?^F p.p_1[p/x]$ following ideas for encoding value passing in SCCS from [Mil83].

Let FPr be the set of processes built according to the following syntax:

Definition 2.8.1

$$p ::= nil \mid a?^F p_1.p_2 \mid a!p_1.p_2 \mid \tau.p_1 \mid p_1 + p_2 \mid p_1 \mid p_2 \mid p_1\backslash a \mid p_1[S] \mid \Omega \mid x$$

where $a \in Names$, $x \in V$ *and* $S : Names \to Names$.

Let $CFPr$ be the set of processes built without the use of variables. $CFPr$ is the set of closed Finite CHOCS processes. Note that since there is no variable binding construct in Finite CHOCS we may interpret FPr as the free Σ-algebra $T_\Sigma(V)$ generated by V and the following (one-sorted) signature Σ.

Definition 2.8.2 *Let* $\Sigma = \{\Sigma_n\}_{n\in\omega}$ *where* Σ_n *is the set of operation symbols of arity* n *in* Σ*:*

$$\begin{aligned}
\Sigma_0 &= \{nil, \Omega\} \\
\Sigma_1 &= \{_\backslash a : a \in Names\} \cup \\
&\quad \{_[S] : S : Names \to Names\} \cup \\
&\quad \{\tau._\} \\
\Sigma_2 &= \{a?^F_._ : a \in Names\} \cup \\
&\quad \{a!_._ : a \in Names\} \cup \\
&\quad \{+, |\} \\
\Sigma_n &= \emptyset, n > 2
\end{aligned}$$

We define a subsignature $\Sigma' \subseteq \Sigma$ by omitting the operators for restriction, renaming and parallel composition. *CFPr* is the term algebra T_Σ induced by the operators in Σ.

The operational semantics for Finite CHOCS is given as a labelled transition system with divergence:

Definition 2.8.3 *Let* $\longrightarrow$ *be the smallest subset of* $FPr \times FAct \times FPr$, *where* $FAct = Names \times \{?,!\} \times FPr \cup \{\tau\}$, *closed under the rules:*

---

prefixing: $a?^F p'.p \xrightarrow{a?p'} p$ $\qquad$ $a!p'.p \xrightarrow{a!p'} p$ $\qquad$ $\tau.p \xrightarrow{\tau} p$

choice: $\dfrac{p \xrightarrow{\Gamma} p'}{p+q \xrightarrow{\Gamma} p'}$ $\qquad$ $\dfrac{p \xrightarrow{\Gamma} p'}{q+p \xrightarrow{\Gamma} p'}$

parallel: $\dfrac{p \xrightarrow{\Gamma} p'}{p \mid q \xrightarrow{\Gamma} p' \mid q}$ $\qquad$ $\dfrac{p \xrightarrow{\Gamma} p'}{q \mid p \xrightarrow{\Gamma} q \mid p'}$ $\qquad$ $\dfrac{p \xrightarrow{\Gamma} p'' \; q \xrightarrow{\overline{\Gamma}} q''}{p \mid q \xrightarrow{\tau} p'' \mid q''}$

restriction: $\dfrac{p \xrightarrow{a?p'} p''}{p\backslash b \xrightarrow{a?p'} p''\backslash b}, a \neq b$ $\qquad$ $\dfrac{p \xrightarrow{a!p'} p''}{p\backslash b \xrightarrow{a!p'} p''\backslash b}, a \neq b$ $\qquad$ $\dfrac{p \xrightarrow{\tau} p''}{p\backslash b \xrightarrow{\tau} p''\backslash b}$

renaming: $\dfrac{p \xrightarrow{a?p'} p''}{p[S] \xrightarrow{S(a)?p'} p''[S]}$ $\qquad$ $\dfrac{p \xrightarrow{a!p'} p''}{p[S] \xrightarrow{S(a)!p'} p''[S]}$ $\qquad$ $\dfrac{p \xrightarrow{\tau} p''}{p[S] \xrightarrow{\tau} p''[S]}$

Also let $\uparrow$ *be the maximal relation on FPr satisfying the following rules:*

$$\frac{}{\Omega\uparrow} \qquad \frac{p\uparrow}{p+p'\uparrow} \qquad \frac{p'\uparrow}{p+p'\uparrow} \qquad \frac{p\uparrow}{p \mid p'\uparrow} \qquad \frac{p'\uparrow}{p \mid p'\uparrow} \qquad \frac{p\uparrow}{p\backslash a\uparrow} \qquad \frac{p\uparrow}{p[S]\uparrow}$$

Table 2.8.1: Operational semantics for Finite CHOCS

We now have transition systems $\mathcal{FP} = (FPr, FAct, \rightarrow, \uparrow)$ and $\mathcal{KP} = (T_\Sigma, CFAct, \rightarrow, \uparrow)$, where $CFAct = Names \times \{?, !\} \times CFPr \cup \{\tau\}$, implicitly defined above.

The following proposition gives a more explicit description of these systems:

Proposition 2.8.4 *For all $p_1, p_2 \in FPr$:*

$$
\begin{array}{lll}
(i)(a) & nil\downarrow & (b)\ nil \not\rightarrow \\
(ii)(a) & \Omega\uparrow & (b)\ \Omega \not\rightarrow \\
(iii)(a) & a?^F p_1.p_2 \downarrow & \\
(b) & a?^F p_1.p_2 \xrightarrow{\Gamma} p \iff & \Gamma = a?p_1 \ \&\ p = p_2 \\
(iv)(a) & a!p_1.p_2 \downarrow & \\
(b) & a!p_1.p_2 \xrightarrow{\Gamma} p \iff & \Gamma = a!p_1 \ \&\ p = p_2 \\
(v)(a) & \tau.p_1 \downarrow & \\
(b) & \tau.p_1 \xrightarrow{\Gamma} p \iff & \Gamma = \tau \ \&\ p = p_1 \\
(vi)(a) & (p_1 + p_2)\uparrow \iff & p_1\uparrow \ or\ p_2\uparrow \\
(b) & p_1 + p_2 \xrightarrow{\Gamma} p \iff & p_1 \xrightarrow{\Gamma} p \ or\ p_2 \xrightarrow{\Gamma} p
\end{array}
$$

Parallel composition:

$$
\begin{array}{lllll}
(vii)(a) & (p_1 \mid p_2)\uparrow & \iff & p_1\uparrow \ or\ p_2\uparrow & \\
(b) & (p_1 \mid p_2) \xrightarrow{\Gamma} p & \iff & either & \exists p_1''.p_1 \xrightarrow{\Gamma} p_1'' \ \&\ p'' \equiv p_1'' \mid p_2 \\
 & & & or & \exists p_2''.p_2 \xrightarrow{\Gamma} p_2'' \ \&\ p'' \equiv p_1 \mid p_2'' \\
 & & or & \Gamma = \tau\ \& & \exists \Gamma', p_1'', p_2''.p_1 \xrightarrow{\Gamma'} p_1'' \ \&\ p_2 \xrightarrow{\overline{\Gamma'}} p_2''
\end{array}
$$

Restriction:

$$
\begin{array}{lllll}
(viii)(a) & (p_1\backslash a)\uparrow & \iff & p_1\uparrow & \\
(b) & (p_1\backslash a) \xrightarrow{\Gamma} p'' & \iff & \Gamma = b?p'\ \& & \exists p_1''.p_1 \xrightarrow{b?p'} p_1''\ \&\ b \neq a\ \&\ p'' \equiv p_1''\backslash a \\
 & & or & \Gamma = b!p'\ \& & \exists p_1''.p_1 \xrightarrow{b!p'} p_1''\ \&\ b \neq a\ \&\ p'' \equiv p_1''\backslash a \\
 & & or & \Gamma = \tau\ \& & \exists p_1''.p_1 \xrightarrow{\tau} p_1''\ \&\ p'' \equiv p_1''\backslash a
\end{array}
$$

Renaming:

$$
\begin{array}{lllll}
(ix)(a) & (p_1[S])\uparrow & \iff & p_1\uparrow & \\
(b) & (p_1[S]) \xrightarrow{\Gamma} p'' & \iff & \Gamma = b?p'\ \& & \exists p_1''.p_1 \xrightarrow{a?p'} p_1''\ \&\ b = S(a)\ \&\ p'' \equiv p_1''[S] \\
 & & or & \Gamma = b!p'\ \& & \exists p_1''.p_1 \xrightarrow{a!p'} p_1''\ \&\ b = S(a)\ \&\ p'' \equiv p_1''[S] \\
 & & or & \Gamma = \tau\ \& & \exists p_1''.p_1 \xrightarrow{\tau} p_1''\ \&\ p'' \equiv p_1''[S]
\end{array}
$$

PROOF: By induction on the number of inferences used to establish $p \uparrow$ and $p \xrightarrow{\Gamma} p''$. □

Proposition 2.8.5 1. $\forall p \in FPr.p$ *is image finite.*

2. $\forall p, q \in FPr.p \sqsubseteq^B q \iff p \sqsubseteq_w q$

PROOF: 1. follows easily from proposition 2.8.4 and 2. follows from (1) and proposition 2.7.5 □

For Finite CHOCS we may "eliminate" the use of the restriction, renaming and parallel composition constructs modulo higher order bisimulation, i.e. the following equations hold for renaming and restriction:

Proposition 2.8.6

$$\begin{array}{rcl} \Omega[S] & \sim^B & \Omega \\ (a?^F p'.p)[S] & \sim^B & S(a)?^F p'.p[S] \\ \Omega\backslash a & \sim^B & \Omega \\ (a?^F p'.p)\backslash b & \sim^B & \begin{cases} a?^F p'.p\backslash b & \text{if } b \neq a \\ nil & \text{otherwise} \end{cases} \end{array}$$

For parallel composition we have the following version of the expansion theorem:

Proposition 2.8.7

$$\begin{array}{lrcl} if & p & = & \Sigma_i a_i?^F p'_i.p_i + \Sigma_j a_j!p'_j.p_j \ [+\Omega] \\ and & q & = & \Sigma_k b_k?^F q'_k.q_k + \Sigma_l b_l!q'_l.q_l \ [+\Omega] \\ then & p \mid q & \sim^B & \Sigma_i a_i?^F p'_i.(p_i \mid q) + \Sigma_j a_j!p'_j.(p_j \mid q) + \\ & & & \Sigma_k b_k?^F q'_k.(p \mid q_k) + \Sigma_l b_l!q'_l.(p \mid q_l) + \\ & & & \Sigma_{(i,l)\in\{(i,l)\,:\,a_i=b_l \ \&\ p'_i=q'_l\}} \tau.(p_i \mid q_l) + \\ & & & \Sigma_{(j,k)\in\{(j,k)\,:\,a_j=b_k \ \&\ p'_j=q'_k\}} \tau.(p_j \mid q_k) \ [+\Omega] \end{array}$$

where $[+\Omega]$ *means that the summand* Ω *is optional.* $\Sigma_i \Gamma_i.p_i$ *describes the sum* $\Gamma_1.p_1 + \ldots + \Gamma_n.p_n$ *when* $n > 0$ *and nil if* $n = 0$.

Note that communication only takes place when both port names and the value (process) communicated are equal.

In SCCS [Mil83] Milner introduced a generalized choice operator $\Sigma_{i\in I} p_i$ where I is a countable index set. The operational semantics of this construct is defined by the following rule:

$$\frac{p_i \xrightarrow{\Gamma} p}{\Sigma_{i\in I} p_i \xrightarrow{\Gamma} p}$$

With this construct we can encode value passing in pure synchronization using the following constructs for input prefix: $a?x.p \equiv \Sigma_v a_v.p\{v/x\}$ and $a!v.p \equiv \overline{a_v}.p$ for output prefix. Using this strategy we may attempt to encode process passing in the following way: $a?x.p \equiv \Sigma_{p' \in P_I} a?^F p'.p[p'/x]$ only using the finite input prefix and eliminating the use of variables. Clearly $a?x.p \sim \Sigma_{p' \in P_I} a?^F p'.p[p'/x]$, but the index set is unfortunately self referential.

If we restrict the index set I to a finite set we do not need to introduce a new operator; we can merely use $\Sigma_I p_i$ as shorthand notation for $p_{i_1} + \ldots + p_{i_n}$ where $\{i_1, \ldots, i_n\} = I$ is an enumeration of I.

We shall use this fact in the approximation of $a?x.p$ in Finite CHOCS.

Definition 2.8.8

$$Lev_0 = \{\Omega\}$$

$$Lev_{n+1} = \{\Sigma_{i \in I} p_i : I \textit{ is finite and } p_i \textit{ is either } \Omega,\ a?^F p_1.p_2,\ a!p_1.p_2 \textit{ or } \tau.p_1 \textit{ where } p_1, p_2 \in Lev_n\} \cup Lev_n$$

Note that if *Names* is finite then each set Lev_n is finite. Then for any process in CHOCS we define its n'th approximation p^n in Finite CHOCS as follows:

Definition 2.8.9

$$p^0 = \Omega \textit{ for all } p$$

We define p^{n+1} *structurally:*

$$\begin{aligned}
nil^{n+1} &= nil \\
\Omega^{n+1} &= \Omega \\
(a?x.p_1)^{n+1} &= \Sigma_{p \in Lev_n} a?^F p.p_1^n[p/x] \\
(a!p_1.p_2)^{n+1} &= a!p_1^n.p_2^n \\
(\tau.p_1)^{n+1} &= \tau.p_1^n \\
(p_1 + p_2)^{n+1} &= p_1^n + p_2^n \\
(p_1 \mid p_2)^{n+1} &= p_1^n \mid p_2^n \\
(p_1 \backslash a)^{n+1} &= p_1^n \backslash a \\
(p_1[S])^{n+1} &= p_1^n[S] \\
x^{n+1} &= x
\end{aligned}$$

If p is closed then $p^n \in CFPr$. The approximation p^n does not necessarily reside in Lev_n since $p_1^n[p/x]$ where $p \in Lev_n$ may introduce elements in Lev_{2n}, but we may state the following relationship between p and it approximation p^n:

Proposition 2.8.10 *Assume Names is finite, then:*

$$\forall n.p \sim_n p^n$$

PROOF: It is laborious to prove directly that $\forall n.p \sim_n p^n$. Instead we prove it indirectly by adapting the technique presented in [Hen81]:

Let $F \subseteq Names$ be a finite set and T_F be the set of closed terms which contains no occurrences of any operator $a?^F$, $a?$, $a!$ where $a \notin F$. Define $A_0^F = \{\Omega\}$. Assume there exists a finite set $A_n^F \subseteq FPr$ such that for every $p \in T_F$ there exists some element $p^n \in A_n^F$ such that $p \sim_n p^n$. Let $A_{n+1}^F = \{\Sigma_{i\in I} p_i : I \text{ is finite and } p_i \text{ is either } \Omega,\ a?^F p_1.p_2,\ a!p_1.p_2 \text{ or } \tau.p_1 \text{ where } a \in F,\ p_1, p_2 \in A_n^F,\ i \neq j \implies p_i \neq p_j\}$. Note that $A_{n+1}^F \subseteq FPr$ and A_{n+1}^F is finite. For any p let:

$$\begin{aligned} p^{n+1} \quad = \quad & \Sigma\{a?^F p_1^n.p_2^n \ : \ p \xrightarrow{a?p_1} p_2\} + \\ & \Sigma\{a!p_1^n.p_2^n \ : \ p \xrightarrow{a!p_1} p_2\} + \\ & \Sigma\{\tau.p_1^n \ : \ p \xrightarrow{\tau} p_1\} + \\ & \{\Omega \ : \ p\uparrow\} \end{aligned}$$

Note that p^n is well defined under the assumption that *Names* is finite and $p^{n+1} \in A^{Names}$ and $p \sim_{n+1} p^{n+1}$. □

This proposition will be an important cornerstone in the full abstraction theorem for CHOCS which we establish in chapter 4. The assumption about *Names* being finite might seem a bit too restrictive from a theoretical point of view. (From an implementational point of view it is quite reasonable.) However, none of the results we have or are going to present about CHOCS need to assume that *Names* is infinite. In the theory of CCS there is at least one theorem [Mil80, Wal88] which needs the assumption that *Names* is infinite. This theorem shows that the observational congruence can be characterized in terms of +-contexts. Since this is not the case for CHOCS (see example 2.5.18) we have not found any use for assuming *Names* infinite.

Chapter 3

Using CHOCS

A process calculus should possess the capability of describing computational phenomena in a way which enables analyses of both existing and new systems. In this chapter we apply CHOCS to three examples. The first example is the untyped λ–Calculus. We show how to simulate various reduction strategies from the λ–Calculus by translations into CHOCS. We shall see that some of the most interesting properties of the λ–Calculus are carried over via the translations. We also study the relationship between abstracting equivalences for the λ–Calculus and CHOCS. The main theorems of this section are the full abstraction results (under certain restrictions of observations) for the call-by-name λ–Calculus and for the call-by-value λ–Calculus presented in theorem 3.1.21 respectively theorem 3.1.32.

The second example consists of a semantics for an imperative programming language P studied in both [Mil80] and [Mil89]. We show how we may solve the problem of giving semantics to concurrent procedure invocations with various parameter mechanisms.

The third example is a description of a fault tolerant editor system inspired by the general presentation of such systems in [Pra88]. We show how we may specify and analyze such a system using CHOCS.

3.1 CHOCS and the λ–Calculus

CCS is a powerful language; it is capable of expressing all Turing definable functions by encoding of Turing machines [Mil83]. Since CCS is a sublanguage of CHOCS this must be true for CHOCS as well. But the nature of CHOCS is much closer to the λ–Calculus and in this section we study their relationship.

The language of the λ–Calculus consists of variables, function abstraction and function application:

Definition 3.1.1 *The set of λ–terms Λ is defined inductively as follows:*

1. $x \in \Lambda$
2. $M \in \Lambda \Rightarrow (\lambda x.M) \in \Lambda$
3. $M, N \in \Lambda \Rightarrow (M\ N) \in \Lambda$

where $x \in V$ (a set of variables).

The operator $\lambda x.$ is a variable binder. This introduces a notion of free and bound variables.

Definition 3.1.2 *The set of free variables $FV(M)$ of a term M is defined structurally on M as:*

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x.M) &= FV(M) \setminus \{x\} \\ FV(M\ N) &= FV(M) \cup FV(N) \end{aligned}$$

A variable x occurring in a term M is bound in M if $x \notin FV(M)$.

A very important concept in the λ–Calculus is the notion of substitution. We may substitute a term M' for a free variable occurring in a term M provided we do not bind free variables in M'. This is captured in the following definition:

Definition 3.1.3 *Substitution $M[y := M']$ is defined structurally on M as:*

$$x[y := M'] = \begin{cases} M' & \text{if } y = x \\ x & \text{otherwise} \end{cases}$$

$$(\lambda x.M)[y := M'] = \begin{cases} \lambda x.M & \text{if } x = y \\ \lambda x.(M[y := M']) & \text{if } y \neq x \text{ and } x \notin FV(M') \\ \lambda z.((M[x := z])[y := M']) & \text{otherwise} \\ z \notin FV(M) \cup FV(M') \cup \{x\} \cup \{y\} & \end{cases}$$

$$(M\ N)[y := M'] = (M[y := M'])\ (N[y := M'])$$

A term M is closed if $FV(M) = \emptyset$. The set of closed terms is denoted by Λ^0.

Note that the above definition differs slightly from the definition of substitution given in [Bar84] where all bound variables are assumed to be different and the sets of bound and free variables are assumed not to intersect.

We shall use the following standard terms:

Definition 3.1.4

$$\begin{aligned} I &= \lambda x.x \\ K &= \lambda x.\lambda y.x \\ Y &= \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) \\ \Omega &= (\lambda x.xx)(\lambda x.xx) \end{aligned}$$

The λ–Calculus has a rich theory as documented in e.g. [Bar84], consisting of concepts such as conversion, reduction, theories and models.

We focus on the various notions of reduction (sometimes referred to as evaluation strategies) and the notions of convergence and equivalence.

First we study the perhaps simplest reduction/evaluation/conversion strategy; the call-by-name or lazy reduction strategy. Formally the theory of the Lazy–λ–Calculus is based on the notion of convergence to principal weak head normal form.

Definition 3.1.5 *The relation $M{\Downarrow}N$ is defined inductively over Λ^0 as:*

$$\lambda x.M{\Downarrow}\lambda x.M \qquad \frac{M{\Downarrow}\lambda x.P \quad P[x := N]{\Downarrow}Q}{M\ N{\Downarrow}Q}$$

This relation induces an (unlabelled) transition system $(\Lambda^0, \Downarrow)$. As noted in [Abr90a] the relation $\_ \Downarrow \_$ is itself too "shallow" to yield information about the behaviour of a term under all experiments. Motivated by the theory of concurrency [Abr90a] introduces the notion of applicative (bi)simulation which may be obtained as the maximal fixed point of the following functional:

Definition 3.1.6 *Let R be a binary relation on Λ^0 then*

$$\begin{aligned} (M,N) \in \mathcal{AB}(R) \quad \textit{iff} \quad & M{\Downarrow}\lambda x.M' \Rightarrow N{\Downarrow}\lambda x.N' \ \& \\ & \forall P \in \Lambda^0.(M'[x := P], N'[x := P]) \in R \end{aligned}$$

R is an applicative simulation iff $R \subseteq \mathcal{AB}(R)$. If there exists an applicative simulation R containing (M,N) we write $M \leq^B N$. We use $\simeq^B$ to denote the equivalence induced by $\leq^B$.

We now give a simple translation of the λ–Calculus and we will show that the evaluation strategy enforced by this encoding coincides with lazy reduction.

Definition 3.1.7 *We define* $[\![\ ]\!] : \Lambda \rightarrow$ *CHOCS structurally:*

1. $[\![x]\!] = x$
2. $[\![\lambda x.M]\!] = i?x.i![\![M]\!].nil$
3. $[\![M\ N]\!] = ([\![M]\!][o/i] \mid o![\![N]\!].o?x.x)\backslash o$

Note that for any $M \in \Lambda$: $[\![M]\!] :: \{i\}$ and that application only needs two communication channels. Since the function $[\![\ ]\!] : \Lambda \rightarrow$ *CHOCS* has no additional arguments we may view it as a definition of a set of derived operators in CHOCS. Clause 3. shows how we may view parallel composition as a generalization of function application. However, we need a rather elaborate protocol to ensure that we do not mix arguments in applications and we therefore feed the arguments sequentially. A tempting definition of the clause for application is $[\![M\ N]\!] = ([\![M]\!] \mid i![\![N]\!].nil)\backslash i$. Unfortunately this definition does not work since the restriction $\backslash i$ prevents application to other arguments as in e.g. $M\ N\ N'$. A different approach is presented in [Bou89] where a special operator takes care of this problem. The cost of this is a complication of the definition of equivalence between processes.

In the following we shall see that some of the most interesting properties of the λ–Calculus are carried over via the translation. First we make clear the connection between substitution in the λ–Calculus and in CHOCS.

Lemma 3.1.8

$$[\![M[x := N]]\!] \equiv [\![M]\!][[\![N]\!]/x]$$

PROOF: By structural induction on M. □

Using this lemma we may show that β–conversion in the λ–Calculus is "preserved" by the translation:

Proposition 3.1.9

$$[\![(\lambda x.M)N]\!] \approx [\![M[x := N]]\!]$$

PROOF: We demonstrate how the left hand side of this equation may do an initial series of internal τ–moves to a process equivalent to the right hand side.

$$[\![(\lambda x.M)N]\!] = ((i?x.i![\![M]\!].nil)[o/i] \mid o![\![N]\!].o?x.x)\backslash o$$

$$\downarrow\tau$$

$$((i!([\![M]\!][[\![N]\!]/x]).nil)[o/i] \mid o?x.x)\backslash o$$

$$\downarrow_\tau$$

$$(nil[o/i] \mid (\llbracket M \rrbracket[\llbracket N \rrbracket/x]))\backslash o$$

$$\sim$$

$$\llbracket M \rrbracket[\llbracket N \rrbracket/x]$$

Since $\llbracket M \rrbracket :: \{i\}$ for all $M \in \Lambda$ we may use the properties of proposition 2.3.13 and proposition 2.4.7 to infer the conclusion of this proposition. □

The connection to the theory of concurrency for the applicative (bi)simulation predicate may at first seem somewhat artificial, but we shall attempt to make it more explicit in the following. Notice that in general we do not have the full η–conversion i.e. $\lambda \vdash \lambda x.M\ x = M$ if $x \notin FV(M)$ but if M has the form $\lambda y.M'$ we have $\llbracket \lambda x.(\lambda y.M')\ x \rrbracket \approx \llbracket \lambda x.M[y := x] \rrbracket \approx \llbracket M \rrbracket$ which is easy to establish using the properties of proposition 3.1.9 and lemma 3.1.8. This restricted version of η–conversion is close to the restricted version valid in the Lazy–λ–Calculus of [Abr90a]. Furthermore connections to the Lazy–λ–Calculus are strengthened by the following proposition:

Proposition 3.1.10 $\llbracket \Omega \rrbracket \sim \mathtt{rec}\, x.\tau.x \sim Y_x[x]$

This shows that the standard unsolvable term Ω of the λ–Calculus yields a divergent process in CHOCS, i.e. a process only capable of performing an infinite series of internal moves. These preliminary suggestions may be explored as follows:

Theorem 3.1.11

1. $\llbracket M \rrbracket \approx \llbracket N \rrbracket \ \Rightarrow\ M \simeq^B N$
2. $M \simeq^B N \ \not\Rightarrow\ \llbracket M \rrbracket \approx \llbracket N \rrbracket$

PROOF:

1. follows from proposition 3.1.14 and theorem 3.1.21 which we prove later.
2. follows from the following counter example also studied in [Mil90].
   Let $M = \lambda x.x\ (\lambda y.x\ \Xi\ \Omega\ y)\ \Xi$ and $N = \lambda x.x\ (x\ \Xi\ \Omega\ )\ \Xi$ where $\Xi = Y\ K$. Ong shows that $M \simeq^B N$ in [Ong88]. Let $\mathtt{c} = i?x.i!(x[o/i] \mid i!nil.\llbracket I \rrbracket).nil$. Then $\llbracket M \rrbracket \stackrel{i?\mathtt{c}}{\Longrightarrow} q_1 \stackrel{i?p}{\Longrightarrow}$ for any p, but $\llbracket N \rrbracket \stackrel{i?\mathtt{c}}{\Longrightarrow} q_1 \stackrel{i?p}{\not\Longrightarrow}$. In fact c implements the convergence test used as a counter example for the full abstraction result of the canonical model of the Lazy-λ-Calculus discussed in [Abr90a]. Another troublesome process is $\mathtt{p} = i?x.i!(i?y.i!(x[o/i] \mid y[o/i] \mid i!nil.\llbracket I \rrbracket).nil).nil$ which implements the parallel convergence test.

□

This theorem states that the equivalence on λ-terms induced by $\approx$ is stronger than $\simeq^B$. This is because $\approx$ also takes processes which are not translations of λ-terms into account as e.g. c above, but using a restricted version of the observational equivalence introduced in section 2.5 we can obtain an equivalence on translated terms which coincides with $\simeq^B$.

Definition 3.1.12 *A* weak higher order bisimulation restricted to λ–observations *R is a binary relation on Pr such that whenever pRq and $\Phi \in (Act \setminus \{\tau\}) \cup \{\varepsilon\}$ then:*

(i) Whenever $p \stackrel{\Phi}{\Longrightarrow} p'$, then $q \stackrel{\Phi'}{\Longrightarrow} q'$ for some q', Φ' with $\Phi \widehat{R}_\lambda \Phi'$ and $p' R q'$

(ii) Whenever $q \stackrel{\Phi}{\Longrightarrow} q'$, then $p \stackrel{\Phi'}{\Longrightarrow} p'$ for some p', Φ' with $\Phi' \widehat{R}_\lambda \Phi$ and $p' R q'$

Where $\widehat{R}_\lambda = \{(\Phi, \Phi') : (\Phi = a?[\![P]\!] \,\&\, \Phi' = a?[\![P]\!],\ P \in \Lambda^0) \vee (\Phi = a!p'' \,\&\, \Phi' = a!q'' \,\&\, p'' R q'') \vee (\Phi = \Phi' = \varepsilon)\}$.
Two processes p and q are said to be λ–observational equivalent iff there exists a weak higher order bisimulation restricted to λ–observations R containing (p, q). In this case we write $p \approx_\lambda q$.

If we think of observational equivalence as experimenting with the system by selecting a channel and supplying a process or receiving a process we now restrict ourselves to supply only processes which are translations of λ–terms.

Proposition 3.1.13 *$\approx_\lambda$ is an equivalence*

PROOF: It is straightforward to see that $Id = \{(p, p) : p \in Pr\}$ is a weak higher order bisimulation restricted to λ–observations and $R^T = \{(q, p) : (p, q) \in R\}$ is a weak higher order bisimulation restricted to λ–observations if R is. Finally composition of weak higher order bisimulations restricted to λ–observations are again weak higher order bisimulations restricted to λ–observations. □

Proposition 3.1.14 *$\approx$ implies $\approx_\lambda$*

We now turn our attention to proving that the notion of applicative bisimulation on λ–terms and the notion of weak higher order bisimulation restricted to λ–observations on translated λ–terms coincide.

Lemma 3.1.15 *Let $[\![\Lambda]\!] = \{q : \exists M \in \Lambda . q \approx [\![M]\!]\}$. Then if $q \in [\![\Lambda]\!]$ and $q \stackrel{i?[P]}{\longrightarrow} q'$ for some q' then $q \sim [\![\lambda x.M']\!]$ for some M'.*

PROOF: Obvious, since only translated λ–terms of the form $\lambda x.M$ have $\stackrel{i?[P]}{\longrightarrow}$–transitions. □

Lemma 3.1.16 *if* $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} q$ *for some* q *then* $[\![M]\!] \approx [\![\lambda x.M']\!]$ *for some* M'.

PROOF: By definition 2.5.1 we have $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} q \equiv [\![M]\!] \stackrel{\tau}{\longrightarrow}^* q' \stackrel{i?[P]}{\longrightarrow} q$. From this it is obvious that $q' \approx [\![M]\!]$ and therefore $q' \in [\![\Lambda]\!]$. Since $q' \stackrel{i?[P]}{\longrightarrow} q$ we have $q' \sim [\![\lambda x.M']\!]$ for some M' by lemma 3.1.15 and we therefore have $[\![M]\!] \approx [\![\lambda x.M']\!]$. □

Corollary 3.1.17 *if* $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} q$ *then* $q \approx i![\![M'[x := P]]\!].nil$ *for some* M'.

PROOF: Assume $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} q$ then by lemma 3.1.16 we have $[\![M]\!] \approx [\![\lambda x.M']\!]$ for some M'. By $\rightarrow$ we have $[\![\lambda x.M']\!] \stackrel{i?[P]}{\longrightarrow} i![\![M'[x := P]]\!].nil$. Since $[\![M]\!] \approx [\![\lambda x.M']\!]$ we must have $q \approx i![\![M'[x := P]]\!].nil$. □

Lemma 3.1.18 $M \Downarrow \lambda x.M' \Rightarrow [\![M]\!] \approx [\![\lambda x.M']\!]$

PROOF: By induction on the number of inferences used to establish $M \Downarrow \lambda x.M'$ and cases of the structure of M.

$\underline{M \equiv x}$ Then $M \Downarrow \lambda x.M'$ can not hold and the lemma holds trivially.

$\underline{M \equiv \lambda x.M'}$ Then $M \Downarrow \lambda x.M'$ by an inference of length one.
We also have $[\![\lambda x.M']\!] \approx [\![\lambda x.M']\!]$.

$\underline{M \equiv M''\ N''}$ If $M \Downarrow \lambda x.M'$ then, by definition of $\_ \Downarrow \_$, this is only the case if $M'' \Downarrow \lambda x.Q$ and $Q[x := N''] \Downarrow \lambda x.M'$ for some Q by shorter inferences. Applying the induction hypothesis we have: $[\![M'']\!] \approx [\![\lambda x.Q]\!]$ and $[\![Q[x := N'']]\!] \approx [\![\lambda x.M']\!]$. Using the congruence properties of $\approx$ with respect to the operators used on translated λ–terms we can infer that
$[\![M''\ N'']\!] = ([\![M'']\!][o/i] \mid o![\![N'']\!].o?x.x) \backslash o \approx ([\![\lambda x.Q]\!][o/i] \mid o![\![N'']\!].o?x.x) \backslash o$
$= [\![(\lambda x.Q)\ N'']\!] \approx [\![Q[x := N'']]\!] \approx [\![\lambda x.M']\!]$ which proves the lemma in this case.

□

These properties will enable us to see the relationship between convergence to principal weak head normal form and λ–experiments on translated λ–terms.

Lemma 3.1.19 $M \Downarrow \lambda x.M' \Rightarrow \forall P.\exists q.[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil\ \&\ q \approx_\lambda [\![M'[x := P]]\!]$

PROOF: By induction on the number of inferences used to establish $M \Downarrow \lambda x.M'$ and cases of the structure of M.

$\underline{M \equiv x}$ Then $M \Downarrow \lambda x.M'$ can not hold and the lemma holds trivially.

$\underline{M \equiv \lambda x.M'}$ Then $M \Downarrow \lambda x.M'$ by an inference of length one. We also have $\forall P.[\![M]\!] = i?x.i![\![M']\!].nil \xrightarrow{i?[P]} i![\![M'[x := P]]\!].nil$ which establishes the lemma in this case.

$\underline{M \equiv M''\ N''}$ Assume $M \Downarrow \lambda x.M'$. Then, by definition of $\_\Downarrow\_$, we must have $M'' \Downarrow \lambda x.Q$ and $Q[x := N''] \Downarrow \lambda x.M'$ for some Q by shorter inferences. Applying the induction hypothesis we have:

(i) $\forall P.\exists q.[\![M'']\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \ \&\ q \approx_\lambda [\![Q[x := P]]\!]$

(ii) $\forall P.\exists q.[\![Q[x := N'']]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \ \&\ q \approx_\lambda [\![M'[x := P]]\!]$

By lemma 3.1.16 we have $[\![M'']\!] \approx [\![\lambda x.Q]\!]$. Therefore, by lemma 3.1.9, we can infer that $[\![M''\ N'']\!] \approx [\![Q[x := N'']]\!]$. By (ii) we have $[\![Q[x := N'']]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \ \&\ q \approx_\lambda [\![M'[x := P]]\!]$. This establishes the lemma in this case.

□

Lemma 3.1.20 $\forall P.[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \ \&\ q \approx_\lambda [\![M'[x := P]]\!] \Rightarrow M \Downarrow \lambda x.M'$ *for some* M'.

PROOF: By induction on the number of inferences used to establish $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} q$ and cases of the structure of M.

$\underline{M \equiv x}$ Then $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil$ can not hold and the lemma holds trivially.

$\underline{M \equiv \lambda x.M'}$ Clearly $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \ \&\ q \approx_\lambda [\![M'[x := P]]\!]$. Also $\lambda x.M' \Downarrow \lambda x.M'$ which yields the lemma in this case.

$\underline{M \equiv M''\ N''}$ Assume that for some M' the following holds: $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \ \&\ q \approx_\lambda [\![M'[x := P]]\!]$ for all P. Then by definition of $\Rightarrow$ (and $\rightarrow$) this is only the case if $[\![M'']\!] \xrightarrow{\tau}{}^* q' \xrightarrow{i?[N'']} q''$ and $(q''[o/i] \mid o?x.x)\backslash o \stackrel{[P]}{\Longrightarrow} i!q.nil$ by shorter inferences. By lemma 3.1.16 we have $q' \sim [\![\lambda x.M''']\!]$ for some M''' and therefore $q'' \sim i![\![M'''[x := N''].nil]\!]$. This shows we are able to establish $[\![M'''[x := N]]\!] \stackrel{i?[P]}{\Longrightarrow} q''' \sim i!q.nil$ by a number of inferences not higher than the number used to establish $(q''[o/i] \mid o?x.x)\backslash o \stackrel{i?[P]}{\Longrightarrow} i!q.nil$. By the induction hypothesis we then have $M'' \Downarrow \lambda x.M'''$ and $M'''[x := N''] \Downarrow \lambda x.M'$. By definition 3.1.5 we then have $M''\ N'' \Downarrow \lambda x.M'$ which establishes the lemma in this case.

□

The above properties give the essential keys to our main theorem of this section.

Theorem 3.1.21 $[\![M]\!] \approx_\lambda [\![N]\!] \iff M \simeq^B N$.

PROOF:

$\Rightarrow$ To see this we show that the relation $R = \{(M,N) : [\![M]\!] \approx_\lambda [\![N]\!]\}$ is an applicative (bi)simulation. The result then follows from symmetry of $\approx_\lambda$.
To see that R is an applicative (bi)simulation observe that if $M \Downarrow \lambda x.M'$ for some M' then by lemma 3.1.19 we have $\forall P.\exists q.[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} i!q.nil \;\&\; q \approx_\lambda [\![M'[x := P]]\!]$. Since $[\![M]\!] \approx_\lambda [\![N]\!]$ we know that $[\![N]\!] \stackrel{i?[P]}{\Longrightarrow} i!q'.nil$ with $q \approx_\lambda q'$. By lemma 3.1.16 and corollary 3.1.17 we know that $[\![N]\!] \approx_\lambda [\![\lambda x.N']\!]$ for some N' and $q' \approx_\lambda [\![N'[x := P]]\!]$. Then from lemma 3.1.20 we can infer that $N \Downarrow \lambda x.N'$. Clearly $\forall P.(M'[x := P], N'[x := P]) \in R$ which yields the theorem in this direction.

$\Leftarrow$ To see this we show that the relation $R = R' \cup \{(i!p.nil, i!q.nil) : (p,q) \in R'\} \cup \{(nil,nil)\}$ where $R' = \{(p,q) : \exists M.\exists N.p \approx_\lambda [\![M]\!]\ ,\ q \approx_\lambda [\![N]\!]\ ,\ M \simeq^B N\}$ is a weak higher order bisimulation restricted to λ–observations. To see this observe that if $(p,q) \in R'$ then if $p \stackrel{i?[P]}{\Longrightarrow} p'$ then for some M we have $[\![M]\!] \stackrel{i?[P]}{\Longrightarrow} p'$. By lemma 3.1.16 and corollary 3.1.17 we have $[\![M]\!] \approx_\lambda [\![\lambda x.M']\!]$ and $p' \approx_\lambda i![\![M'[x := P]]\!].nil$. So by lemma 3.1.20 we have $M \Downarrow \lambda x.M'$. To find a matching move for q we look at N. Since $M \simeq^B N$ we know that $N \Downarrow \lambda x.N'$ and $\forall P.M'[x := P] \simeq^B N'[x := P]$. By lemma 3.1.19 we have $[\![N]\!] \stackrel{i?[P]}{\Longrightarrow} i!q'.nil \;\&\; q' \approx_\lambda [\![N'[x := P]]\!]$. Since $q \approx_\lambda [\![N]\!]$ we know that $q \stackrel{i?[P]}{\Longrightarrow} q''$ with $q'' \approx_\lambda i!q'.nil$. This is a matching move since $(p', q'') \in R$. We do not need to check if $p \stackrel{a!p'}{\Longrightarrow} p''$ since we assume $p \approx_\lambda [\![M]\!]$ for some M and $p \stackrel{a!p'}{\Longrightarrow} p''$ is impossible. The theorem in this direction then follows by a symmetric argument for q.

□

From a concurrency point of view the Lazy–λ–Calculus is not very interesting since the calculus enforces sequential evaluation in application; we first evaluate the function until it reaches a weak head normal form, then we do β-reduction and the argument is then evaluated (every time) when needed.

It would be interesting to investigate which properties are necessary to encode the full β-reduction strategy as defined in [Bar84] by the following rules:

Definition 3.1.22

$$(\lambda x.M)\ N \longrightarrow M[x := N] \qquad \frac{M \longrightarrow M'}{M\ N \longrightarrow M'\ N}$$

$$\frac{N \longrightarrow N'}{M\ N \longrightarrow M\ N'} \qquad \frac{M \longrightarrow M'}{\lambda x.M \longrightarrow \lambda x.M'}$$

As we have seen in lemma 3.1.9 the translation of λ–Calculus given in definition 3.1.7 "preserves" β-reduction, but it does not seem to be possible to mimic the last two of the above rules in CHOCS with the semantics given in definition 2.2.5.

In [Bou89] Boudol presents a calculus which features operators from both the λ-Calculus and CCS. The calculus is called the γ-Calculus. Boudol gives a translation of the λ-Calculus which is very similar to the one given by definition 3.1.7, but the evaluation strategy is a bit more eager since he has the following inference rule for output-prefix:

$$\frac{p' \xrightarrow{\tau} p''}{a!p'.p \xrightarrow{\tau} a!p''.p}$$

which means that the argument in the application is allowed to have internal activity of its own. The effect of this is that the evaluation strategy is similar to the Lazy-λ-Calculus, but a bit eager too. In [Nie89] Nielson introduces a merge between the typed λ-Calculus and CSP. This language is called TPL. The operational semantics for this language is very close to a merge of the call-by-value typed λ-Calculus and the operational semantics for CHOCS with the above rule from [Bou89] included together with the following rule:

$$\frac{p \xrightarrow{\tau} p''}{a!p'.p \xrightarrow{\tau} a!p'.p''}$$

For the translation of the Lazy-λ-Calculus this will have no effect since it is constructed such that the output prefix is only used in the context $a!p.nil$ and nil has no transitions. But perhaps these rules together with the following rule for input-prefix:

$$\frac{p \xrightarrow{\tau} p''}{a?x.p \xrightarrow{\tau} a?x.p''}$$

will bring the evaluation closer to full β-reduction? For example: $[\![\lambda x.\Omega]\!] \lesssim [\![\Omega]\!]$ since $[\![\lambda x.\Omega]\!]\Uparrow$ and any transition from $[\![\Omega]\!]$ can be matched by $[\![\lambda x.\Omega]\!]$. However, all the above suggested extensions to the CHOCS semantics seem to violate the idea that the prefixes are primitives for sequential behaviour. It is hard to see how changing the CHOCS semantics would affect the general theory, but as already mentioned in section 2.5 the rule employed by both Boudol and Nielson will mean that the theory of observational congruence will be affected.

Another question related to the subject of reduction strategies in the λ–Calculus is the matter of which abstracting equivalence of the λ–Calculus to relate to abstracting equivalences in CHOCS.

The standard theory λ as presented in [Bar84] may be related to the translation $[\![\ ]\!] : \Lambda \rightarrow CHOCS$ and the properties of CHOCS by the following theorem:

Theorem 3.1.23 *if* $\lambda \vdash M = N$ *then* $[\![M]\!] \approx [\![N]\!]$.

PROOF: By structure of $\lambda \vdash M = N$ □

The converse does not hold in general, but $\approx$ induces an equality relation on Λ and it is straightforward to verify that the relation $R = \{(M,N) : [\![M]\!] \approx [\![N]\!],\ M,N \in \Lambda\}$ is a compatible congruence relation. Proposition 3.1.19 shows that $\beta = \{((\lambda x.M)N, M[x := N]) : M,N \in \Lambda\} \subseteq R$ and therefore $=_\beta \subseteq =_R$. The notion of β–equality is important, but in the standard theory of the λ–Calculus it is the notion of head normal form, based on Böhm trees, which yields the meaning of a λ–term. Terms without a head normal form are identified. But $=_R$ (even with the suggested extensions to the CHOCS semantics) distinguishes more λ–terms than the standard theory since e.g. $[\![\lambda x.\Omega]\!] \not\approx [\![\Omega]\!]$ because $[\![\lambda x.\Omega]\!] \xrightarrow{\lambda?p}$ whereas $[\![\Omega]\!] \overset{\lambda?p}{\not\rightarrow}$. We have not pursued this any further but this opens a range of possibilities for future studies.

Both the Lazy–λ–Calculus and the standard theory for the λ–Calculus contain the full β-reduction rule. But there are interesting reduction strategies which only partly adapt the β-rule. One such interesting reduction strategy for the λ–Calculus is call-by-value reduction. Formally we may put the theory of the call-by-value λ–Calculus on a similar footing to the Lazy–λ–Calculus and define an unlabelled transition system $(\Lambda^0, \Downarrow_v)$ based on the call-by-value convergence predicate defined by the following rules:

Definition 3.1.24 *The relation* $M \Downarrow_v N$ *is defined inductively over* Λ^0 *as:*

$$\lambda x.M \Downarrow_v \lambda x.M \qquad \frac{M \Downarrow_v \lambda x.P \quad N \Downarrow_v Q \quad P[x := Q] \Downarrow_v L}{M\ N \Downarrow_v L}$$

Thus both function and argument in an application have to converge to an abstraction before application takes place. The above rule suggests that both function and argument may be evaluated concurrently.

As an abstracting equivalence we may adapt the applicative (bi)simulation of definition 3.1.5 by using $\Downarrow_v$ instead of $\Downarrow$ thus generating a preorder $\leq_v$ and a derived equivalence $\simeq_v$. Note that the two preorders $\leq^B$ and $\leq_v$ are incomparable since $I \leq^B KI\Omega \not\leq^B \Omega$ and $I \not\leq_v KI\Omega \leq_v \Omega$.

We now give a simple translation of the λ–Calculus and we will show that the evaluation strategy enforced by this encoding coincides with call-by-value reduction.

Definition 3.1.25 *We define* $[\![\ ]\!]_v : \Lambda \to CHOCS$ *structurally:*

1. $[\![x]\!]_v = x$
2. $[\![\lambda x.M]\!]_v = i!(i?x.i![\![M]\!]_v.nil).nil$
3. $[\![M\ N]\!]_v = ([\![M]\!]_v[a/i] \mid [\![N]\!]_v[b/i] \mid a?x.b?y.(x[c/i] \mid c!(i!y.nil).c?x.x)\backslash c)\backslash a\backslash b$

where i, a, b, c *are distinct.*

Note that for any $M \in \Lambda$: $[\![M]\!]_v :: \{i\}$ but application now needs four communication channels as opposed to only two in the translation of the Lazy–λ–Calculus.

We have to ensure that the substitution properties of the λ–Calculus are carried over by this translation:

Lemma 3.1.26

$$[\![M[x := N]]\!]_v \equiv [\![M]\!]_v[[\![N]\!]_v/x]$$

PROOF: By structural induction on M. □

Using this lemma we may show that the restricted β-conversion for the call-by-value λ–Calculus is "preserved" by the translation:

Proposition 3.1.27

$$[\![(\lambda x.M)(\lambda y.N)]\!]_v \approx [\![M[x := (\lambda y.N)]]\!]_v$$

PROOF: We demonstrate how the left hand side of this equation may do an initial series of internal τ–moves to a process equivalent to the right hand side.

$$[\![(\lambda x.M)(\lambda y.N)]\!]_v =$$

$$((i!(i?x.i![\![M]\!]_v.nil).nil)[a/i] \mid [\![\lambda y.N]\!]_v \mid a?x.b?y.(x[c/i] \mid c!(i!y.nil).c?x.x)\backslash c)\backslash a\backslash b$$

$$\downarrow_\tau$$

$$(nil[a/i] \mid (i!(i?y.i![\![N]\!]_v.nil).nil)[b/i] \mid$$

$$b?y.((i?x.i![\![M]\!]_v.nil)[c/i] \mid c!(i!y.nil).c?x.x)\backslash c)\backslash a\backslash b$$

$$\downarrow_\tau$$

$$(nil[a/i] \mid nil[b/i] \mid ((i?x.i![\![M]\!]_v.nil)[c/i] \mid c!(i!(i?y.i![\![N]\!]_v.nil).nil).c?x.x)\backslash c)\backslash a\backslash b$$

$$=$$

$$(nil[a/i] \mid nil[b/i] \mid ((i?x.i!\llbracket M\rrbracket_v.nil)[c/i] \mid c!(\llbracket \lambda y.N\rrbracket_v).c?x.x)\backslash c)\backslash a\backslash b$$

$$\downarrow_\tau$$

$$(nil[a/i] \mid nil[b/i] \mid ((i!(\llbracket M\rrbracket_v[(\llbracket \lambda y.N\rrbracket_v)/x]).nil)[c/i] \mid c?x.x)\backslash c)\backslash a\backslash b$$

$$\downarrow_\tau$$

$$(nil[a/i] \mid nil[b/i] \mid (nil[c/i] \mid \llbracket M\rrbracket_v[(\llbracket \lambda y.N\rrbracket_v)/x])\backslash c)\backslash a\backslash b$$

$$\sim$$

$$\llbracket M\rrbracket_v[\llbracket(\lambda y.N)\rrbracket_v/x]$$

Since $\llbracket M\rrbracket_v :: \{i\}$ for all $M \in \Lambda$ we may use the properties of proposition 2.3.13 and proposition 2.4.7 to infer the conclusion of this proposition. □

As for the Lazy–λ–Calculus we may state the relationship between $\simeq_v$ and $\approx$ of translated λ-terms:

Theorem 3.1.28

1. $\llbracket M\rrbracket_v \approx \llbracket N\rrbracket_v \;\Rightarrow\; M \simeq_v N$
2. $M \simeq_v N \;\not\Rightarrow\; \llbracket M\rrbracket_v \approx \llbracket N\rrbracket_v$

PROOF:

1. follows from proposition 3.1.31 and theorem 3.1.32 which we present later.
2. follows from the counter example:
   Let $L_1 = \lambda x.((x\,I)(x\,K))$ and $L_2 = \lambda x.(((\lambda y.y\,(x\,K))(x\,I))$. These two terms are equivalent under $\simeq_v$. However, there is a difference in the way they use x (or rather an argument substituted for x when L_1 and L_2 are applied). L_1 will concurrently evaluate both occurrences of x whereas L_2 will evaluate the second occurrence of x before reaching the first occurrence. We may use this property to distinguish the two terms when we translate them into CHOCS. To illustrate this point consider the following process:

$$\begin{aligned}\texttt{troubles} \;=\; & i!(i?x.i!\llbracket I\rrbracket_v.nil).nil \\ & + d?x.i!x.nil + d!(i?x.i!\llbracket K\rrbracket_v.nil).i!(i?x.i!\llbracket K\rrbracket_v.nil).nil\end{aligned}$$

and consider the following context:

$$C[\,] = ([\,][a/i] \mid \texttt{troubles}[b/i] \mid a?x.b?y.(x[c/i] \mid c!(i!y.nil).c?x.x)\backslash c)\backslash a\backslash b\backslash d$$

Then $C[[\![L_1]\!]_v] \xrightarrow{\tau}{}^* p_1 \sim [\![K]\!]_v$ as expected. But the two occurrences of `troubles` may non-deterministically communicate with one another and then $C[[\![L_1]\!]_v] \xrightarrow{\tau}{}^* p_1' \sim [\![I]\!]_v$. Also $C[[\![L_2]\!]_v] \xrightarrow{\tau}{}^* p_2 \sim [\![K]\!]_v$ as expected, but $C[[\![L_2]\!]_v] \not\xrightarrow{\tau}{}^* p_2' \sim [\![I]\!]_v$ since the only active occurrence of `troubles` will be prevented from choosing to communicate via d.

□

This theorem states that the equivalence on λ-terms induced by $\approx$ is stronger than $\simeq_v$. This is because $\approx$ also takes processes which are not translations of λ-terms into account as e.g. `troubles` above. We may apply the same technique as we used for the Lazy-λ-Calculus and introduce a version of the observational equivalence which only takes inputs of translated λ-terms into account and we show that this equivalence coincides with $\simeq_v$ on translated terms.

Definition 3.1.29 *A* weak higher order bisimulation restricted to λv–observations *R is a binary relation on Pr such that whenever pRq and $\Phi \in (Act \setminus \{\tau\}) \cup \{\varepsilon\}$ then:*

(i) Whenever $p \stackrel{\Phi}{\Longrightarrow} p'$, then $q \stackrel{\Phi'}{\Longrightarrow} q'$ for some q', Φ' with $\Phi \widehat{R}_{\lambda v} \Phi'$ and $p'Rq'$

(ii) Whenever $q \stackrel{\Phi}{\Longrightarrow} q'$, then $p \stackrel{\Phi'}{\Longrightarrow} p'$ for some p', Φ' with $\Phi' \widehat{R}_{\lambda v} \Phi$ and $p'Rq'$

Where $\widehat{R}_{\lambda v} = \{(\Phi, \Phi') : (\Phi = a?[\![P]\!]_v \,\&\, \Phi' = a?[\![P]\!]_v\,,\ P \in \Lambda^0) \vee (\Phi = a!p'' \,\&\, \Phi' = a!q'' \,\&\, p''Rq'') \vee (\Phi = \Phi' = \varepsilon)\}$.
Two processes p and q are said to be λv–observational equivalent iff there exists a weak higher order bisimulation restricted to λv–observations R containing (p, q). In this case we write $p \approx_{\lambda v} q$.

Proposition 3.1.30 *$\approx_{\lambda v}$ is an equivalence*

Proposition 3.1.31 *$\approx$ implies $\approx_{\lambda v}$*

We can prove the following theorem using the above definition of λv–observational equivalence and proposition 3.1.31 together with an analysis (similar to lemma 3.1.15 to lemma 3.1.20) of the relationship between transitions in the call-by-value λ–Calculus and transitions for the translated terms.

Theorem 3.1.32 $[\![M]\!]_v \approx_{\lambda v} [\![N]\!]_v \iff M \simeq_v N$.

PROOF:

$\Rightarrow$ To see this we show that the relation $R = \{(M,N) : [\![M]\!]_v \approx_{\lambda v} [\![N]\!]_v\}$ is an applicative (bi)simulation. The result then follows from symmetry of $\approx_{\lambda v}$.

$\Leftarrow$ To see this we show that the relation $R = R' \cup \{(i?x.i!p.nil, i?x.i!q.nil) : (p,q) \in R'\} \cup \{(i!p.nil, i!q.nil) : (p,q) \in R'\} \cup \{(nil,nil)\}$ where $R' = \{(p,q) : \exists M.\exists N.p \approx_{\lambda v} [\![M]\!]_v , q \approx_{\lambda v} [\![N]\!]_v , M \simeq_v N\}$ is a weak higher order bisimulation restricted to λv–observations.

□

3.2 CHOCS as a Metalanguage

In this section we study how CHOCS may be used as a metalanguage in the definition of the semantics of programming languages. The study is a case analysis of a simple imperative toy language, called P, first studied in [Mil80]. The meaning of the language P is given in a phrase–by–phrase style resembling denotational language definitions though we shall not give any semantic domains. The language P is devised in such a way that a programmer is partly protected from unwanted deadlocks. This is obtained through a disciplined form of communication between components sharing some resources. In [Mil80] Milner points out the difficulties of describing procedures in P using CCS. It is not obvious that CCS or the extension of CCS justified by the developments in [Mil83] can describe concurrent procedure invocations satisfactorily. In [EngNie86] Engberg and Nielsen show how CCS with labels as first class objects may be used to describe concurrent procedure invocations, unfortunately their solution is very complicated and it does not look like procedure descriptions of sequential programming languages. We show how procedures in P may be handled straightforwardly in a way resembling how procedures in sequential imperative languages are handled in denotational descriptions based on the λ–Calculus. Most of the definitions not concerning procedures may be found in [Mil80], but for the sake of completeness we present the full language definition.

To allow for other values in CHOCS than process values we use the technique of [Mil83] and introduce a $\mathcal{D}$–indexed family of actions $a?_d$, $a!_d$, $d \in \mathcal{D}$ for each value domain $\mathcal{D}$. Due to the fact that only finite sums of processes can be handled in the version of CHOCS presented in this thesis we restrict our attention to finite value domains as e.g. the set of booleans and finite subsets of the integers. We let $\alpha?_x.p$

abbreviate $\Sigma_{d\in D}\alpha?_d.p\{d/x\}$ where $\{d/x\}$ means exchanging all occurrences of x in p by d as e.g. $\alpha?_x.\beta!_x.nil\{d/x\} \equiv \Sigma_{d\in D}\alpha?_d.\beta!_d.nil$. We shall use the following construct from [Mil83]: If b is a boolean valued expression in x then let $\alpha?_x.(if\ b\ then\ p\ else\ p')$ be encoded by $\Sigma_{x\in D\&b}\alpha?_x.p + \Sigma_{x\in D\&\neg b}\alpha?_x.p'$. We should not confuse $\alpha?_x.p$ with $\alpha?x.p$ since the first is a convenient shorthand notation and the latter is part of the CHOCS syntax.

Alternatively we could extend the syntax and semantics of CHOCS to include other types of values like in the FACILE language [GiaMisPra89] or TPL [Nie89]. This is beyond the scope of this thesis and the above encoding of values will suffice for the presentation in this section.

The toy language P

Programs in P are built from declarations D, expressions E and commands C, using assignments to program variables X. Some set of functions F is assumed and for the cause of simplicity we do not consider types of expressions. P has the following abstract syntax:

Declarations:	$D ::=$	$\texttt{var}\ X \mid D; D \mid \texttt{proc}\ P(\texttt{value}\ X, \texttt{result}\ X')\ \texttt{is}\ C$
Expressions:	$E ::=$	$X \mid F(E_1, \ldots, E_n)$
Commands:	$C ::=$	$X := E \mid C; C \mid \texttt{if}\ E\ \texttt{then}\ C\ \texttt{else}\ C' \mid$ $\texttt{while}\ E\ \texttt{do}\ C \mid C\ \texttt{par}\ C' \mid \texttt{input}\ X \mid \texttt{output}\ E \mid$ $\texttt{skip} \mid \texttt{begin}\ D; C\ \texttt{end} \mid \texttt{call}\ P(E, X)$

Table 3.2.1: Syntax of P

In the study of concurrent programming languages a question of interest is how to evaluate programs like:

```
begin
var X;
X := 0;
(X := X + 1 par X := X + 1);
output X
end
```

The semantics presented here will yield the answers 1 or 2. Readers are referred to [Mil80] for a discussion of an alternative specification to rule out the answer 1.

To give a smooth definition of the semantics of P we need some auxiliary definitions.

To each variable X we associate a register Reg_X. Generally it follows the pattern:

$$Loc = \alpha?_x.Reg(x)$$

$$Reg(y) = \alpha?_x.Reg(x) + \gamma!_y.Reg(y)$$

and thus for X we will have $Loc_X = Loc[\alpha_X\ \gamma_X/\alpha\ \gamma]$. Initially we write in a value, thereupon we can read this value on γ or overwrite the contents of Loc via α. We have written the above definition in an equation style to make it more readable. The proper CHOCS definition is: $Loc = (\alpha?_x.h!_x.nil \mid Reg)\backslash h$ where $Reg = Y_{Reg}[h?_x.(\alpha?_x.h!_x.Reg + \gamma!_x.h!_x.Reg)] \mid Y_{Keep}[h?_x.h!_x.Keep]$. The second component of this process takes care of the parameters in the recursion of the above equations. (This is in fact a general technique for simulating the parameterized recursion of [Mil83]). We also associate a register to each procedure P. It may be defined in the same way as above with x substituted with x.

To each n–ary function symbol F we associate a function f which is represented by:

$$b_f = \rho_1?_{x_1}.\ldots.\rho_n?_{x_n}.\rho!_{f(x_1\ldots x_n)}.nil$$

Constants will thus be represented as e.g. $b_{true} = \rho!_{true}.nil$. The result of evaluating an expression is always communicated via ρ. It is therefore useful to define:

$$p\ result\ p' = (p \mid p')\backslash\rho$$

We adopt the protocol of signaling successful termination of commands via δ and it is therefore convenient to define:

$$done = \delta!.nil$$

$$p\ before\ p' = (p[\beta/\delta] \mid \beta?.p')\backslash\beta\ \ (\beta\ new)$$

$$p\ par\ p' = (p[\delta_1/\delta] \mid p'[\delta_2/\delta] \mid \delta_1?.\delta_2?.done + \delta_2?.\delta_1?.done)\backslash\delta_1\backslash\delta_2\ \ (\delta_1, \delta_2\ new)$$

We now give the semantics of P by the translation into CHOCS shown in table 3.2.2.

Declarations:

$$\begin{aligned}
[\![\texttt{var}\ X]\!] &= Loc_X \\
[\![D; D']\!] &= [\![D]\!] \mid [\![D']\!] \\
[\![\texttt{proc}\ P(\texttt{value}\ X,\ \texttt{result}\ Y\ \texttt{is}\ C]\!] &= ((Loc_P \mid \alpha_P!(\ procedure\ process\ ).nil)\backslash\alpha_P) \\
&\quad \mid Transform
\end{aligned}$$

where *procedure process* =
$(((\alpha_{P_v}?_x.\alpha_X!_x.done)\ before\ [\![C]\!]\ before\ (\gamma_Y?_x.\gamma_{P_v}!_x.done)$
$|\ Loc_X\ |\ Loc_Y)\backslash\alpha_X\backslash\alpha_Y\backslash\gamma_X\backslash\gamma_Y)[\alpha^P_{X'}\ \gamma^P_{X'}/\alpha_{X'}\ \gamma_{X'}]$
and $Transform = Y_{Tran}[\Sigma_{X'}\alpha^P_{X'}?_x.\alpha_{X'}!_x.Tran + \Sigma_{X'}\gamma_{X'}?_x.\gamma^P_{X'}!_x.Tran]$

Expressions:

$$\begin{aligned}[\![X]\!] &= \gamma_X?_x.\rho!_x.nil \\ [\![F(E_1,\ldots,E_n)]\!] &= ([\![E_1]\!][\rho_1/\rho]\ |\ \ldots\ |\ [\![E_n]\!][\rho_n/\rho]\ |\ b_f)\backslash\rho_1\ldots\backslash\rho_n\end{aligned}$$

Commands:

$$\begin{aligned}[\![X := E]\!] &= [\![E]\!]\ result\ (\rho?_x.\alpha_X?_x.done) \\ [\![C; C']\!] &= [\![C]\!]\ before\ [\![C']\!] \\ [\![\texttt{if}\ E\ \texttt{then}\ C\ \texttt{else}\ C']\!] &= [\![E]\!]\ result\ \rho?_x.(if\ x\ then\ [\![C]\!]\ else\ [\![C']\!]) \\ [\![\texttt{while}\ E\ \texttt{do}\ C]\!] &= Y_w[[\![E]\!]\ result\ \rho?_x.(if\ x\ then\ ([\![C]\!]\ before\ w)\ else\ done)] \\ [\![C\ \texttt{par}\ C']\!] &= [\![C]\!]\ par\ [\![C']\!] \\ [\![\texttt{input}\ X]\!] &= \iota?_x.\alpha_X!_x.done \\ [\![\texttt{output}\ E]\!] &= [\![E]\!]\ result\ (\rho?_x.o!_x.done) \\ [\![\texttt{skip}\ ]\!] &= done \\ [\![\texttt{begin}\ D; C\ \texttt{end}]\!] &= ([\![D]\!]\ |\ [\![C]\!])\backslash L_D \\ [\![\texttt{call}\ P(E,Z)]\!] &= [\![E]\!]\ result\ ((\rho?_x.\alpha_{P_v}!_x.done) \\ &\quad par\ (\gamma_P?x.x)\ par\ (\gamma_{P_v}?_x.\alpha_Z!_x.done))\backslash\alpha_{P_v}\backslash\gamma_{P_v}\end{aligned}$$

Table 3.2.2: Semantics of P

In the equation for $[\![\texttt{begin}\ D; C\ \texttt{end}]\!]$ we let $\backslash L_D$ abbreviate restriction with respect to α and γ channels for all variables and procedures declared in D. The procedure definition creates a location to store the procedure process. The restriction $\backslash\alpha_P$ ensures that this process cannot be overwritten after the definition. The first parameter to a procedure is the argument and it will use call-by-value parameter mechanism. The second parameter is a result variable. The procedure process needs two locations, one for each parameter. These locations are kept local by the restrictions $\backslash\alpha_X\backslash\alpha_Y\backslash\gamma_X\backslash\gamma_Y$. To ensure static binding of variables in a procedure

body the procedure process is enclosed by a renaming of all read and write signals to variable locations. This is done simply by tagging the signals with the name of the procedure. The tagged signals are able to escape the restriction $\backslash L_D$ of any block except the block where the procedure is defined. The *Transform* process, located in the block where the procedure is defined, transforms the tagged signals back to untagged read and write signals. These will of course affect the variable locations in this environment. The value to the value parameter is communicated via α_{P_v}, and the result of the procedure is communicated via γ_{P_v}. These signals are not affected by the embracing renaming. The procedure call first evaluates the argument then reads the location Loc_P to get a copy of the procedure process. Note how each procedure process is self-contained with local environments for the parameters. If a recursive call to the procedure P occurs in the body C a new copy of the procedure process will be obtained. This is true for concurrent activations of the same procedure as well and we have:

$$[\![\texttt{begin proc } P(\texttt{value } X,\ \texttt{result } Y) \texttt{ is } C;$$

$$\texttt{call } P(E, Z) \texttt{ par call } P(E', Z') \texttt{ end}]\!]$$

$$\approx$$

$$[\![\texttt{begin var } X; \texttt{var } Y; X := E; C; Z := Y \texttt{ end}$$

$$\texttt{par begin var } X; \texttt{var } Y; X := E'; C; Z' := Y \texttt{ end}]\!]$$

which may be verified by expanding the semantic clauses.

Another common parameter mechanism used in imperative programming languages is the call-by-reference mechanism. This mechanism can be modelled in CHOCS by the following semantic definitions:

$$[\![\texttt{proc } P(\texttt{ref } X) \texttt{ is } C]\!] = ((Loc_P \mid \alpha_P!(\ \textit{procedure process}\ ).nil)\backslash\alpha_P) \mid Transform$$

where $\textit{procedure process} = [\![C]\!][\alpha_{P_v}\ \gamma_{P_v}/\alpha_X\ \gamma_X][\alpha^P_{X'}\ \gamma^P_{X'}/\alpha_{X'}\ \gamma_{X'}]$.

$$[\![\texttt{call } P(Y)]\!] = \gamma_P?x.(x[\alpha_Y\ \gamma_Y/\alpha_{P_v}\ \gamma_{P_v}])$$

Note how this parameter mechanism works; we just link the register associated with the parameter in the call with the procedure process via renaming. This is obtained by the inner renaming in the procedure body which ensures that read and write signals to the formal parameter escape the outer renaming. This has the effect that they are linked to the actual parameter in the calling environment.

We can also describe the call-by-name parameter mechanism in CHOCS. To do so we need to redefine the semantics for variables used as formal parameters in call-by-name procedure definitions as:

$$[\![X]\!] = \gamma_{X-req}?.\gamma_X?_x.\rho!_x.nil$$

The $\gamma_{X-req}?$ signal in the above definition will be used as a request signal when X occurs as a call-by-name parameter in a procedure. We now present the definition of call-by-name procedure definitions and procedure calls:

$$[\![\texttt{proc } P(\texttt{name } X) \texttt{ is } C]\!] = ((Loc_P \mid \alpha_P!(\textit{ procedure process }).nil)\backslash\alpha_P) \mid Transform$$

where $\textit{procedure process} = [\![C]\!][\gamma_{P-req}\ \gamma_{P_v}/\gamma_{X-req}\ \gamma_X][\alpha^P_{X'}\ \gamma^P_{X'}/\alpha_{X'}\ \gamma_{X'}]$.

$$[\![\texttt{call } P(E)]\!] = \gamma_P?x.(x \mid Y_w[\gamma_{P-req}!.([\![E]\!]\ result\ \rho?_x.\gamma_{P_v}!_x.w)])\backslash\gamma_{P-req}\backslash\gamma_{P_v}$$

Note that the definition of procedure definitions is almost the same as for the call-by-reference parameter mechanism. Since we only read the value of a call-by-name formal parameter we do not have to consider any α_X signals since these are write signals and will not occur in well formed programs. The real difference arises in the procedure call where any use of the parameter X in the procedure body yields a request signal $\gamma_{X-req}?$ which is renamed to a $\gamma_{P-req}?$ signal in the procedure definition. Each time this triggers the $Y_w[\ldots]$ construction to evaluate the argument $[\![E]\!]$ and deliver the result via γ_{P_v} before it restores itself. Note that every time there is a request for $[\![E]\!]$ it is evaluated from scratch.

If $[\![E]\!]$ does not have any side effects, as in the P semantics, it is unnecessary to evaluate the value of $[\![E]\!]$ every time a reference to the call-by-name parameter is made. Instead we may store the value of $[\![E]\!]$ after the first evaluation and then just supply the stored value. This parameter mechanism is known as lazy evaluation and may be defined as:

$$[\![\texttt{proc } P(\texttt{lazy } X) \texttt{ is } C]\!] = ((Loc_P \mid \alpha_P!(\textit{ procedure process }).nil)\backslash\alpha_P) \mid Transform$$

where $\textit{procedure process} = [\![C]\!][\gamma_{P-req}\ \gamma_{P_v}/\gamma_{X-req}\ \gamma_X][\alpha^P_{X'}\ \gamma^P_{X'}/\alpha_{X'}\ \gamma_{X'}]$.

$$\begin{aligned}[\![\texttt{call } P(E)]\!] \;=\; & \gamma_P?x.(x \mid (\gamma_{P-req}!.([\![E]\!]\ result\ \rho?_x.\gamma_{P_v}!_x.\gamma_{Loc}!_x.nil) \\ & \mid \gamma_{Loc}?_x.Y_w[\gamma_{P-req}!.\gamma_{P_v}!_x.w])\backslash\gamma_{Loc})\backslash\gamma_{P-req}\backslash\gamma_{P_v}\end{aligned}$$

Note that in both the above two parameter mechanisms we may have concurrent requests to the call-by-name respectively lazy parameter which could mean a mix-up of values. This does not happen since the γ_{X-req} signal acts as a semaphore around the actual parameter.

All the above parameter mechanisms need the $Transform$ process to ensure static binding of variables. This is due to the dynamic nature of the restriction operator i.e. processes sent out of the scope of the restriction operator are not affected by the operator. The block structure of P ensures that the $Transform$ processes work but for general modelling of programming languages it might be of importance to have a restriction operator with a static nature. We comment further on this in chapter 5 where we review the semantics of P. It is interesting to note that if we omit the $Transform$ process and the renamings it carries we will encounter procedure invocations with the various parameter mechanisms above, but with dynamic binding of variables in the procedure body.

The language P presented in this section bears much resemblance to the imperative concurrent language studied in [HenPlo79]. A challenging exercise left for future studies is the question of giving P a labelled transition system semantics directly using structural operational semantics [Plo81]. This should be a reasonable task since P has much in common with the language studied in [HenPlo79]. This would then lead on to an investigation of full abstraction between the structural operational semantics and the semantics of P presented in this section.

3.3 A Fault Tolerant Editor

A simple command editor (like `ed` in the Unix operating system) is provided in almost every programming environment. The editor takes in a file, accepts a series of commands and by the end of the session outputs the updated file. The commands can be grouped into two categories: altering commands; like `insert` a letter and `delete` a line and non-altering commands; like `search` for a word and `scroll`. Such an editor could be specified as follows:

$$\begin{aligned} PEditor &= infile?.\ PEdit \\ PEdit &= \Sigma_i altop_i?.\ PEdit + \Sigma_i nonaltop_i?.PEdit + exit?.outfile!.nil \end{aligned}$$

Unfortunately this is a very simplistic or idealized editor. Most users of such editors have experienced that the editor crashes due to events (or faults) out of the user's control. If the user is "lucky" the alterations made to the file during the editing session are lost and if unlucky the file is lost as well. To recover the lost work most operating systems provide a log system which monitors the user's actions by storing every command. The editing session can then be recovered simply by fetching the stored commands and running them again. Note that in the first instance the commands to the editor are treated as values stored in the log

system, but when we rerun the stored commands they are treated as a program. The following description in CHOCS specifies a fault tolerant editor system; as in the editor above, we have ignored the file being edited to simplify the description and focus on the log system:

$$
\begin{aligned}
FEditor &= infile?.(\ FEdit\ \mid Logsys \mid Demon \mid Updater)\backslash intops \\
FEdit &= \Sigma_i altop_i?.intaltop_i!.log!(intaltop_i!.Stop).\ FEdit \\
&\quad + \Sigma_i nonaltop_i?.\ FEdit\ + exit?.outfile!.nil + fail!.nil \\
Updater &= \Sigma_i intaltop_i?.Updater \\
Demon &= fail?.restart!.Demon \\
Logsys &= Emptylog \mid UpdateLog \mid Restartlog \\
Emptylog &= Log(Stop) \\
Log(x) &= writelog?y.Log(y) + readlog!x.Log(x) \\
Updatelog &= log?x.readlog?y.writelog!(y\ before\ x).Updatelog \\
Restartlog &= restart?.readlog?x.(x\ before\ (\ FEdit\ \mid Restartlog)) \\
Stop &= d!.nil \\
x\ before\ y &= (x \mid d?.y)\backslash d
\end{aligned}
$$

where $\backslash intops$ is shorthand for $\backslash intop_1 \ldots \backslash intop_n \backslash readlog \backslash writelog \backslash restart \backslash fail$.

We have simplified the description by considering the log system as part of the editor system. A more elaborate version would allow the user to restart the system and faults to occur at any level of interaction. We have specified both *PEditor* and *FEditor* using a recursive definition. This is justified by the simulation of recursion results of section 2.6.

An interesting point to note about the above system is how the log system collects a program by piecing together a sequential program consisting of each command typed in by the user of the editor. This program can then be run in the event of a fault occurring.

We can prove that the ideal editor and the fault tolerant editor are observational equivalent and we may thus regard the ideal editor as a specification and the fault tolerant editor as an implementation. To see this observe that the following relation is a weak higher order bisimulation:

$$
\begin{aligned}
R = \{&(nil\ ,\ nil\backslash intops), \\
&(outfile!.nil\ ,\ (outfile!.nil)\backslash intops),
\end{aligned}
$$

$$
\begin{aligned}
&(\ PEditor\ ,\ \ FEditor\ ),\\
&\quad (\ PEdit\ ,\ (\ FEdit\ \mid Logsys\\
&\qquad \mid Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (intaltop_i!.log!(intaltop_i!.Stop).\ FEdit\\
&\qquad \mid Logsys \mid Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (log!(intaltop_i!.Stop).\ FEdit\\
&\qquad \mid Logsys \mid Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (\ FEdit\ \mid (Log(curstat)\\
&\qquad \mid readlog?y.writelog!(y\ before\ (intaltop_i!.Stop)).Updatelog\\
&\qquad \mid Restartlog) \mid Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (\underbrace{(nil \mid \ldots \mid nil)}_{m} \mid Logsys\\
&\qquad \mid restart!.Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (\underbrace{(nil \mid \ldots \mid nil)}_{m} \mid (Log(curstat) \mid Updatelog\\
&\qquad \mid readlog?x.(x\ before\ \ FEdit\ )) \mid Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (\underbrace{(nil \mid \ldots \mid nil)}_{m} \mid (Log(curstat) \mid Updatelog\\
&\qquad \mid (curstat\ before\ (\ FEdit\ \mid Restartlog)))\\
&\qquad \mid Demon \mid Updater)\backslash intops),\\
&\quad (\ PEdit\ ,\ (\underbrace{(nil \mid \ldots \mid nil)}_{m} \mid (Log(curstat) \mid Updatelog\\
&\qquad \mid\ FEdit\ \mid Restartlog) \mid Demon \mid Updater)\backslash intops)\}
\end{aligned}
$$

where $m > 0$ and m denotes the number of $fail!$. signals which have occurred so far in the execution of *FEditor* and *curstat* is either *Stop*, corresponding to *FEditor* being in its initial state or *curstat* is a sequence $(intaltop_i!.Stop\ before\ previousstat)$ where *previousstat* is a sequence like *curstat*. For presentation purposes we have omitted intermediate states caused by the *before* construct. These can be added but they clutter the presentation unnecessarily. Thus $PEditor \approx FEditor$. However, $FEditor \sqsubseteq PEditor$ but $PEditor \not\sqsubseteq FEditor$ since $PEditor\Downarrow$ whereas $FEditor\uparrow$. The fault tolerant editor *FEditor* may do an infinite sequence of internal actions due to $fail!$ signals occurring infinitely often.

Readers familiar with the studies of restartable systems presented in [Pra88] will notice that *FEditor* resembles the systems studied by Prasad in section 4.1.1 of his thesis. But note that we do not need to parameterize the state of the system

by pairs of the initial state and current state, we only need to program a memory system in CHOCS (or reuse the memory cells defined in the previous section), store each command as they arrive from the user bit by bit and read this program from the memory and run it whenever we need to restart the system.

The faults, signalized by $fail!$ signals, are simplified in the above system. They only occur when the editor is in a state where it is ready to receive instructions from the user. A more refined version could allow faults to occur at any level. We could specify this by adding a $fail!.nil$ process after any guard e.g. use:

$$\Sigma_i altop_i?.(intaltop_i!.(log!(intaltop_i!.Stop).\ FEdit\ + fail!.nil) + fail!.nil)$$

or we could use the displace operator $\leftarrow\!\!+$ from [Pra88] and simply exchange the $+fail!.nil$ component of $FEdit$ by $\leftarrow\!\!+fail!.nil$. The introduction of faults at other levels calls for a more elaborate protocol for adding to the memory of the current state of the editor to ensure that the last instruction to the editor does not get lost between the internal updating and the storage of the instruction. We shall not elaborate further on this but leave it for future studies. For a discussion of fault tolerant systems in general we refer to [Pra88] where a thorough discussion of their description in CCS is presented.

Chapter 4

Denotational Theory of CHOCS

So far we have only studied the semantics of CHOCS from an operational point of view. In this chapter we construct a denotational semantics for the language. One of the main benefits of denotational semantics is its compositional nature which enables a compositional way of reasoning about CHOCS processes. The denotational semantics also highlights certain features of the operational semantics as we shall see in the latter part of this chapter.

We begin this chapter by reviewing the definitions and the results from Domain Theory upon which we build a denotational semantics for CHOCS. In section 4.2 we define a domain equation in which the denotational semantics for CHOCS will reside. The main result of this section is an "internal full abstraction" result showing that if we impose a labelled transition system view on the domain equation the operational preorder of higher order prebisimulation and the domain ordering coincide. As a step towards defining a denotational semantics for CHOCS we define a denotational semantics for finite processes in section 4.3. The main theorem of this section is the full abstraction result for finite processes in theorem 4.3.5. Most of the results in section 4.2 and 4.3 are built on section 3 and 6 of [Abr90a] and are mainly adaptations of the Domain Equation for Synchronization Trees and the Full Abstraction for SCCS to the setting of Higher Order Communication Trees and CHOCS. We define a denotational semantics for all CHOCS processes and extend the full abstraction result in section 4.4. The main theorem of this section is the (limited) full abstraction result presented in theorem 4.4.7. The result is limited to the case where the set of port names *Names* is assumed to be finite and is stated in terms of the preorder $\sqsubseteq_\omega$. These restrictions are due to the well known impossibility of modelling unbounded nondeterminism in the Plotkin Power Domain. Finally in section 4.5 we use the denotational semantics to obtain a very simple proof of the simulation of recursion result from section 2.6.

4.1 Domains and Denotational Semantics

In this section we present a short review of the definitions and results from Domain Theory upon which we build a denotational semantics for CHOCS. It is hard (though possible, see e.g. [Sch86]) to give an overview of this subject without using a bit of category theory jargon and we shall do so freely.

The denotational semantics we are going to construct will reside in a domain of an appropriate "shape". This domain will be an object of the category **CPO** or rather the subcategory **SFP**. The category **CPO** is a very important "tool" for making meanings of programs and programming languages. It has been extensively studied in [Plo81b] and this reference together with section 2.2 of [Nie84] have been our main sources for this section. Two important characteristics of the category **CPO** is that it admits recursive definitions both of its elements and of domains themselves. It also supports a rich type structure and we present the type constructions that we use in this thesis in the latter part of this section.

First we review a few of the basic definitions from Domain Theory:

Definition 4.1.1 *A relation $\sqsubseteq$ on a set D is a partial ordering upon D iff $\sqsubseteq$ is: reflexive, antisymmetric and transitive.*

We call a set with a partial ordering a partially ordered set.

Definition 4.1.2 *An element e of a partially ordered set D is a least element (or a bottom element) denoted $\bot$ iff $e \sqsubseteq d$ for all $d \in D$.*

Definition 4.1.3 *If it exists; a join $a \sqcup b$ of two elements a and b in a partially ordered set D is an element such that:*

1. *$a \sqsubseteq a \sqcup b$ and $b \sqsubseteq a \sqcup b$*
2. *for all $d \in D$: $a \sqsubseteq d$ and $b \sqsubseteq d$ implies $a \sqcup b \sqsubseteq d$*

Definition 4.1.4 *For $X \subseteq D$ a subset of a partially ordered set D a least upper bound $\bigsqcup X$ denotes the element of D such that:*

1. *for all $x \in X$ we have $x \sqsubseteq \bigsqcup X$*
2. *for all $d \in D$ if $x \in X$ and $x \sqsubseteq d$ then $\bigsqcup X \subseteq d$*

Definition 4.1.5 *A subset X of a partially ordered set D is a chain if*

1. *X is nonempty*

2. *for all* $a, b \in X$ *either* $a \sqsubseteq b$ *or* $b \sqsubseteq a$

Often the elements of a chain are enumerated and the chain is denoted by $\{d_n\}_n$.

Definition 4.1.6 *A function* $f : D \to E$ *from a partially ordered set* D *to a partially ordered set* E *is monotone iff* $d_1 \sqsubseteq d_2 \Rightarrow f(d_1) \sqsubseteq f(d_2)$.

Definition 4.1.7 *A monotone function* $f : D \to E$ *from a partially ordered set* D *to a partially ordered set* E *is continuous iff for any chain* $X \subseteq D$: $f(\bigsqcup X) = \bigsqcup\{f(x) : x \in X\}$.

Definition 4.1.8 *A partially ordered set is a complete partially ordered set (or a cpo) iff* D *has a least element* $\perp$ *and has least upper bounds for all chains.*

Definition 4.1.9 *Let* D *be a cpo. An element* $b \in D$ *is compact or finite if, whenever* $X \subseteq D$ *is a chain and* $b \sqsubseteq \bigsqcup X$ *then* $b \sqsubseteq d$ *for some* $d \in X$. *We write* $K(D)$ *for the set of finite elements of* D.

Definition 4.1.10 *A function* $f : D \to E$ *from a cpo* D *to a cpo* E *is strict iff* $f(\perp) = \perp$.

A functional[1] is a continuous function $f : D \to D$; usually D is a domain of the form $A \to B$, but it does not have to be.

Definition 4.1.11 *An element* d *of a partially ordered set* D *is a fixed point for a functional* $F : D \to D$ *if* $F(d) = d$. *A fixed point* d *for a functional* F *is a least fixed point iff for all fixed points* e *for* F *we have* $d \sqsubseteq e$.

Theorem 4.1.12 *For any functional* $F : D \to D$ *on a cpo there exists a least fixed point* `fix` F *given by*

$$\texttt{fix}\ F = \bigsqcup\{F^i(\perp) : i \geq 0\}$$

where $F^0(\perp) = \perp$ *and* $F^{i+1} = F(F^i(\perp))$.

Definition 4.1.13 *The category* **CPO** *is the category whose objects are complete partial ordered sets (cpo's) with continuous functions as morphisms.*

[1]This terminology is used in [Sch86]. As pointed out to me by S. Abramsky this is non-standard terminology

Formally a category consists of a set of objects and for each two objects D_1 and D_2 a set of morphisms $f : D_1 \rightarrow D_2$. There has to be a composition of morphisms as in $f \circ g : D_1 \rightarrow D_3$ which is the composition of $f : D_2 \rightarrow D_3$ and $g : D_1 \rightarrow D_2$. For each object D there has to be an identity morphism $Id_D : D \rightarrow D$. Composition and identity have to satisfy the associative law: $(f \circ g) \circ h = f \circ (g \circ h)$ respectively the identity laws: $f \circ Id_D = f$ and $Id_D \circ f = f$. It is easily checked that these criteria are satisfied for **CPO** [Plo81b].

Analogous to the notion of morphisms between objects of categories we may consider "functions" between categories. These are called functors:

Definition 4.1.14 *A functor F consists of two maps: one from objects D of category $\mathbf{D}$ to objects $F(D)$ of category $\mathbf{E}$ and one that sends a morphism $f : D_1 \rightarrow D_2$ of category $\mathbf{D}$ to a morphism $F(f) : F(D_1) \rightarrow F(D_2)$ of category $\mathbf{E}$. It must satisfy the composition law: $F(f \circ g) = F(f) \circ F(g)$ and the identity law: $F(Id_D) = Id_{F(D)}$.*

Functors are extensively used in the definition of (recursive) domain equations. To do so we may draw upon an analogy between cpo's and categories. A cpo-category is a category where the set of morphisms between any two objects forms a cpo and where composition is continuous with respect to the partial order.

Definition 4.1.15 *A functor between two cpo-categories is locally continuous (monotonic) if its effect upon morphisms is continuous (monotonic). Continuity for a functor F means $F(\bigsqcup_n f_n) = \bigsqcup_n F(f_n)$ for all chains $\{f_n\}_n$ of continuous functions.*

For a continuous functor the analogy to a fixed point of a continuous function on a cpo is captured by:

Definition 4.1.16 *The pair (D, Φ) is a fixed point for a continuous functor F over some category iff D is an object and $\Phi : F(D) \rightarrow D$ is an isomorphism. (An isomorphism Φ in a category is a morphism $\Phi : D \rightarrow E$ for which there exists an inverse $\Phi^{-1} : E \rightarrow D$ such that $\Phi \circ \Phi^{-1} = Id_D$ and $\Phi^{-1} \circ \Phi = Id_E$*

Definition 4.1.17 *A pair (D, Φ) is an initial fixed point of a continuous functor F over some cpo-category iff it is a fixed point and for every fixed point (D', Φ') there exists precisely one embedding $e : D \rightarrow D'$ such that $e \circ \Phi = \Phi' \circ F(e)$.*

An embedding in a cpo-category is a morphism $e : D_1 \rightarrow D_2$ for which there exists another morphism $e^u : D_2 \rightarrow D_1$ such that $e^u \circ e = Id_{D_1}$ and $e \circ e^u \sqsubseteq Id_{D_2}$.

There are constructs analogous to the notions of chain and least upper bounds, these are called directed sequences and limiting cones:

Definition 4.1.18 *A directed sequence is a pair* $(\{D_n\}_n, \{e_n\}_n)$ *where each* D_n *is an object and each* $e_n : D_n \to D_{n+1}$ *is a morphism.*

Definition 4.1.19 *A limiting cone, for a directed sequence* $(\{D_n\}_n, \{e_n\}_n)$*, is a pair* $(D, \{r_n\}_n)$ *where* D *is an object and each* r_n *is a morphism satisfying* $r_n : D_n \to D$ *and* $r_{n+1} \circ e_n = r_n$*. (*D *is called the limit of the directed sequence)*

We may use this to define a construction `FIX` F for a continuous functor, corresponding to `fix` F for a continuous function.

Definition 4.1.20

$$\texttt{FIX}\ F = (\{D_n\}_n, \{e_n\}_n)$$

where $D_0 = U$ *(the one point domain* $\{\bot\}$*) and* $D_{n+1} = F(D_n)$*.* $e_0 = \bot$ *and* $e_{n+1} = F(e_n)$*.*

In the subcategory **CPO-E** of **CPO** where all functions are embeddings this construct generates a cone. Let $(D, \{r_n\}_n)$ be the limiting cone for `FIX` F then (D, Φ) where $\Phi = \bigsqcup_n r_{n+1} \circ F(r_n)^u$ is an initial fixed point of F.

However, initial fixed points for continuous functors need not be unique, though they are all isomorphic, but it is common to say the initial fixed point about the fixed point generated for `FIX` F.

In denotational semantics it is quite common to write domains as recursive definitions like $D = F(D)$ involving functors such as F. With the above machinery we may solve such equations. A solution to recursive domain equation is simply taken to be the initial fixed point for its defining functor. This means that we only solve the domain equation up to isomorphism since the initial fixed point is defined in terms of isomorphisms and we write $D \cong F(D)$ to emphasize this.

The domain equation we present in the next section will use the Plotkin Power Domain and we therefore need to work in a category closed under this construction. We will use **SFP** [Plo76, Plo81b].

Definition 4.1.21 *The category* **SFP** *(Sequences of Finite Posets) has as objects those cpo's* D *which are limits of directed sequences* $(\{D_n\}_n, \{p_n\}_n)$ *of finite cpo's* D_m*. Its morphisms are the continuous functions with the usual composition.*

The only thing we need to know about **SFP**, apart from the above, is that it admits recursive domain equations and it is closed under the constructs of Cartesian product, separated sum and the Plotkin Power Domain.

Cartesian Product

Let D and D' be domains. The Cartesian product $D \times D'$ is the domain of pairs of elements with first component from D and second component from D'. We write (d, d') for elements of $D \times D'$ where $d \in D$ and $d' \in D'$. $D \times D'$ is ordered component wise i.e.:

$$(d_1, d_1') \sqsubseteq_{D \times D'} (d_2, d_2') \Leftrightarrow d_1 \sqsubseteq_D d_2 \;\&\; d_1' \sqsubseteq_{D'} d_2'$$

We may turn the Cartesian product into a functor:
Given

$$f : D_1 \to D_1' \text{ and } g : D_2 \to D_2'$$

then

$$\times(f, g) : D_1 \times D_2 \to D_1' \times D_2'$$

is given by

$$\times(f, g)(d_1, d_2) = (f d_1, g d_2)$$

Separated Sum

Let A be a countable set and $\{D_a\}_{a \in A}$ be a family of A–indexed domains. The separated sum $\sum_{a \in A} D_a$ is the domain formed by the disjoint union of the D_a's and adjoining a bottom element. We write $\langle a, d \rangle$ for the elements of the disjoint union and $\perp$ for the bottom element of the separated sum. We order the domain as follows:

$$x \sqsubseteq_{(\sum_{a \in A} D_a)} y \Leftrightarrow x = \perp \text{ or } [x = \langle a, d \rangle \;\&\; y = \langle a, d' \rangle \;\&\; d \sqsubseteq_{D_a} d']$$

Separated sum may be treated as a functor:
Given a family of functions:

$$f_a : D_a \to E_a$$

Then

$$\sum_{a \in A} f_a : \sum_{a \in A} D_a \to \sum_{a \in A} E_a$$

is defined by:

$$\begin{aligned} (\sum_{a \in A} f_a)\, \perp &= \perp \\ (\sum_{a \in A} f_a)\langle a, d \rangle &= \langle a, f_a d \rangle \end{aligned}$$

Note that for each $a \in A$, the function:

$$\begin{array}{rcl} D_a & \rightarrow & \sum_{a \in A} D_a \\ d & \longmapsto & \langle a, d \rangle \end{array}$$

is continuous.

The Plotkin Power Domain

Let D be a domain. We say that a subset $X \subseteq D$ is closed if $X = X^*$ where $X^* = Con \circ Cl(X)$ and $Con(X) = \{d : \exists d_1, d_2 \in X. d_1 \sqsubseteq d \sqsubseteq d_2\}$ and Cl is the closure operation associated with the Lawson Topology (see [Plo81b]). The Plotkin Power Domain $P[D]$ over D is defined as the set of nonempty closed subsets of D. The elements of $P[D]$ are given by $\{X \subseteq D : X \neq \emptyset, X = X^*\}$. The Plotkin Power Domain $P[D]$ over D is ordered by the Egli–Milner order:

$$X \sqsubseteq_{EM} Y \Leftrightarrow \forall x \in X. \exists y \in Y. x \sqsubseteq y \;\&\; \forall y \in Y. \exists x \in X. x \sqsubseteq y$$

We shall make use of a number of continuous operations associated with the Plotkin Power Domain:
P is functorial:
Given

$$f : D \rightarrow E$$

then

$$Pf : P[D] \rightarrow P[E]$$

is defined by

$$Pf(X) = \{f(x) : x \in X\}^*$$

Other useful operations are:
Singleton:

$$\{\!|\cdot|\!\} : D \rightarrow P[D]$$

defined by:

$$\{\!|\, d \,|\!\} = \{d\}^* = \{d\}$$

Union:

$$\uplus : P[D]^2 \rightarrow P[D]$$

defined by

$$X \uplus Y \equiv (X \uplus Y)^* = Con(X \cup Y)$$

Big Union:

$$\biguplus : P[P[D]] \to P[D]$$

defined by

$$\biguplus(\Theta) \equiv (\bigcup \Theta)^\star = Con(\bigcup \Theta)$$

and Tensor Product:
Given

$$f : D^n \to D$$

Then

$$f^\dagger P[D]^n \to P[D]$$

is defined by

$$f^\dagger(X_1 \ldots X_n) = \{f(x_1 \ldots x_n) \ : \ x_i \in X_i\}^\star$$

The Tensor Product has the property:

$$f^\dagger(X_1 \ldots X_i \uplus X_i' \ldots X_n) = f^\dagger(X_1 \ldots X_i \ldots X_n) \uplus f^\dagger(X_1 \ldots X_i' \ldots X_n)$$

for $(1 \leq i \leq n)$. For $n = 1$ we have $f^\dagger = Pf$.

4.2 A Domain Equation for Higher Order Communication Trees

With the machinery from the previous section in hand we are now ready to construct a Domain in which the denotational semantics for CHOCS will reside. As in [Abr90a] we shall use the Plotkin Power Domain with the empty set adjoined. We use the empty set to denote the process *nil* (i.e. the convergent process with no action). The empty set is added to the Plotkin Power Domain without being related to anything but itself under the Egli-Milner ordering and we write $P^0[D]$ for the Plotkin Power Domain over D with the empty set adjoined. The elements of $P^0[D]$ are given by $\{X \subseteq D \ : \ X = X^\star\} = P[D] \cup \{\emptyset\}$ with the ordering:

$$X \sqsubseteq Y \Leftrightarrow X = \{\bot\} \text{ or } X \sqsubseteq_{EM} Y$$

The operations on $P[D]$ described above may be extended to $P^0[D]$, and $\uplus$, $\biguplus$ and $\{\!|\ \cdot\ |\!\}$ are continuous on $P^0[D]$. For $P^0 f$ to work we need to assume that f is strict and for $f^\dagger$ to work we need to assume that f is strict in each argument. We write $\{\!|\ d \mid A\ |\!\}$ where $d \in D$ and A is some sentence, meaning $\{\!|\ d\ |\!\}$ if A is true, and $\emptyset$ otherwise.

Definition 4.2.1 *Let Names (the set of port names) be a countable set and let* $Ev = Names \times \{!, ?\} \cup \{\tau\}$ *(ranged over by e). Then D, the domain of higher order communication trees, is defined as the initial solution of the domain equation:*

$$D \cong P^0[\sum_{e \in Ev} D_e]$$

where $D_{a?} \equiv D \times D$, $D_{a!} \equiv D \times D$ *and* $D_\tau \equiv D$.

This domain equation is essentially that of [Abr90a] with the structure of actions taken into account. We write $\perp$ for the bottom element of $\sum_{e \in Ev} D_e$ and $\{\!|\perp|\!\}$ for the bottom element of $P^0[\sum_{e \in Ev} D_e]$.

The structure of D is recursive and may be unpacked by the following two parts:

1. Let Φ and Φ^{-1} be a specified isomorphism pair such that:

$$D \underset{\Phi}{\overset{\Phi^{-1}}{\rightleftarrows}} P^0[\sum_{e \in Ev} D_e]$$

   We shall treat $D \cong P^0[\sum_{e \in Ev} D_e]$ as identity and thus elide the use of Φ and Φ^{-1}. They can be put back in without any difficulties, but they will clutter the presentation.

2. Initiality. As described in the following.

Definition 4.2.2 *We define a sequence of functions:*

$$\pi_k : D \to D$$

as follows:

$$\begin{aligned} \pi_0 &\equiv \lambda x.\{\!|\perp|\!\} \\ \pi_{k+1} &\equiv P^0 \sum_{e \in Ev} f_e \end{aligned}$$

Where $f_{a?} = \times(\pi_k, \pi_k)$, $f_{a!} = \times(\pi_k, \pi_k)$ *and* $f_\tau = \pi_k$.

D is the "internal colimit" of the π_k i.e.:

Proposition 4.2.3 *The following properties hold:*

(i) Each π_k *is continuous and* $\pi_k \sqsubseteq \pi_{k+1}$

(ii) $\bigsqcup_k \pi_k = id_D$

(iii) $\pi_k \circ \pi_k = \pi_k$

(iv) $\forall d_1, d_2 \in D. d_1 \sqsubseteq d_2 \Leftrightarrow \forall k. \pi_k d_1 \sqsubseteq \pi_k d_2$

We may think of elements d of D as (finite and/or infinite) trees. $\pi_k d$ cuts the tree to a depth of k. $\pi_k D$ is the set of all trees with depth at most k.

The following definition gives an inductive definition of the set of elements d of D which only have depth k:

Definition 4.2.4

$$\begin{aligned} LEV_0 &= \{|\perp|\} \\ LEV_{k+1} &= (\{\langle a?, (d_1', d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in LEV_k\} \\ &\quad \cup \{\langle a!, (d_1', d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in LEV_k\} \\ &\quad \cup \{\langle \tau, d_1''\rangle \ : \ d_1'' \in LEV_k\})^\star \\ &\quad \uplus LEV_k \end{aligned}$$

Proposition 4.2.5

$$\forall k. LEV_k = \pi_k D$$

PROOF: By induction on k:

$\underline{k = 0}$

$$LEV_0 = \{|\perp|\} = \pi_0 D$$

$\underline{k+1}$

$$\begin{aligned} LEV_{k+1} &= (\{\langle a?, (d_1', d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in LEV_k\} \\ &\quad \cup \{\langle a!, (d_1', d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in LEV_k\} \\ &\quad \cup \{\langle \tau, d_1''\rangle \ : \ d_1'' \in LEV_k\})^\star \\ &\quad \uplus LEV_k \\ &= (\{\langle a?, (d_1', d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in \pi_k D\} \\ &\quad \cup \{\langle a!, (d_1', d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in \pi_k D\} \\ &\quad \cup \{\langle \tau, d_1''\rangle \ : \ d_1'' \in \pi_k D\})^\star \\ &\quad \uplus \pi_k D \\ &\quad \text{by the induction hypothesis} \\ &= (\{\langle a?, (\pi_k d_1', \pi_k d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in D\} \end{aligned}$$

$$\begin{aligned} & \cup\{\langle a!, (\pi_k d_1', \pi_k d_1'')\rangle \ : \ a \in Names, d_1', d_1'' \in D\} \\ & \cup\{\langle \tau, \pi_k d_1''\rangle \ : \ d_1'' \in D\})^\star \\ & \uplus \pi_k D \\ = \ & \pi_{k+1} D \end{aligned}$$

Corollary 4.2.6 □

$$D = \bigsqcup_k LEV_k$$

The elements of each of the LEV_k's are compact elements of D. We may give an explicit description of the compact elements of D.

Definition 4.2.7 *We define $K(D) \subseteq D$ inductively:*

- $\emptyset \in K(D)$
- $\{\!|\bot|\!\} \in K(D)$
- $a \in Names, d_1, d_2 \in K(D) \ \Rightarrow \ \{\!|\ \langle a?, d_1, d_2\rangle\ |\!\} \in K(D)$
  $\{\!|\ \langle a!, d_1, d_2\rangle\ |\!\} \in K(D)$
  $\{\!|\ \langle \tau, d_1\rangle\ |\!\} \in K(D)$
  $d_1 \uplus d_2 \in K(D)$

Proposition 4.2.8 *$K(D)$ is exactly the set of compact elements of D.*

PROOF: Follows from standard results (see [Plo76, Plo81b]). □

Proposition 4.2.9 $(\bigcup\{LEV_k \ : \ k \geq 0\})^\star = K(D)$.

PROOF: For each k: $LEV_k \subseteq K(D)$ follows from definition 4.2.4, definition 4.2.7 and proposition 4.2.8 The opposite direction follows by showing that each element of $K(D)$ is an element of LEV_k for some k. This is done by induction on the construction of elements of $K(D)$:

Base cases $\emptyset \subseteq LEV_k$ for all k and $\{\!|\bot|\!\} \subseteq LEV_k$ for all k.

Inductive step Assume $d_1, d_2 \in K(D)$ and $d_1, d_2 \in LEV_k$ for some k. Then:

$$\begin{aligned} \{\!|\ \langle a?, d_1, d_2\rangle\ |\!\} \ & \in \ LEV_{k+1} \\ \{\!|\ \langle a!, d_1, d_2\rangle\ |\!\} \ & \in \ LEV_{k+1} \\ \{\!|\ \langle \tau, d_1\rangle\ |\!\} \ & \in \ LEV_{k+1} \\ d_1 \uplus d_2 \ & \in \ LEV_k \subseteq LEV_{k+1} \end{aligned}$$

□

D as a Transition System

Consider D as a transition system $(D, Act, \rightarrow, \uparrow)$:

- $d\uparrow \equiv \bot \in d$
- $d \xrightarrow{a?d'} d'' \equiv \langle a?, (d', d'')\rangle \in d$
- $d \xrightarrow{a!d'} d'' \equiv \langle a!, (d', d'')\rangle \in d$
- $d \xrightarrow{\tau} d' \equiv \langle \tau, d'\rangle \in d$

We can now show that D is "internally fully abstract" i.e.:

Proposition 4.2.10

$$\forall d_1, d_2 \in D. d_1 \lesssim^B d_2 \iff d_1 \sqsubseteq d_2$$

PROOF: The proof of this proposition follows the pattern of the proof of proposition 3.11 in [Abr90a]. We shall prove:

(i) $\forall k. d_1 \lesssim_k d_2 \Rightarrow \pi_k d_1 \sqsubseteq \pi_k d_2$

(ii) $\sqsubseteq \subseteq \lesssim^B$

Clearly (i) implies $\lesssim_\omega \subseteq \sqsubseteq$ and since $\lesssim^B \subseteq \lesssim_\omega$ we have $\lesssim^B = \sqsubseteq$.
To see that (i) holds we proceed by induction on k.

$\underline{k=0}$ Since $d_1 \lesssim_0 d_2$ always holds and clearly $\pi_0 d_1 \sqsubseteq \pi_0 d_2$ holds, and we have $d_1 \lesssim_0 d_2 \Rightarrow \pi_0 d_1 \sqsubseteq \pi_0 d_2$. This establishes the base case.

$\underline{k+1}$ Assume $d_1 \lesssim_{k+1} d_2$. Now $d_1 = \emptyset$ and $d_1 \lesssim_{k+1} d_2$ implies $d_2 = \emptyset$. $d_1 = \{\!|\bot|\!\}$ implies $d_1 \sqsubseteq d_2$ so we may assume that $d_1 \neq \emptyset \neq d_2$. It is sufficient to prove $d_1 \sqsubseteq_{EM} d_2$.

We have $\pi_{k+1} d_1 = X^*$ where

$$\begin{aligned} X \;=\; & \{\langle a?, (\pi_k d_1', \pi_k d_1'')\rangle \;:\; \langle a?, (d_1', d_1'')\rangle \in d_1\} \cup \\ & \{\langle a!, (\pi_k d_1', \pi_k d_1'')\rangle \;:\; \langle a!, (d_1', d_1'')\rangle \in d_1\} \cup \\ & \{\langle \tau, \pi_k d_1''\rangle \;:\; \langle \tau, d_1''\rangle \in d_1\} \cup \\ & \{\bot : \bot \in d_1\} \end{aligned}$$

and similarly $\pi_{k+1} d_2 = Y^*$. Now

- $\langle a?,(d_1',d_1'')\rangle \in X$

$\Rightarrow d_1 \xrightarrow{a?d_1'} d_1''$

$\Rightarrow \exists d_2', d_2''.d_2 \xrightarrow{a?d_2'} d_2''$ & $d_1' \leftrightarrows_k d_2'$ & $d_1'' \leftrightarrows_k d_2''$

$\Rightarrow \exists d_2', d_2''.d_2 \xrightarrow{a?d_2'} d_2''$ & $\pi_k d_1' \sqsubseteq \pi_k d_2'$ & $\pi_k d_1'' \sqsubseteq \pi_k d_2''$ by the induction hypothesis

$\Rightarrow \exists \langle a?,(\pi_k d_2', \pi_k d_2'')\rangle \in Y.\langle a?,(\pi_k, d_1', \pi_k d_1'')\rangle \sqsubseteq \langle a?,(\pi_k, d_2', \pi_k d_2'')\rangle$

and similarly for $\langle a!,(\pi_k, d_1', \pi_k d_1'')\rangle \in X$ and we have

- $\langle \tau, d_1'\rangle \in X$

$\Rightarrow d_1 \xrightarrow{\tau} d_1'$

$\Rightarrow \exists d_2'.d_2 \xrightarrow{\tau} d_2'$ & $d_1' \leftrightarrows_k d_2'$

$\Rightarrow \exists d_2'.d_2 \xrightarrow{\tau} d_2'$ & $\pi_k d_1' \sqsubseteq \pi_k d_2'$ by the induction hypothesis

$\Rightarrow \exists \langle \tau, \pi_k d_2'\rangle \in Y.\langle \tau, \pi_k, d_1'\rangle \sqsubseteq \langle \tau, \pi_k, d_2'\rangle$

Also

- $\perp \notin X$

$\Rightarrow \perp \notin d_1$

$\Rightarrow \perp \notin d_2$ & $[[d_2 \xrightarrow{a?d_2'} d_2'' \Rightarrow \exists d_1', d_1''.d_1 \xrightarrow{a?d_2'} d_1''$ & $d_1' \leftrightarrows_k d_2'$ & $d_1'' \leftrightarrows_k d_2'']$
& $[d_2 \xrightarrow{a!d_2'} d_2'' \Rightarrow \exists d_1', d_1''.d_1 \xrightarrow{a!d_1'} d_1''$ & $d_1' \leftrightarrows_k d_2'$ & $d_1'' \leftrightarrows_k d_2'']$
& $[d_2 \xrightarrow{\tau} d_2' \Rightarrow \exists d_1', d_1''.d_1 \xrightarrow{\tau} d_1'$ & $d_1' \leftrightarrows_k d_2']]$

$\Rightarrow \perp \notin Y.$ & $[[\forall \langle a?,(\pi_k d_2', \pi_k d_2'')\rangle \in Y.$
$\exists \langle a?,(\pi_k d_1', \pi_k d_1'')\rangle \in X.\pi_k d_1' \sqsubseteq \pi_k d_2'$ & $\pi_k d_1'' \sqsubseteq \pi_k d_2'']$
& $[\forall \langle a!,(\pi_k d_2', \pi_k d_2'')\rangle \in Y.$
$\exists \langle a!,(\pi_k d_1', \pi_k d_1'')\rangle \in X.\pi_k d_1' \sqsubseteq \pi_k d_2'$ & $\pi_k d_1'' \sqsubseteq \pi_k d_2'']$
& $[\forall \langle \tau, \pi_k d_2'\rangle \in Y.\exists \langle \tau, \pi_k d_1'\rangle \in X.\pi_k d_1' \sqsubseteq \pi_k d_2']]$

by the induction hypothesis

Furthermore we have shown $X \sqsubseteq_{EM} Y$ which implies $X^* \sqsubseteq_{EM} Y^*$.

To see (ii) we show that $\sqsubseteq$ is a higher order bisimulation. Observe that:

- $d_1 \sqsubseteq d_2$

$\Rightarrow$ $[[\forall \langle a?,(d_1', d_1'')\rangle \in d_1.\exists \langle a?,(d_2', d_2'')\rangle \in d_2.d_1' \sqsubseteq d_2'$ & $d_1'' \sqsubseteq d_2'']$

$$
\begin{array}{l}
\quad \& \ [\forall \langle a!,(d_1',d_1'')\rangle \in d_1.\exists \langle a!,(d_2',d_2'')\rangle \in d_2.d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \\
\quad \& \ [\forall \langle \tau, d_1'\rangle \in d_1.\exists \langle \tau, d_2'\rangle \in d_2.d_1' \sqsubseteq d_2'] \\
\quad \& \ \ \bot \notin d_1 \Rightarrow \bot \notin d_2 \ \ \& \ [\forall \langle a?,(d_2',d_2'')\rangle \in d_2.\exists \langle a?,(d_1',d_1'')\rangle \in d_1. \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \& \ [\forall \langle a!,(d_2',d_2'')\rangle \in d_2.\exists \langle a!,(d_1',d_1'')\rangle \in d_1. \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \& \ [\forall \langle \tau, d_2'\rangle \in d_2.\exists \langle \tau, d_1'\rangle \in d_1.d_1' \sqsubseteq d_2']]
\end{array}
$$

$$
\begin{array}{lll}
\Rightarrow & \forall a \in Names. & [[d_1 \xrightarrow{a?d_1'} d_1'' \Rightarrow \exists d_2', d_2''.d_2 \xrightarrow{a?d_2'} d_2'' \ \&\ d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \ \& \\
 & & [d_1 \xrightarrow{a!d_1'} d_1'' \Rightarrow \exists d_2', d_2''.d_2 \xrightarrow{a!d_2'} d_2'' \ \&\ d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \ \& \\
 & & [d_1 \xrightarrow{\tau} d_1' \Rightarrow \exists d_2'.d_2 \xrightarrow{\tau} d_2' \ \&\ d_1' \sqsubseteq d_2'] \ \& \\
d_1 \downarrow \Rightarrow & d_2 \downarrow \ \& & [d_2 \xrightarrow{a?d_2'} d_2'' \Rightarrow \exists d_1', d_1''.d_1 \xrightarrow{a?d_1'} d_1'' \ \&\ d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \ \& \\
 & & [d_2 \xrightarrow{a!d_2'} d_2'' \Rightarrow \exists d_1', d_1''.d_1 \xrightarrow{a!d_1'} d_1'' \ \&\ d_1' \sqsubseteq d_2' \ \&\ d_1'' \sqsubseteq d_2''] \ \& \\
 & & [d_2 \xrightarrow{\tau} d_2' \Rightarrow \exists d_1'.d_1 \xrightarrow{\tau} d_1' \ \&\ d_1' \sqsubseteq d_2']]
\end{array}
$$

□

This result shows that the denotational domain and the labelled transition system model are equally expressive. The above result is syntax free and therefore not compositional. We need syntax to introduce compositionality and this is the subject of the next section.

4.3 A Denotational Semantics for Finite CHOCS

In section 2.8 we saw that the syntax of Finite CHOCS induces a term algebra T_Σ. By standard results [Gog etal77] there exists a unique Σ–homomorphism $A[\![\ ]\!] : T_\Sigma \longrightarrow A$ for any Σ–algebra A.

We now use this result to form a denotational semantics $D[\![\ ]\!]$ in the domain D for finite CHOCS. It suffices to define each operation in Σ as a function of the appropriate arity over D.

Definition 4.3.1

$$
\begin{array}{rcl}
nil^D & \equiv & \emptyset \\
\Omega^D & \equiv & \{\!|\bot|\!\} \\
a?^D\_\_ & \equiv & \lambda(d_1,d_2) \in D \times D.\{\!|\ \langle a?,(d_1,d_2)\rangle\ |\!\} \\
a!^D\_\_ & \equiv & \lambda(d_1,d_2) \in D \times D.\{\!|\ \langle a!,(d_1,d_2)\rangle\ |\!\} \\
\tau^D\_ & \equiv & \lambda d \in D.\{\!|\ \langle \tau, d\rangle\ |\!\} \\
+^D & \equiv & \uplus
\end{array}
$$

Restriction:

$$\_\backslash a^D \equiv \mu F \in [D \to D].\biguplus \circ P^0(g_a F)$$

where $g_a : [D \to D] \to [\sum_{e \in E_V} D_e \to D]$ *is defined by*

$$\begin{aligned}
g_a F \perp &= \{\!|\perp|\!\} \\
g_a F\langle b?, (d_1, d_2)\rangle &= \begin{cases} \{\!|\ \langle b?, (d_1, F d_2)\rangle\ |\!\} & \text{if } b \neq a \\ \emptyset & \text{otherwise} \end{cases} \\
g_a F\langle b!, (d_1, d_2)\rangle &= \begin{cases} \{\!|\ \langle b!, (d_1, F d_2)\rangle\ |\!\} & \text{if } b \neq a \\ \emptyset & \text{otherwise} \end{cases} \\
g_a F\langle \tau, d\rangle &= \{\!|\ \langle \tau, F d\rangle\ |\!\}
\end{aligned}$$

Renaming:

$$\_[S]^D \equiv \mu F \in [D \to D].P^0(g_S F)$$

where $g_S : [D \to D] \to [\sum_{e \in E_V} D_e \to \sum_{e \in E_V} D_e]$ *is defined by*

$$\begin{aligned}
g_S F \perp &= \perp \\
g_S F\langle a?, (d_1, d_2)\rangle &= \langle S(a)?, (d_1, F d_2)\rangle \\
g_S F\langle a!, (d_1, d_2)\rangle &= \langle S(a)!, (d_1, F d_2)\rangle \\
g_S F\langle \tau, d\rangle &= \langle \tau, F d\rangle
\end{aligned}$$

Parallel Composition:

$$\_ |^D \_ \equiv \mu F \in [D \times D \to D].\biguplus \circ P^0(lF \uplus rF \uplus cF)$$

where $l, r, c : [D^2 \to D] \to [(\sum_{e \in E_V} D_e)^2 \to D]$ *are defined by*

$$\begin{aligned}
lF(x, \perp) = lF(\perp, x) &= \{\!|\perp|\!\} \\
lF(\langle a?, (d_1, d_2)\rangle, x) &= \{\!|\ \langle a?, (d_1, F(d_2, \{\!|\ x\ |\!\}))\rangle\ |\!\} \\
lF(\langle a!, (d_1, d_2)\rangle, x) &= \{\!|\ \langle a!, (d_1, F(d_2, \{\!|\ x\ |\!\}))\rangle\ |\!\} \\
lF(\langle \tau, d\rangle, x) &= \{\!|\ \langle \tau, F(d, \{\!|\ x\ |\!\})\rangle\ |\!\}
\end{aligned}$$

and

$$\begin{aligned}
rF(x, \perp) = rF(\perp, x) &= \{\!|\perp|\!\} \\
rF(x, \langle a?, (d_1, d_2)\rangle) &= \{\!|\ \langle a?, (d_1, F(\{\!|\ x\ |\!\}, d_2))\rangle\ |\!\} \\
rF(x, \langle a!, (d_1, d_2)\rangle) &= \{\!|\ \langle a!, (d_1, F(\{\!|\ x\ |\!\}, d_2))\rangle\ |\!\} \\
rF(x, \langle \tau, d\rangle) &= \{\!|\ \langle \tau, F(\{\!|\ x\ |\!\}, d)\rangle\ |\!\}
\end{aligned}$$

and

$$cF(x,y) = \begin{cases} \{\!|\perp|\!\} & \text{if } x = \perp \text{ or } y = \perp \\ \{\!|\ \langle\tau, F(d_2, d_2')\rangle\ |\!\} & \text{if } x = \langle a?,(d_1,d_2)\rangle \text{ and } y = \langle a!,(d_1,d_2')\rangle \\ & \text{or } x = \langle a!,(d_1,d_2)\rangle \text{ and } y = \langle a?,(d_1,d_2')\rangle \\ \emptyset & \text{otherwise} \end{cases}$$

We need to check that the above functions are well defined. This follows since all functions are strict (bistrict) and in fact they are all continuous which is easily checked from their definitions.

We denote the continuous Σ–algebra defined above by D_Σ.

Restriction, renaming and parallel composition are defined recursively. This corresponds to the fact that they may be "eliminated" for finite CHOCS processes modulo higher order bisimulation (see section 2.8).

It follows from proposition 4.2.10 and definition 4.3.1 that

Proposition 4.3.2 *The semantic function*

$$D[\![\ ]\!] : T_\Sigma \longrightarrow D_\Sigma$$

cuts down to surjections

$$T_\Sigma \twoheadrightarrow K(D) \text{ and } T_{\Sigma'} \twoheadrightarrow K(D)$$

Finite CHOCS thus provides a syntax for the finite elements of D. We shall later (in proposition 4.3.6 and 4.3.7) see how this statement can be strengthened to the elements of Lev_k of section 2.8 and LEV_k of section 4.2.

As for the operational semantics for finite CHOCS we may relate the denotational operators to the transition system view of D. In the following we shall abuse the notation from the operational description of CHOCS and we write: $d \xrightarrow{\Gamma} d''$ if $\Gamma = a?d'$ & $d \xrightarrow{a?d'} d''$ or $\Gamma = a!d'$ & $d \xrightarrow{a!d'} d''$ or $\Gamma = \tau$ & $d \xrightarrow{\tau} d''$ and we use the notation $\overline{\Gamma}$ for actions of the form $a?d'$ or $a!d'$ with the following meaning: $\overline{a?d'} = a!d'$ and $\overline{a!d'} = a?d'$.

Proposition 4.3.3 *For all $d_1, d_2 \in K(D)$:*

$$
\begin{array}{lll}
(i)(a) & nil^D \downarrow & (b)\ nil^D \not\rightarrow \\
(ii)(a) & \Omega^D \uparrow & (b)\ \Omega^D \not\rightarrow \\
(iii)(a) & a?^D(d_1,d_2) \downarrow & \\
\quad (b) & a?^D(d_1,d_2) \xrightarrow{\Gamma} d \iff & \Gamma = a?d_1 \ \&\ d = d_2 \\
(iv)(a) & a!^D(d_1,d_2) \downarrow & \\
\quad (b) & a!^D(d_1,d_2) \xrightarrow{\Gamma} d \iff & \Gamma = a!d_1 \ \&\ d = d_2 \\
(v)(a) & \tau^D(d_1,d_2) \downarrow & \\
\quad (b) & \tau^D(d_1) \xrightarrow{\Gamma} d \iff & \Gamma = \tau \ \&\ d = d_1 \\
(vi)(a) & (d_1 +^D d_2) \uparrow \iff & d_1 \uparrow \text{ or } d_2 \uparrow \\
\quad (b) & d_1 +^D d_2 \xrightarrow{\Gamma} d \iff & d_1 \xrightarrow{\Gamma} d \text{ or } d_2 \xrightarrow{\Gamma} d
\end{array}
$$

Parallel composition:

$$
\begin{array}{lll}
(vii)(a) & (d_1 \mid^D d_2) \uparrow \iff & d_1 \uparrow \text{ or } d_2 \uparrow \\
\quad (b) & (d_1 \mid^D d_2) \xrightarrow{\Gamma} d \iff & \Gamma = a?d' \ \& \quad \exists \Gamma_i', e_i', e_i''(i=1,2).d_1 \xrightarrow{\Gamma_i'} e_i'' \\
& & \quad \& \ \Gamma_i' = a?e_i' \ \& \ e_1' \sqsubseteq d' \sqsubseteq e_2' \\
& & \quad \& \ e_1'' \mid^D d_2 \sqsubseteq d'' \sqsubseteq e_2'' \mid^D d_2 \\
& & \text{or} \quad \exists \Gamma_i', e_i', e_i''(i=1,2).d_2 \xrightarrow{\Gamma_i'} e_i'' \\
& & \quad \& \ \Gamma_i' = a?e_i' \ \& \ e_1' \sqsubseteq d' \sqsubseteq e_2' \\
& & \quad \& \ d_1 \mid^D e_1'' \sqsubseteq d'' \sqsubseteq d_1 \mid^D e_2'' \\
& & \text{or} \quad \Gamma = a!d' \ \& \quad \exists \Gamma_i', e_i', e_i''(i=1,2).d_1 \xrightarrow{\Gamma_i'} e_i'' \\
& & \quad \& \ \Gamma_i' = a!e_i' \ \& \ e_1' \sqsubseteq d' \sqsubseteq e_2' \\
& & \quad \& \ e_1'' \mid^D d_2 \sqsubseteq d'' \sqsubseteq e_2'' \mid^D d_2 \\
& & \text{or} \quad \exists \Gamma_i', e_i', e_i''(i=1,2).d_2 \xrightarrow{\Gamma_i'} e_i'' \\
& & \quad \& \ \Gamma_i' = a!e_i' \ \& \ e_1' \sqsubseteq d' \sqsubseteq e_2' \\
& & \quad \& \ d_1 \mid^D e_1'' \sqsubseteq d'' \sqsubseteq d_1 \mid^D e_2'' \\
& & \text{or} \quad \Gamma = \tau \ \& \quad \exists e_i''(i=1,2).d_1 \xrightarrow{\tau} e_i'' \\
& & \quad \& \ e_1'' \mid^D d_2 \sqsubseteq d'' \sqsubseteq e_2'' \mid^D d_2 \\
& & \text{or} \quad \exists e_i''(i=1,2).d_2 \xrightarrow{\tau} e_i'' \\
& & \quad \& \ d_1 \mid^D e_1'' \sqsubseteq d'' \sqsubseteq d_1 \mid^D e_2'' \\
& & \text{or} \quad \exists \Gamma_i', e_i'', f_i''(i=1,2). \\
& & \quad d_1 \xrightarrow{\Gamma_i'} e_i'' \ \& \ d_2 \xrightarrow{\overline{\Gamma_i'}} f_i'' \\
& & \quad \& \ e_1'' \mid^D f_1'' \sqsubseteq d'' \sqsubseteq e_2'' \mid^D f_2''
\end{array}
$$

Restriction:

$$
\begin{array}{lllll}
(viii)(a) & (d_1\backslash a^D)\uparrow & \Longleftrightarrow & d_1\uparrow & \\
(b) & (d_1\backslash a^D)\xrightarrow{\Gamma} d'' & \Longleftrightarrow & \Gamma = b?d' \;\&\; & \exists \Gamma'_i, e'_i, e''_i (i=1,2).d_1 \xrightarrow{\Gamma'_i} e''_i \\
 & & & & \;\&\; \Gamma'_i = b?e'_i \;\&\; b \neq a \;\&\; e'_1 \sqsubseteq d' \sqsubseteq e'_2 \\
 & & & & \;\&\; e''_1\backslash a^D \sqsubseteq d'' \sqsubseteq e''_2\backslash a^D \\
 & & or & \Gamma = b!d' \;\&\; & \exists \Gamma'_i, e'_i, e''_i (i=1,2).d_1 \xrightarrow{\Gamma'_i} e''_i \\
 & & & & \;\&\; \Gamma'_i = b!e'_i \;\&\; b \neq a \;\&\; e'_1 \sqsubseteq d' \sqsubseteq e'_2 \\
 & & & & \;\&\; e''_1\backslash a^D \sqsubseteq d'' \sqsubseteq e''_2\backslash a^D \\
 & & or & \Gamma = \tau \;\&\; & \exists e''_i (i=1,2).d_1 \xrightarrow{\tau} e''_i \\
 & & & & \;\&\; e''_1\backslash a^D \sqsubseteq d'' \sqsubseteq e''_2\backslash a^D
\end{array}
$$

Restriction:

$$
\begin{array}{lllll}
(ix)(a) & (d_1[S]^D)\uparrow & \Longleftrightarrow & d_1\uparrow & \\
(b) & (d_1[S]^D)\xrightarrow{\Gamma} d'' & \Longleftrightarrow & \Gamma = b?d' \;\&\; & \exists \Gamma'_i, e'_i, e''_i (i=1,2).d_1 \xrightarrow{\Gamma'_i} e''_i \\
 & & & & \;\&\; \Gamma'_i = a?e'_i \;\&\; b = S(a) \\
 & & & & \;\&\; e'_1 \sqsubseteq d' \sqsubseteq e'_2 \\
 & & & & \;\&\; e''_1[S]^D \sqsubseteq d'' \sqsubseteq e''_2[S]^D \\
 & & or & \Gamma = b!d' \;\&\; & \exists \Gamma'_i, e'_i, e''_i (i=1,2).d_1 \xrightarrow{\Gamma'_i} e''_i \\
 & & & & \;\&\; \Gamma'_i = a!e'_i \;\&\; b = S(a) \\
 & & & & \;\&\; e'_1 \sqsubseteq d' \sqsubseteq e'_2 \\
 & & & & \;\&\; e''_1[S]^D \sqsubseteq d'' \sqsubseteq e''_2[S]^D \\
 & & or & \Gamma = \tau \;\&\; & \exists e''_i (i=1,2).d_1 \xrightarrow{\tau} e''_i \\
 & & & & \;\&\; e''_1[S]^D \sqsubseteq d'' \sqsubseteq e''_2[S]^D
\end{array}
$$

PROOF: $(i)-(v)$ are immediate from definition 4.3.1.
(vi) is derived from $d_1 +^D d_2 = d_1 \uplus d_2 = Con(d_1 \cup d_2)$.
For (vii) we define

$$
\begin{aligned}
\Theta \equiv\ & \{\{\langle a?,(d'_1, d''_1 \mid^D d_2)\rangle\} : \langle a?,(d'_1,d''_1)\rangle \in d_1\} \\
& \cup\{\{\langle a!,(d'_1, d''_1 \mid^D d_2)\rangle\} : \langle a!,(d'_1,d''_1)\rangle \in d_1\} \\
& \cup\{\{\langle \tau,(d''_1 \mid^D d_2)\rangle\} : \langle \tau, d''_1\rangle \in d_1\} \\
& \cup\{\{\langle a?,(d'_2, d_1 \mid^D d''_2)\rangle\} : \langle a?,(d'_2,d''_2)\rangle \in d_2\} \\
& \cup\{\{\langle a!,(d'_2, d_1 \mid^D d''_2)\rangle\} : \langle a!,(d'_2,d''_2)\rangle \in d_2\} \\
& \cup\{\{\langle \tau,(d_1 \mid^D d''_2)\rangle\} : \langle \tau, d''_2\rangle \in d_2\} \\
& \cup\{\{\langle \tau,(d''_1 \mid^D d''_2)\rangle\} : \langle a?,(d'_1,d''_1)\rangle \in d_1 \;\&\; \langle a!,(d'_1,d''_2)\rangle \in d_2\} \\
& \cup\{\{\langle \tau,(d''_1 \mid^D d''_2)\rangle\} : \langle a!,(d'_1,d''_1)\rangle \in d_1 \;\&\; \langle a?,(d'_1,d''_2)\rangle \in d_2\} \\
& \cup\{\{\bot\} : \bot\in d_1 \text{ or } \bot\in d_2\}
\end{aligned}
$$

and

$$\begin{aligned}
\Phi \equiv\ & \{\langle a?,(d_1', d_1'' \mid^D d_2)\rangle \ :\ \langle a?,(d_1', d_1'')\rangle \in d_1\} \\
& \cup\{\langle a!,(d_1', d_1'' \mid^D d_2)\rangle \ :\ \langle a!,(d_1', d_1'')\rangle \in d_1\} \\
& \cup\{\langle \tau,(d_1'' \mid^D d_2)\rangle\} \ :\ \langle \tau, d_1''\rangle \in d_1\} \\
& \cup\{\langle a?,(d_2', d_1 \mid^D d_2'')\rangle \ :\ \langle a?,(d_2', d_2'')\rangle \in d_2\} \\
& \cup\{\langle a!,(d_2', d_1 \mid^D d_2'')\rangle \ :\ \langle a!,(d_2', d_2'')\rangle \in d_2\} \\
& \cup\{\langle \tau,(d_1 \mid^D d_2'')\rangle \ :\ \langle \tau, d_2''\rangle \in d_2\} \\
& \cup\{\langle \tau,(d_1'' \mid^D d_2'')\rangle \ :\ \langle a?,(d_1', d_1'')\rangle \in d_1 \ \&\ \langle a!,(d_1', d_2'')\rangle \in d_2\} \\
& \cup\{\langle \tau,(d_1'' \mid^D d_2'')\rangle \ :\ \langle a!,(d_1', d_1'')\rangle \in d_1 \ \&\ \langle a?,(d_1', d_2'')\rangle \in d_2\} \\
& \cup\{\perp : \perp\in d_1 \text{ or } \perp\in d_2\}
\end{aligned}$$

Now

$$\begin{aligned}
(d_1 \mid^D d_2) &= Con(\bigcup \Theta^\star) \\
&= Con((\bigcup \Theta)^\star) \text{ by [Plo76] p. 477} \\
&= Con(\bigcup \Theta) \text{ since } d \in K(D) \\
&= Con(\Phi)
\end{aligned}$$

and (vii) is derived from this description.

$(viii)$ and (ix) are derived similarly. □

Proposition 4.3.4 *For all* $p \in T_\Sigma . p \sim^B D[\![p]\!]$.

PROOF: Let us define a height function on T_Σ in the following way:

$$ht(\sigma(p_1,\ldots,p_n)) = sup\{ht(p_i) \ :\ 1 \le i \le n\} + 1$$

Note that both p_1 and p_2 contribute to the height of p in $a?^F p_1.p_2$ and $a!p_1.p_2$. As an easy consequence of proposition 4.3.3, we have:

$$\begin{aligned}
p \xrightarrow{a?p'} p'' &\implies ht(p') < ht(p) \ \&\ ht(p'') < ht(p) \\
p \xrightarrow{a!p'} p'' &\implies ht(p') < ht(p) \ \&\ ht(p'') < ht(p) \\
p \xrightarrow{\tau} p'' &\implies ht(p'') < ht(p)
\end{aligned}$$

The proposition is proved by induction on $ht(p)$, and the structure of p. The cases arising from the operators in Σ' are obvious from the close match of moves as can be seen from proposition 2.8.4 and proposition 4.3.3. We give the case where:

$\underline{p \equiv p_1 \mid p_2}$

Firstly,

$$\begin{array}{rcll} p\uparrow & \iff & p_1\uparrow \text{ or } p_2\uparrow & \text{by proposition 2.8.4} \\ & \iff & D[\![p_1]\!]\uparrow \text{ or } D[\![p_2]\!]\uparrow & \text{by induction hypothesis} \\ & \iff & (D[\![p_1]\!] \mid^D D[\![p_2]\!])\uparrow & \text{by proposition 4.3.3} \\ & \iff & D[\![p_1 \mid^D p_2]\!]\uparrow & \end{array}$$

Next,

$$\begin{array}{ll} & p \xrightarrow{a?p'} p'' \\ \Longrightarrow & p_1 \xrightarrow{a?p'} p_1'' \;\&\; p'' \equiv p_1'' \mid p_2 \\ \text{or} & p_2 \xrightarrow{a?p'} p_2'' \;\&\; p'' \equiv p_1 \mid p_2'' \\ & \text{by proposition 2.8.4} \\ \Longrightarrow & \exists d_1'.d_1''.D[\![p_1]\!] \xrightarrow{a?d_1'} d_1'' \;\&\; p' \sqsubseteq^B d_1' \;\&\; p_1'' \sqsupseteq^B d_1'' \\ & \text{by induction hypothesis on } p_1 \\ \text{or} & \exists d_2'.d_2''.D[\![p_2]\!] \xrightarrow{a?d_2'} d_2'' \;\&\; p' \sqsubseteq^B d_2' \;\&\; p_2'' \sqsupseteq^B d_2'' \\ & \text{by induction hypothesis on } p_2 \\ \Longrightarrow & p_1'' \mid p_2 \sim^B D[\![p_1'' \mid p_2]\!] \\ & \text{by induction hypothesis on } p_1'' \mid p_2 \\ & = D[\![p_1'']\!] \mid^D D[\![p_2]\!] \\ & \sqsupseteq^B d_1'' \mid^D D[\![p_2]\!] \\ & \text{by proposition 4.2.10 and monotonicity of } \mid^D \\ \text{or} & p_1 \mid p_2'' \sim^B D[\![p_1 \mid p_2'']\!] \\ & \text{by induction hypothesis on } p_1 \mid p_2'' \\ & = D[\![p_1]\!] \mid^D D[\![p_2'']\!] \\ & \sqsupseteq^B D[\![p_1]\!] \mid^D d_2'' \\ & \text{by proposition 4.2.10 and monotonicity of } \mid^D \\ \Longrightarrow & \exists d', d''.D[\![p]\!] \xrightarrow{a?d'} d'' \;\&\; p' \sqsubseteq^B d' \;\&\; p'' \sqsupseteq^B d'' \end{array}$$

Similarly, we can show

$$p \xrightarrow{a?p'} p'' \Longrightarrow \exists d', d''.D[\![p]\!] \xrightarrow{a?d'} d'' \;\&\; d' \sqsubseteq^B p' \;\&\; d'' \sqsupseteq^B p''$$

If $p \xrightarrow{a!p'} p''$ we may argue as above with ? replaced with !.

$p \xrightarrow{\tau} p''$

$\Longrightarrow$ $p_1 \xrightarrow{\tau} p_1'' \;\&\; p'' \equiv p_1'' \mid p_2$

or $p_2 \xrightarrow{\tau} p_2'' \;\&\; p'' \equiv p_1 \mid p_2''$

or $\exists \mathrm{r}'.p_1 \xrightarrow{\Gamma'} p_1'' \;\&\; p_2 \xrightarrow{\overline{\Gamma'}} p_2'' \;\&\; p'' \equiv p_1'' \mid p_2''$

by proposition 2.8.4

$\Longrightarrow$ $\exists d_1''.D[\![p_1]\!] \xrightarrow{\tau} d_1'' \;\&\; p_1'' \sqsubseteq^B d_1''$

by induction hypothesis on p_1

or $\exists d_2''.D[\![p_2]\!] \xrightarrow{\tau} d_2'' \;\&\; p_2'' \sqsubseteq^B d_2''$

by induction hypothesis on p_2

or $\mathrm{r}' = a?p' \;\&\; \exists .d_1', d_1''.D[\![p_1]\!] \xrightarrow{a?d_1'} d_1'' \;\&\; p' \sqsubseteq^B d_1' \;\&\; p_1'' \sqsubseteq^B d_1''$

by induction hypothesis on p_1

$\&\; D[\![p_2]\!] \xrightarrow{a!d_2'} d_2'' \;\&\; p' \sqsubseteq^B d_2' \;\&\; p_2'' \sqsubseteq^B d_2''$

by induction hypothesis on p_2

$p' \sim^B D[\![p']\!] \sqsubseteq d_i', (i = 1, 2)$

by induction hypothesis on p'

$D[\![p_2]\!] \xrightarrow{a?D[\![p']\!]} d_1'' \;\&\; D[\![p_2]\!] \xrightarrow{a!D[\![p']\!]} d_2''$

since $D[\![p]\!]$ is convex closed

or $\mathrm{r} = a!p'$ and we may argue as above with ? substituted for !

$\Longrightarrow$ $p_1'' \mid p_2 \sim^B D[\![p_1'' \mid p_2]\!]$

by induction hypothesis on $p_1'' \mid p_2$

$= D[\![p_1'']\!] \mid^D D[\![p_2]\!]$

$\sqsubseteq^B d_1'' \mid^D D[\![p_2]\!]$

by proposition 4.2.10 and monotonicity of $\mid^D$

or $p_1 \mid p_2'' \sim^B D[\![p_1 \mid p_2'']\!]$

by induction hypothesis on $p_1 \mid p_2''$

$= D[\![p_1]\!] \mid^D D[\![p_2'']\!]$

$\sqsubseteq^B D[\![p_1]\!] \mid^D d_2''$

by proposition 4.2.10 and monotonicity of $\mid^D$

or $p_1'' \mid p_2'' \sim^B D[\![p_1'' \mid p_2'']\!]$

by induction hypothesis on $p_1'' \mid p_2''$

$= D[\![p_1'']\!] \mid^D D[\![p_2'']\!]$

$\sqsubseteq^B d_1'' \mid^D d_2''$

by proposition 4.2.10 and monotonicity of $\mid^D$

$\Longrightarrow$ $\exists d''.D[\![p]\!] \xrightarrow{\tau} d'' \;\&\; p'' \sqsubseteq^B d''$

Similarly, we can show

$$p \xrightarrow{\tau} p'' \Longrightarrow \exists d''.D[\![p]\!] \xrightarrow{\tau} d'' \;\&\; d'' \sqsubseteq^B p''$$

Also,

$$D[\![p]\!] \xrightarrow{a?d'} d''$$

$\Longrightarrow$ $\exists d'_i, d''_i (i = 1,2).D[\![p_1]\!] \xrightarrow{a?d'_i} d''_i$
$\& \; d'_1 \sqsubseteq d' \sqsubseteq d'_2 \;\&\; d''_1 \mid^D D[\![p_2]\!] \sqsubseteq d'' \sqsubseteq d''_2 \mid^D D[\![p_2]\!]$
by proposition 4.3.3

or $\exists d'_i, d''_i (i = 1,2).D[\![p_2]\!] \xrightarrow{a?d'_i} d''_i$
$\& \; d'_1 \sqsubseteq d' \sqsubseteq d'_2 \;\&\; D[\![p_1]\!] \mid^D d''_1 \sqsubseteq d'' \sqsubseteq D[\![p_1]\!] \mid^D d''_2$
by proposition 4.3.3

$\Longrightarrow$ $\exists p'_i.p''_i (i = 1,2).p_1 \xrightarrow{a?p'_i} p''_i \;\&\; p'_1 \lesssim^B d'_1, d'_2 \lesssim^B p'_2 \;\&\; p''_1 \lesssim^B d''_1, d''_2 \lesssim^B p''_2$
by induction hypothesis on p_1

or $\exists p'_i.p''_i (i = 1,2).p_2 \xrightarrow{a?p'_i} p''_i \;\&\; p'_1 \lesssim^B d'_1, d'_2 \lesssim^B p'_2 \;\&\; p''_1 \lesssim^B d''_1, d''_2 \lesssim^B p''_2$
by induction hypothesis on p_2

$\Longrightarrow$ $p \xrightarrow{a?p'_1} p''_1 \mid p_2 \;\&\; p''_1 \mid p_2 \sim^B D[\![p''_1 \mid p_2]\!] = D[\![p''_1]\!] \mid^D D[\![p_2]\!] \lesssim^B d$
by induction hypothesis on $p''_1 \mid p_2$

or $p \xrightarrow{a?p'_1} p_1 \mid p''_1 \;\&\; p_1 \mid p''_1 \sim^B D[\![p_1 \mid p''_1]\!] = D[\![p_1]\!] \mid^D D[\![p''_1]\!] \lesssim^B d$
by induction hypothesis on $p_1 \mid p''_1$

and similarly $d \lesssim^B p''_2 \mid p_2$ or $d \lesssim^B p_1 \mid p''_2$.

If $D[\![p]\!] \xrightarrow{a!d'} d''$ we may argue as above with ? substituted for !.

$$D[\![p]\!] \xrightarrow{\tau} d''$$

$\Longrightarrow$ $\exists d''_i (i = 1,2).D[\![p_1]\!] \xrightarrow{\tau} d''_i \;\&\; d''_1 \mid^D D[\![p_2]\!] \sqsubseteq d'' \sqsubseteq d''_2 \mid^D D[\_p_2]\!]$

or $\exists d''_i (i = 1,2).D[\![p_2]\!] \xrightarrow{\tau} d''_i \;\&\; D[\![p_1]\!] \mid^D d''_1 \sqsubseteq d'' \sqsubseteq D[\![p_1]\!] \mid^D d''_2$

or $\exists d'_i, d''_i, e''_i (i = 1,2).D[\![p_1]\!] \xrightarrow{a?d'_i} d''_i \;\&\; D[\![p_2]\!] \xrightarrow{a!d'_i} e''_i$
$\& \; d''_1 \mid^D e''_1 \sqsubseteq d'' \sqsubseteq d''_2 \mid^D e''_2$

or $\exists d'_i, d''_i, e''_i (i = 1,2).D[\![p_1]\!] \xrightarrow{a!d'_i} d''_i \;\&\; D[\![p_2]\!] \xrightarrow{a?d'_i} e''_i$
$\& \; d''_1 \mid^D e''_1 \sqsubseteq d'' \sqsubseteq d''_2 \mid^D e''_2$
by proposition 4.3.3

$\Longrightarrow$ $\exists p''_i (i = 1,2).p_1 \xrightarrow{\tau} p''_i \;\&\; p''_1 \lesssim^B d''_1, d''_2 \lesssim^B p''_2$
by induction hypothesis on p_1

or $\exists p''_i (i = 1,2).p_2 \xrightarrow{\tau} p''_i \;\&\; p''_1 \lesssim^B d''_1, d''_2 \lesssim^B p''_2$
by induction hypothesis on p_2

or $\exists p'_i, p''_i, q''_i (i = 1,2).p_1 \xrightarrow{a?p'_i} p''_i \;\&\; p_2 \xrightarrow{a?p'_i} q''_i \;\&$
$p'_1 \lesssim^B d'_1, d'_2 \lesssim^B p'_2 \;\&\; p''_1 \lesssim^B d''_1, d''_2 \lesssim^B p''_2 \;\&\; q''_1 \lesssim^B e''_1, q''_2 \lesssim^B e''_2$
by induction hypothesis on p_1 and p_2

$\Longrightarrow$ $p \xrightarrow{\tau} p''_1 \mid p_2 \;\&\; p''_1 \mid p_2 \sim^B D[\![p''_1 \mid p_2]\!] = D[\![p''_1]\!] \mid^D D[\![p_2]\!] \lesssim^B d$
by induction hypothesis on $p''_1 \mid p_2$

or $p \xrightarrow{\tau} p_1 \mid p''_1 \;\&\; p_1 \mid p''_1 \sim^B D[\![p_1 \mid p''_1]\!] = D[\![p_1]\!] \mid^D D[\![p''_1]\!] \lesssim^B d$
by induction hypothesis on $p_1 \mid p''_1$

or $p \xrightarrow{\tau} p''_1 \mid q''_1 \;\&\; p_1 \mid q''_1 \sim^B D[\![p''_1 \mid q''_1]\!] = D[\![p''_1]\!] \mid^D D[\![q''_1]\!] \lesssim^B d$
by induction hypothesis on $p''_1 \mid q''_1$

and similarly $d \sqsubseteq^B p_2'' \mid p_2$ or $d \sqsubseteq^B p_1 \mid p_2''$ or $d \sqsubseteq^B p_2'' \mid q_2''$.

The cases where $p \equiv p_1\backslash a$ or $p \equiv p_1[S]$ can be derived similarly.
Altogether we have $p \sim^B D[\![p]\!]$. □

Theorem 4.3.5 *(Full Abstraction for finite processes) For all* $p_1, p_2 \in T_\Sigma$*:*

$$p_1 \sqsubseteq^B p_2 \iff D[\![p_1]\!] \sqsubseteq D[\![p_2]\!]$$

PROOF: Follows from proposition 4.2.10 and proposition 4.3.4. □

Proposition 4.3.6 *Assume Names is finite, then*

$$\forall k. \forall p \in Lev_k. \exists d \in LEV_k. D[\![p]\!] = d$$

PROOF: By induction on k:

$\underline{k = 0}$

$$Lev_0 = \{\Omega\};\ D[\![\Omega]\!] = \{|\perp|\} \in LEV_0$$

$\underline{k+1}$ If $p \in Lev_{k+1}$ then

either $p \in Lev_k$ and by induction there is a $d \in LEV_k \subseteq LEV_{k+1}$ such that $D[\![p]\!] = d$ and we are through

or p has the form $\Sigma_{i \in I} p_i$ where I is a finite index set and p_i has one of the following forms: $a?^F p_i'.p_i''$, $a!p_i'.p_i''$, $\tau.p_i'$ or Ω, where $p_i', p_i'' \in Lev_k$ and $a \in Names$. By induction there is $d_i', d_i'' \in LEV_k$ such that $D[\![p_i']\!] = d_i'$ and $D[\![p_i'']\!] = d_i''$. Then $D[\![a?^F p_i'.p_i'']\!] = a?^D(D[\![p_i']\!], D[\![p_i'']\!]) = a?^D(d_i', d_i'') \in LEV_{k+1}$ and $D[\![a!p_i'.p_i'']\!] = a!^D(D[\![p_i']\!], D[\![p_i'']\!]) = a!^D(d_i', d_i'') \in LEV_{k+1}$ and $D[\![\tau.p_i']\!] = \tau^D(D[\![p_i']\!]) = \tau^D(d_i') \in LEV_{k+1}$ and $D[\![\Omega]\!] = \{|\perp|\} \in LEV_0 \subseteq LEV_{k+1}$. Since $\Sigma_{i \in I} p_i$ is shorthand for $p_1 + \ldots + p_n$ where $\{1, \ldots, n\}$ is an enumeration of I we have $D[\![\Sigma_{i \in I} p_i]\!] = D[\![p_1]\!] \uplus \ldots \uplus D[\![p_n]\!] \in LEV_{k+1}$. This proves the proposition in this case.

□

Proposition 4.3.7 *Assume Names is finite, then*

$$\forall k.\forall d \in LEV_k.\exists p \in Lev_k.D[\![p]\!] = d$$

PROOF: By induction on k:

$\underline{k = 0}$

$$Lev_0 = \{\Omega\};\ D[\![\Omega]\!] = \{|\perp|\} \in LEV_0$$

$\underline{k+1}$ Then

either $d \in LEV_k$ and by induction there exists a $p \in Lev_k \subseteq Lev_{k+1}$ such that $D[\![p]\!] = d$ and we are through.

or

$$\begin{aligned} d \;=\; & (\{\langle a?,(d',d'')\rangle \;:\; a \in Names, d', d'' \in LEV_k\} \\ & \cup\{\langle a!,(d',d'')\rangle \;:\; a \in Names, d', d'' \in LEV_k\} \\ & \cup\{\langle \tau, d'\rangle \;:\; a \in Names, d' \in LEV_k\} \\ & \cup\{d \;:\; d \in LEV_k\})^\star \end{aligned}$$

By induction there exist $p', p'' \in Lev_k$ such that $D[\![p']\!] = d'$ and $D[\![p'']\!] = d''$. If *Names* is finite we may write d as the union of singleton sets $\{|\ \langle a?,(d',d'')\rangle\ |\}$, $\{|\ \langle a!,(d',d'')\rangle\ |\}$, or $\{|\ \langle \tau, d''\rangle\ |\}$ and for each such set define p_i as $a?^F p'.p''$, $a!p'.p''$ or $\tau.p''$ respectively. Clearly $D[\![p_i]\!] = d_i$ and $p_i \in Lev_{k+1}$. Then define p as Σp_i. Clearly $D[\![p]\!] = d$ and $p \in Lev_{k+1}$ and we are through.

□

4.4 A Denotational Semantics for CHOCS

The semantics given in the previous section only applies to finite processes. We now use these results to extend the denotational semantics to the CHOCS language.

The semantic function has to take free variables (to be bound by input prefix) into account and therefore takes an environment $\rho : V \to D$ as an argument. We use the standard notation $\rho[d/x]$ for updating an environment. The environment $\rho[d/x]$ is the same as ρ except on x where it returns d.

Definition 4.4.1 $D[\![\ ]\!] : CHOCS \to D^V \to D$

$$
\begin{aligned}
D[\![nil]\!]\rho &= \emptyset \\
D[\![\Omega]\!]\rho &= \{|\perp|\} \\
D[\![a?^F p_1.p_2]\!]\rho &= a?^D(D[\![p_1]\!]\rho, D[\![p_2]\!]\rho) \\
D[\![a?x.p_1]\!]\rho &= FD(\lambda d.D[\![p_1]\!]\rho[d/x]) \\
&= (\bigcup\{a?^D(d, D[\![p_1]\!]\rho[d/x]) \ : \ d \in D\})^\star \\
D[\![a!p_1.p_2]\!]\rho &= a!^D(D[\![p_1]\!]\rho, D[\![p_2]\!]\rho) \\
D[\![\tau.p_1]\!]\rho &= \tau^D(D[\![p_1]\!]\rho) \\
D[\![p_1 + p_2]\!]\rho &= D[\![p_1]\!]\rho +^D D[\![p_2]\!]\rho \\
D[\![p_1 \mid p_2]\!]\rho &= (D[\![p_1]\!]\rho) \mid^D (D[\![p_2]\!]\rho) \\
D[\![p_1 \backslash a]\!]\rho &= (D[\![p_1]\!]\rho)\backslash a^D \\
D[\![p_1[S]]\!]\rho &= (D[\![p_1]\!]\rho)[S]^D \\
D[\![x]\!]\rho &= \rho(x)
\end{aligned}
$$

where $F = \lambda D \in P^0[D].\lambda f \in [D \to D].\biguplus((P^0(\lambda d.a?^D(d, fd)))D)$

Note that the semantics of input prefix is given as the Big Union of all possible sets of triples $\langle a?, (d, [\![p_1]\!]\rho[d/x])\rangle$ where $d \in D$, reflecting that any value d could be received. Alternatively we could say that input prefix has a choice of any value $d \in D$ which is similar to the intuition in the operational semantics of CHOCS.

We need to check that the above definition is sound i.e. that $D[\![\ ]\!]$ is continuous in its environment argument. The proof of this is similar to the proof of continuity of the denotational semantics for the λ–Calculus as presented in [Bar84].

Proposition 4.4.2 $\lambda d.D[\![p]\!]\rho[d/x]$ *is continuous.*

PROOF: We proceed by structural induction on p. The only nontrivial cases are:

$\underline{p \equiv a?y.p_1}$ Define $G = FD$. Then:

$$
\begin{aligned}
D[\![p]\!]\rho[d/x] &= G(\lambda e.D[\![p_1]\!]\rho[d/x][e/y]) \\
&= G(\lambda e.f(d, e))
\end{aligned}
$$

for some f. Clearly f is continuous in each argument separately by the induction hypothesis and thus continuous. Let $g(d) = G(\lambda e.f(d, e))$ then $g = G \circ \hat{f}$ and $\hat{f} = curry f$. Then clearly g is continuous since G, f, $curry$ and $\circ$ are continuous.

$\underline{p \equiv y}$

$$g(d) = D[\![y]\!]\rho[d/x] = \begin{cases} d & \text{if } x = y \\ \rho(y) & \text{otherwise} \end{cases}$$

Clearly $g(d)$ is continuous.

All other cases follow straightforwardly from structural induction. We give one case for illustration:

$\underline{p \equiv p_1 \mid p_2}$

$$g(d) = D[\![p_1 \mid p_2]\!]\rho[d/x] = D[\![p_1]\!]\rho[d/x] \mid^D D[\![p_2]\!]\rho[d/x] = g_1(d) \mid^D g_2(d)$$

By induction g_1, g_2 are continuous and $\mid^D$ is continuous in both arguments. Thus g is continuous.

□

Before proceeding to extend the full abstraction result for finite CHOCS to the CHOCS language we present the following useful relation between the syntactic substitution as defined in definition 2.2.3 and updating of environments:

Proposition 4.4.3

$$D[\![p[q/x]]\!]\rho = D[\![p]\!]\rho[D[\![q]\!]\rho/x]$$

PROOF: We proceed by structural induction on p.

$\underline{p \equiv nil}$

$$D[\![nil[q/x]]\!]\rho \equiv D[\![nil]\!]\rho = nil^D = D[\![nil]\!]\rho[D[\![q]\!]\rho/x]$$

$\underline{p \equiv \Omega}$

$$D[\![\Omega[q/x]]\!]\rho \equiv D[\![\Omega]\!]\rho = \Omega^D = D[\![\Omega]\!]\rho[D[\![q]\!]\rho/x]$$

$\underline{p \equiv a?y.p_1}$ Assume $y \neq x$ and $y \notin fv(q)$ (otherwise use α-conversion on y):

$$\begin{aligned} D[\![(a?y.p_1)[q/x]]\!]\rho &\equiv D[\![a?y.(p_1[q/x])]\!]\rho \\ &\quad \text{by the definition of } D[\![\ ]\!] \\ &= FD(\lambda d.D[\![p_1[q/x]]\!]\rho[d/y]) \\ &\quad \text{by the induction hypothesis} \\ &= FD(\lambda d.D[\![p_1]\!]\rho[d/y][D[\![q]\!]\rho[d/y]/x]) \\ &\quad \text{since } y \neq x \text{ and } y \notin fv(q) \\ &= FD(\lambda d.D[\![p_1]\!]\rho[D[\![q]\!]\rho/x][d/y]) \\ &\quad \text{by the definition of } D[\![\ ]\!] \\ &= D[\![a?y.p_1]\!]\rho[D[\![q]\!]/x] \end{aligned}$$

$\underline{p \equiv y}$ if $y = x$ then

$$D[\![y[q/x]]\!]\rho \equiv D[\![q]\!]\rho = D[\![y]\!]\rho[D[\![q]\!]\rho/x]$$

$\underline{p \equiv y}$ if $y \neq x$ then

$$D[\![y[q/x]]\!]\rho \equiv D[\![y]\!]\rho = \rho(y) = \rho[D[\![q]\!]\rho/x](y) = D[\![y]\!]\rho[D[\![q]\!]\rho/x]$$

All other cases follow straightforwardly from structural induction. We give one case for illustration:

$\underline{p \equiv p_1 \mid p_2}$

$$\begin{aligned}
D[\![(p_1 \mid p_2)[q/x]]\!]\rho &\equiv D[\![(p_1[q/x]) \mid (p_2[q/x])]\!]\rho \\
&\quad \text{by the definition of } D[\![\ ]\!] \\
&= D[\![(p_1[q/x])]\!]\rho \mid^D D[\![(p_2[q/x])]\!]\rho \\
&\quad \text{by the induction hypothesis} \\
&= D[\![(p_1)]\!]\rho[D[\![q]\!]\rho/x] \mid^D D[\![(p_2)]\!]\rho[D[\![q]\!]\rho/x] \\
&\quad \text{by the definition of } D[\![\ ]\!] \\
&= D[\![p_1 \mid p_2]\!]\rho[D[\![q]\!]\rho/x]
\end{aligned}$$

□

In section 2.8 we defined a set of operational approximations p^n to p. We now define a set of denotational approximations $a_n(p)\rho$. These are given relative to an environment ρ.

Definition 4.4.4 *For every $p \in Pr \cup FPr$ and every n we define $a_n(p)\rho \in D$*

$$a_0(p)\rho = \{|\perp|\} \text{ for any } p$$

$$\begin{aligned}
a_{n+1}(nil)\rho &= \emptyset \\
a_{n+1}(\Omega)\rho &= \{|\perp|\} \\
a_{n+1}(a?^F p_1.p_2)\rho &= a?^D(a_n(p_1)\rho, a_n(p_2)\rho) \\
a_{n+1}(a?x.p_1)\rho &= FLEV_n(\lambda d.a_n(p_1)\rho[d/x]) \\
&= (\bigcup\{a?^D(d, a_n(p_1)\rho[d/x]) : d \in LEV_n\})^\star \\
a_{n+1}(a!p_1.p_2)\rho &= a!^D(a_n(p_1)\rho, a_n(p_2)\rho) \\
a_{n+1}(\tau.p_1)\rho &= \tau^D.(a_n(p_1)\rho) \\
a_{n+1}(p_1 + p_2)\rho &= (a_n(p_1)\rho) +^D (a_n(p_2)\rho)
\end{aligned}$$

$$\begin{aligned} a_{n+1}(p_1 \mid p_2)\rho &= (a_n(p_1)\rho) \mid^D (a_n(p_2)\rho) \\ a_{n+1}(p_1\backslash a)\rho &= (a_n(p_1)\rho)\backslash a^D \\ a_{n+1}(p_1[S])\rho &= (a_n(p_1)\rho)[S]^D \\ a_{n+1}(x)\rho &= \rho(x) \end{aligned}$$

Note that if $\rho(x) \in K(D)$ for every $x \in FV(p)$ then $a_n(p)\rho \in K(D)$.
The above definition is sound by arguments similar to those given for $D[\![\ ]\!]$.

The following proposition establishes the relationship between the operational approximation p^n and the denotational approximation $a_n(p)$ of p.

Proposition 4.4.5 *For all n and p and any environment ρ: $D[\![p^n]\!]\rho = a_n(p)\rho$.*

PROOF: We proceed by induction on n.

$\underline{n = 0}$ Trivial since for all p and ρ: $D[\![p^0]\!]\rho = \{\!|\perp|\!\} = a_0(p)\rho$.

$\underline{n+1}$ For the induction step we use a subinduction on the structure of p:

$\underline{p \equiv nil}$

$$D[\![nil^{n+1}]\!]\rho = D[\![nil]\!]\rho = \emptyset = a_{n+1}(nil)\rho$$

$\underline{p \equiv \Omega}$

$$D[\![\Omega^{n+1}]\!]\rho = D[\![\Omega]\!]\rho = \Omega^D = a_{n+1}(\Omega)\rho$$

$\underline{p \equiv a?x.p_1}$

$$\begin{aligned} D[\![(a?x.p_1)^{n+1}]\!]\rho &= D[\![\Sigma_{p \in Lev_n} a?^F p.p_1^n[p/x]]\!]\rho \\ &\quad \text{by definition of } p^n \text{ and definition of } D[\![\ ]\!] \\ &= (\bigcup\{a?^D([\![p]\!]\rho, [\![p_1^n[p/x]]\!]\rho) \ : \ p \in Lev_n\})^\star \\ &\quad \text{by proposition 4.4.2} \\ &= (\bigcup\{a?^D([\![p]\!]\rho, [\![p_1^n]\!]\rho[[\![p]\!]\rho/x]) \ : \ p \in Lev_n\})^\star \\ &\quad \text{by the induction hypothesis} \\ &= (\bigcup\{a?^D([\![p]\!]\rho, a_n(p_1)\rho[[\![p]\!]\rho/x]) \ : \ p \in Lev_n\})^\star \\ &\quad \text{by proposition 4.3.6 and proposition 4.3.6} \\ &= (\bigcup\{a?^D(d, a_n(p_1)\rho[d/x]) \ : \ d \in LEV_n\})^\star \\ &\quad \text{by the definition of } a_n(p)\rho \\ &= a_{n+1}(a?x.p_1)\rho \end{aligned}$$

$\underline{p \equiv x}$

$$D[\![x^{n+1}]\!]\rho = D[\![x]\!]\rho = \rho(x) = a_{n+1}(x)\rho$$

All other cases follow straightforwardly from structural induction. We give one case for illustration:

$\underline{p \equiv p_1 \mid p_2}$

$$\begin{aligned} D[\![(p_1 \mid p_2)^{n+1}]\!]\rho &= D[\![p_1^n \mid p_2^n]\!]\rho \\ &\quad \text{by definition of } p^n \text{ and definition of } D[\![\ ]\!] \\ &= (D[\![p_1^n]\!]\rho) \mid^D (D[\![p_2^n]\!]\rho) \\ &\quad \text{by the induction hypothesis} \\ &= (a_n(p_1)\rho) \mid^D (a_n(p_2)\rho) \\ &\quad \text{by the definition of } a_n(p)\rho \\ &= a_{n+1}(p_1 \mid p_2)\rho \end{aligned}$$

□

Proposition 4.4.6 *For all p and ρ: $D[\![p]\!]\rho = \bigsqcup_n a_n(p)\rho$.*

PROOF: By structural induction on p:

$\underline{p \equiv nil}$

$$D[\![nil]\!]\rho = \emptyset = \bigsqcup_n a_n(nil)\rho$$

$\underline{p \equiv \Omega}$

$$D[\![\Omega]\!]\rho = \{\!|\perp|\!\} = \bigsqcup_n a_n(\Omega)\rho$$

$\underline{p \equiv a?x.p_1}$

$$\begin{aligned} D[\![a?x.p_1]\!]\rho &= FD(\lambda d.D[\![p_1]\!]\rho[d/x]) \\ &\quad \text{by the definition of } D[\![\ ]\!] \\ &= FD(\lambda d.\bigsqcup_n a_n(p_1)\rho[d/x]) \\ &\quad \text{by the induction hypothesis} \\ &= \bigsqcup_n FD(\lambda d.a_n(p_1)\rho[d/x]) \\ &\quad \text{by continuity of } F \end{aligned}$$

$$
\begin{aligned}
&= \bigsqcup_n F(\bigsqcup_m LEV_m)(\lambda d.a_n(p_1)\rho[d/x]) \\
&\quad \text{by corollary 4.2.6} \\
&= \bigsqcup_n \bigsqcup_m FLEV_m(\lambda d.a_n(p_1)\rho[d/x]) \\
&\quad \text{by continuity of } F \\
&= \bigsqcup_n FLEV_n(\lambda d.a_n(p_1)\rho[d/x]) \\
&\quad \text{by [Plo81b]} \\
&= \bigsqcup_n a_{n+1}(a?x.p_1)\rho \\
&\quad \text{by the definition of } a_n(p)\rho \\
&= \bigsqcup_n a_n(a?x.p_1)\rho
\end{aligned}
$$

$\underline{p \equiv x}$

$$D[\![x]\!]\rho = \rho(x) = \bigsqcup_n a_n(x)\rho$$

All other cases follow straightforwardly from structural induction. We give one case for illustration:

$\underline{p \equiv p_1 \mid p_2}$

$$
\begin{aligned}
D[\![p_1 \mid p_2]\!]\rho &= (D[\![p_1]\!]\rho) \mid^D (D[\![p_2]\!]\rho) \\
&\quad \text{by definition of } D[\![\ ]\!] \\
&= (\bigsqcup_n a_n(p_1)\rho) \mid^D (\bigsqcup_n a_n(p_2)\rho) \\
&\quad \text{by the induction hypothesis} \\
&= \bigsqcup_n ((a_n(p_1)\rho) \mid^D (a_n(p_2)p)) \\
&\quad \text{by continuity of } \mid^D \\
&= \bigsqcup_n a_{n+1}(p_1 \mid p_2)\rho \\
&\quad \text{by the definition of } a_n(p)\rho \\
&= \bigsqcup_n a_n(p_1 \mid p_2)\rho
\end{aligned}
$$

□

We may now combine all the results obtained so far and state the main theorem of this section:

Theorem 4.4.7 *(Full Abstraction for CHOCS processes) Assume that the set Names is finite, then:*

$$p \lesssim_\omega q \iff D[\![p]\!] \sqsubseteq D[\![q]\!]$$

PROOF:

$$\begin{aligned}
p \lesssim_\omega q &\iff \forall n.p \lesssim_n q \\
&\quad \text{by definition 2.7.3} \\
&\iff \forall n.p^n \lesssim_n q^n \\
&\quad \text{by proposition 2.8.10} \\
&\iff \forall n.p^n \lesssim^B q^n \\
&\quad \text{by proposition 4.3.5} \\
&\iff \forall n.D[\![p^n]\!] \sqsubseteq D[\![q^n]\!] \\
&\quad \text{by proposition 4.4.5} \\
&\iff \forall n.a_n(p) \sqsubseteq a_n(q) \\
&\quad \text{by continuity} \\
&\iff \bigsqcup_n a_n(p) \sqsubseteq \bigsqcup_n a_n(q) \\
&\quad \text{by proposition 4.4.6} \\
&\iff D[\![p]\!] \sqsubseteq D[\![q]\!]
\end{aligned}$$

□

The full abstraction result for CHOCS processes is limited in two ways. It only applies under the assumption that the set of port names *Names* is finite. As discussed in section 2.8 this is not a significant constraint and from an implementational point of view it is quite natural. The other limitation is that the theorem is stated in terms of the preorder $\lesssim_\omega$ and not in terms of $\lesssim^B$. This restriction is due to the well known impossibility of modelling unbounded nondeterminism in the Plotkin Power Domain. We may consider the preorder $\lesssim_\omega$ as representing the "finitary" part of $\lesssim^B$ in line with the view of Abramsky [Abr87a, Abr90a].

4.5 Recursion

Let us use the denotational semantics of CHOCS to obtain a much simpler proof of the simulation of recursion theorem i.e.:

$$\mathtt{rec}\, x.\tau.(p\backslash a) \sim Y_x[p]$$

Let $[\![\mathtt{rec}\, x.p]\!]\rho = \mathtt{fix}\ \lambda d.[\![p]\!]\rho[d/x]$ (i.e. we give a least fixed point semantics to recursion) and let us demonstrate that

$$\forall \rho.[\![Y_x[p]]\!]\rho = [\![\mathtt{rec}\, x.\tau.(p\backslash a)]\!]\rho$$

To see this we apply the semantic equations given in definition 4.4.1:

$$\begin{aligned} D[\![W_x[p]]\!]\rho &= FD(\lambda d.D[\![p[(x \mid a!x.nil)\backslash a/x]]\!]\rho[d/x]) \\ &= (\bigcup\{a?^D(d, D[\![p[(x \mid a!x.nil)\backslash a/x]]\!]\rho[d/x]) \ :\ d \in D\})^* \end{aligned}$$

$$\begin{aligned} & D[\![Y_x[p]]\!]\rho \\ = \ & (D[\![W_x[p]]\!]\rho \mid^D a!^D(D[\![W_x[p]]\!]\rho, \emptyset))\backslash a^D \\ = \ & ((\bigcup\{a?^D(d, D[\![p[(x \mid a!x.nil)\backslash a/x]]\!]\rho[d/x] \mid^D a!^D(D[\![W_x[p]]\!]\rho, \emptyset)) \ :\ d \in D\})^* \\ & \uplus a!^D(D[\![W_x[p]]\!]\rho, D[\![W_x[p]]\!]\rho \mid^D \emptyset) \\ & \uplus \tau^D(D[\![p[(x \mid a!x.nil)\backslash a/x]]\!]\rho[D[\![W_x[p]]\!]\rho/x]) \mid^D \emptyset))\backslash a^D \\ = \ & \tau^D((D[\![p[(x \mid a!x.nil)\backslash a/x]]\!]\rho[D[\![W_x[p]]\!]\rho/x])\backslash a^D) \\ = \ & \tau^D((D[\![p]\!]\rho[[\![(x \mid a!x.nil)\backslash a/x]]\!]\rho[D[\![W_x[p]]\!]\rho/x]/x)\backslash a^D) \\ = \ & \tau^D((D[\![p]\!]\rho[D[\![Y_x[p]]\!]\rho/x])\backslash a^D) \\ = \ & \tau^D(D[\![p\backslash a]\!]\rho[D[\![Y_x[p]]\!]\rho/x]) \\ = \ & D[\![\tau.(p\backslash a)]\!]\rho[D[\![Y_x[p]]\!]\rho/x] \end{aligned}$$

Since we have chosen the initial solution to the domain equation we have

$$D[\![Y_x[p]]\!]\rho = \mathtt{fix}\ \lambda d.(D[\![\tau.(p\backslash a)]\!]\rho[d/x]) = D[\![\mathtt{rec}\, x.\tau.(p\backslash a)]\!]\rho$$

Chapter 5

Plain CHOCS

In the previous three chapters we have studied the CHOCS calculus and shown how CCS can be extended with processes as first class objects. We have seen that we can model rather important computational phenomena such as recursion and the λ–Calculus. But some peculiarities may arise due to the dynamic binding of port names in processes sent and received. Port names that intuitively would be considered restricted or bound can become unbound and vice versa as e.g.

$$(b?x.(x \mid q)) \mid ((b!p'.p)\backslash a) \xrightarrow{\tau} p' \mid q \mid (p\backslash a) \tag{5.1}$$

$$(b?x.((x \mid q)\backslash a)) \mid (b!p'.p) \xrightarrow{\tau} ((p' \mid q)\backslash a) \mid p \tag{5.2}$$

In (5.1) any occurrence of a in p' becomes unbound after the communication even though we would expect them to be bound if we analyze the system before the communication. In (5.2) we have the opposite situation. Now any occurrence of a in p' unbound before the communication would be bound after the communication. These examples show that sending the process p' amounts to passing the text of p'. This is closely related to the treatment of function parameters in LISP as originally defined by McCarthy and often referred to as dynamic binding. This parameter mechanism is complicated to work with when analyzing the behaviour of programs from their text.

The approach in the previous three chapters was chosen because the semantics of CHOCS could be given as a straightforward extension of the CCS semantics and because it yielded simple algebraic laws. However, some of the laws included reference to the sort of the process (i.e. the set of port names the process might use). The calculation of the sort is either a costly calculation needing to run the process (or even worse needing all possible runs of the process) or a very rough approximation to the actual sort. This approximation often yields infinite sort for processes intuitively having finite sort.

Inspired by the idea presented in [EngNie86, MilParWal89] of the restriction operator $p\backslash a$ being a scope binder, which intuitively should bind all occurrences of a in p, we now present a calculus of higher order communicating systems with static binding of port names by restriction. We call this calculus Plain CHOCS.

We are looking for a calculus which has the property that scope extrusion, as we call the technique to take care of the problem in (5.1) above, will automatically take care of a static binding mechanism for the restriction operator. For example (5.1) becomes:

$$(b?x.(x \mid q)) \mid ((b!p'.p)\backslash a) \xrightarrow{\tau} (p'\{b/a\} \mid q \mid p\{b/a\})\backslash b \tag{5.3}$$

where $\{b/a\}$ is a label substitution such that b does not belong to the set of free names in q and the restriction will therefore not bind any port in q only in p and p'. Also scope intrusion, as we call the problem in (5.2), will be taken care of by a new definition of process substitution which takes the static nature of the restriction operator into account. Therefore (5.2) above becomes:

$$(b?x.((x \mid q)\backslash a)) \mid (b!p'.p) \xrightarrow{\tau} ((x \mid q)\backslash a)[p'/x] \mid p \equiv ((p' \mid q\{b/a\})\backslash b) \mid p \tag{5.4}$$

where $\{b/a\}$ is a label substitution such that b does not belong to the set of free names in p' and the restriction will therefore not bind any port in p' only in q. It turns out that it is interesting to have the capability of describing a kind of dynamic binding of port names of processes received in communication. This is obtained by allowing free names to be renamed to bound names upon reception of a process:

$$(b?x.((x[a \mapsto a'] \mid \{a'/a\}q)\backslash a')) \mid (b!p'.p) \xrightarrow{\tau} ((p'[a \mapsto a'] \mid \{a'/a\}q)\backslash a') \mid p \tag{5.5}$$

where a' does not belong to the set of free names of p' and q. This construction simulates the behaviour described in (5.2). However, we cannot program the behaviour described in (5.1) since in Plain CHOCS a bound name remains bound and can never become unbound again.

To illustrate these concepts, before presenting a formal syntax and semantics of the Plain CHOCS calculus, we first study a small example. In this example we shall rely on the knowledge of CHOCS and the above remarks.

Example 5.0.1 *The example consists of a simple user/resource system similar to the system studied in [EngNie86]. The system is constructed from a number of users, a resource manager and a resource. In this example the resource is a process which takes in a number and multiplies it by 2. A resource is obtained on the c channel, then put into use in parallel with the user process. Note how free names of the resource are renamed and bound when received by the user process.*

$$
\begin{aligned}
U_1 &= c?x.(x[b \mapsto a] \mid a!8.a?y.d_1!y.nil)\backslash a \\
U_2 &= c?x.(x[b \mapsto a] \mid a!5.a?y.d_2!y.nil)\backslash a \\
RM &= (c!(R).fin?.RM)\backslash fin \\
R &= b?x.b![2 * x].fin!.nil \\
SYS &= (U_1 \mid U_2 \mid RM)\backslash c
\end{aligned}
$$

The $fin!$ signal from the resource R tells the resource manager RM when the resource has finished its task for a user. The resource manager can then (recursively) restore itself and thus provide a resource for other users. The restriction of fin ensures that there is a private communication channel between resource and resource manager which cannot be interfered by any user process.

It is interesting to observe how the system executes and how scope extrusion takes care of preserving private links with the sending process. We give an example of one execution sequence where U_2 gets the resource first.

$SYS = (U_1 \mid U_2 \mid RM)\backslash c$

$\downarrow_\tau$ Since $U_2 \xrightarrow{c?x} U_2' = (x[b \mapsto a] \mid a!5.a?y.d_2!y.nil)\backslash a$ and $RM \xrightarrow{c!\{fin\}^R} RM' = fin?.RM$

$(U_1 \mid ((R[b \mapsto a] \mid a!5.a?y.d_2!y.nil)\backslash a \mid fin?.RM)\backslash fin)\backslash c$

$\downarrow_\tau$ Since $R[b \mapsto a] \xrightarrow{b?x} b![2*x].fin!.nil$ and $a!5.a?y.d_2!y.nil \xrightarrow{a!5} a?y.d_2!y.nil$

$(U_1 \mid (((b![2 * 5].fin!.nil)[b \mapsto a] \mid a?y.d_2!y.nil)\backslash a \mid fin?.RM)\backslash fin)\backslash c$

$\downarrow_\tau$ Since $b![2*5].fin!.nil \xrightarrow{b!10} fin!.nil$ and $a?y.d_2!y.nil \xrightarrow{a?y} d_2!y.nil$

$(U_1 \mid (((fin!.nil)[b \mapsto a] \mid d_2!10.nil)\backslash a \mid fin?.RM)\backslash fin)\backslash c$

$\downarrow_{d_2!10}$ Since $d_2!10.nil \xrightarrow{d_2!10} nil$

$(U_1 \mid (((fin!.nil)[b \mapsto a] \mid nil)\backslash a \mid fin?.RM)\backslash fin)\backslash c$

$\downarrow_\tau$ Since $fin!.nil \xrightarrow{fin!} nil$ and $fin?.RM \xrightarrow{fin?} RM$

$(U_1 \mid ((nil[b \mapsto a] \mid nil)\backslash a \mid RM)\backslash fin)\backslash c$

$\downarrow_\tau$

$\vdots$

This derivation of transitions illustrates how the system may evolve. However, the linear representation of the system in the Plain CHOCS syntax does not show very well how the underlying process network dynamically reconfigures itself. As an attempt to illustrate this the following cartoon is intended to show how the system evolves spacially when going through the first of the above transitions:

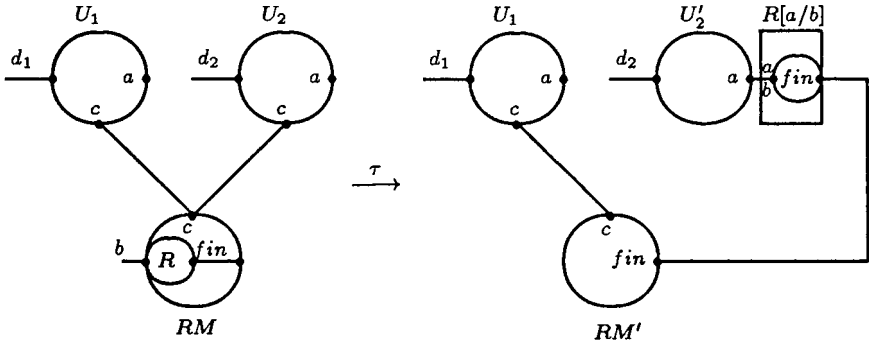

Fig 5.0.1: Dynamic reconfiguration of user/resource system.

We have adopted the convention from the process diagrams in [MilParWal89] and displayed private links inside the circles representing processes and public links along the edges of the connections. The box around the resource R is symbolizing the renaming of the public name b to the private name a which is not a process, but more like an encapsulation construct.

It is interesting to note that should a user make copies of the received resource i.e.: $U_3 = c?x(x[b \mapsto a] \mid x[b \mapsto a] \mid a!4.a?y.d_3!y.nil)\backslash c$ *then the resource manager will restore itself after the first copy has finished its task. Since the signal is private between the resource manager and the particular copy of the resource sent to* U_3 *the signal from the "second" copy will be ignored (or rather the "second" copy will be in the state* $fin!.nil$ *which is equivalent to nil since no process is able to match this signal).* U_3 *will nondeterministically send out either* 8 *or* 16 *on* d_3 *depending on whether only one copy of the resource or both copies are used.*

Note that the number of users and resources is not hard wired into the system. As for the system studied in [EngNie86] we may add any number of users or resources without changing the structure of the overall system e.g.:

$$SYS_1 = (U_1 \mid \ldots \mid U_n \mid RM_1 \mid \ldots RM_m)\backslash c$$

The above system is very simple, but it easily generalizes to systems with a queue system for resource requests from users, multiple resources or even systems where the resource is returned to the resource manager instead of just stopping and allowing a new copy to be used. Some quite elaborate examples of user/resource systems with the above facilities which use process passing have been studied by Cozens in [Coz90]. This work presents a promising motivation for the use of process passing in system description.

5.1 Syntax and Semantics

The syntax of Plain CHOCS is essentially that of "dynamic" CHOCS with a restricted renaming construct.

Processes are built from the inactive process nil, three types of action prefixing, often referred to as input, output and tau prefix, (nondeterministic) choice, parallel composition, restriction, renaming and variables to be bound by input prefix. We presuppose an infinite set *Names* (the set of port names) ranged over by $a, b, c, ..$ and an infinite set V of process variables ranged over by $x, y, z, ...$ We denote by Pr the set of expressions built according to the following syntax:

$$p ::= nil \mid a?x.p \mid a!p'.p \mid \tau.p \mid p + p' \mid p \mid p' \mid p\backslash a \mid p[a \mapsto b] \mid x$$

To avoid heavy use of brackets we adopt the following precedence of operators: restriction or renaming > prefix > parallel composition > choice.
We shall write $p[S]$ for $p[a \mapsto b]$ where $S = a \mapsto b$ and let $Dom(S) = \{a\}$ and $Im(S) = \{b\}$. The operator $\backslash a$ acts as a kind of λ–binder for port names (elements of *Names*) in a sense to be formalized later, e.g. we have a notion of α–convertibility of restricted names. To formalize this we define the set of free names $fn(p)$ of a process p.

Definition 5.1.1 *We define free names $fn(p)$ structurally on p:*

$$\begin{aligned}
fn(nil) &= \emptyset \\
fn(a?x.p) &= \{a\} \cup fn(p) \\
fn(a!p'.p) &= \{a\} \cup fn(p') \cup fn(p) \\
fn(\tau.p) &= fn(p) \\
fn(p + p') &= fn(p) \cup fn(p') \\
fn(p \mid p') &= fn(p) \cup fn(p') \\
fn(p\backslash a) &= fn(p) \setminus \{a\} \\
fn(p[S]) &= fn(p) \cup Dom(S) \cup Im(S) \\
fn(x) &= \emptyset
\end{aligned}$$

The set of free names of processes constructed using the renaming construct carries a potential overhead since it is not necessarily the case that the names in $Dom(S) \cup Im(S)$ are going to be used, but the overhead is necessary since we may receive processes in communication with free names which will be renamed by S. The free names of Plain CHOCS processes are going to play an important rôle in the

definition of the semantics of the language and as we shall see in the next section, where we define a notion of equivalence, the free names are the windows through which we can observe the processes. As opposed to the static sort of dynamic CHOCS we point out that processes to be sent contribute to the free names of the overall system whereas the empty set of free names is ascribed to process variables.

We may need to syntactically substitute one port name for another. Using the above definition we may now define a label substitution.

Definition 5.1.2 *First for $a, b, c \in Names$ let*

$$\{b/c\}a = \begin{cases} b & \text{if } c = a \\ a & \text{otherwise} \end{cases}$$

Then label substitution $\{b/c\}p$ is defined structurally on p:

$$\begin{aligned}
\{b/c\}nil &\equiv nil \\
\{b/c\}(a?x.p) &\equiv (\{b/c\}a)?x.(\{b/c\}p) \\
\{b/c\}(a!p'.p) &\equiv (\{b/c\}a)!(\{b/c\}p').(\{b/c\}p) \\
\{b/c\}(\tau.p) &\equiv \tau.(\{b/c\}p) \\
\{b/c\}(p + p') &\equiv (\{b/c\}p) + (\{b/c\}p') \\
\{b/c\}(p \mid p') &\equiv (\{b/c\}p) \mid (\{b/c\}p') \\
\{b/c\}(p\backslash a) &\equiv \begin{cases} p\backslash a & \text{if } a = c \\ (\{b/c\}(\{d/a\}p))\backslash d & \text{otherwise for some } d \notin fn(p\backslash a) \cup \{b\} \end{cases} \\
\{b/c\}(p[a \mapsto a']) &\equiv (\{b/c\}p)[(\{b/c\}a) \mapsto (\{b/c\}a')] \\
\{b/c\}(x) &\equiv x
\end{aligned}$$

Input prefix is a variable binder. This implies a notion of free and bound variables.

Definition 5.1.3 *We define the set of free variables $FV(p)$ structurally on p:*

$$\begin{aligned}
FV(nil) &= \emptyset \\
FV(a?x.p) &= FV(p) \setminus \{x\} \\
FV(a!p'.p) &= FV(p) \cup FV(p') \\
FV(\tau.p) &= FV(p) \\
FV(p + p') &= FV(p) \cup FV(p') \\
FV(p \mid p') &= FV(p) \cup FV(p')
\end{aligned}$$

$$\begin{aligned} FV(p\backslash a) &= FV(p) \\ FV(p[S]) &= FV(p) \\ FV(x) &= \{x\} \end{aligned}$$

A variable which is not free i.e. does not belong to $FV(p)$ is said to be bound in p.

An expression p is closed if $FV(p) = \emptyset$. Closed expressions are referred to as processes. The set of closed expressions is denoted by CPr.

To allow processes received in communication to be used we need a way of substituting the received processes for bound variables. We shall use the definition of label substitution to avoid unintentional binding of free names when processes are substituted.

Definition 5.1.4 *The substitution $p[q/x]$ is defined structurally on p:*

$$\begin{aligned}
nil[q/x] &\equiv nil \\
(a?y.p)[q/x] &\equiv \begin{cases} a?y.(p[q/x]) & \text{if } y \neq x \text{ and } y \notin FV(q) \\ a?z.((p[z/y])[q/x]) & \text{otherwise} \\ & z \notin FV(p) \cup FV(q) \cup \{x,y\} \end{cases} \\
(a!p'.p)[q/x] &\equiv a!(p'[q/x]).(p[q/x]) \\
(\tau.p)[q/x] &\equiv \tau.(p[q/x]) \\
(p + p')[q/x] &\equiv (p[q/x]) + (p'[q/x]) \\
(p \mid p')[q/x] &\equiv (p[q/x]) \mid (p'[q/x]) \\
(p\backslash a)[q/x] &\equiv (((\{d/a\}p)[q/x])\backslash d \text{ for some } d \notin (fn(p\backslash a) \cup fn(q)) \\
(p[S])[q/x] &\equiv (p[q/x])[S] \\
y[q/x] &\equiv \begin{cases} q & \text{if } x = y \\ y & \text{otherwise} \end{cases}
\end{aligned}$$

The only difference between the above definition of substitution and the one given for dynamic CHOCS in definition 2.2.3 is in the clause for restriction. In the above definition we ensure that we do not restrict names in q.

Here are a few useful properties of substitution:

Proposition 5.1.5

1. *If $x \neq y$ then $p[p'/x][p''/y] \equiv p[p''/y][p'[p''/y]/x]$.*
2. *$p[p'/x] \equiv p$ if $x \notin FV(p)$.*

PROOF: The proof of 1. is easily established by structural induction on p. Then 2. is a corollary of 1. □

With the above machinery in hand we may now give the operational semantics for Plain CHOCS. The operational semantics is given in terms of a labelled transition system in the style of [Plo81]. The transition relation $\rightarrow$ is a family of binary labelled relations $\xrightarrow{\Gamma}$ between elements of CPr (processes) and $CPr \cup [CPr \rightarrow CPr]$ (either processes or functions from processes to processes) of the form $p \xrightarrow{\Gamma} p'$. The action Γ may have one of the following forms: $a?x$, $a!_B p$, τ, where $a \in Names$, $B \subseteq Names$, $x \in V$ and $p \in CPr$. Let the bound names bn of an action be defined as:

$$bn(\Gamma) = \begin{cases} B & \text{if } \Gamma = a!_B p' \\ \emptyset & \text{otherwise} \end{cases}$$

In the definition of the semantics of Plain CHOCS it is convenient to write $p\backslash B$ where $B \subseteq Names$ is a finite set: $p\backslash B$ is shorthand for $p\backslash b_1 \ldots \backslash b_n$ where $B = \{b_1, \ldots, b_n\}$ and p if $B = \emptyset$.

Definition 5.1.6

Let $\rightarrow$ be the smallest transition relation closed under the rules of table 5.1.1.

---

input $\quad a?x.p \xrightarrow{a?x} p$

output $\quad a!p'.p \xrightarrow{a!_\emptyset p'} p$

tau $\quad \tau.p \xrightarrow{\tau} p$

choice $\quad \dfrac{p \xrightarrow{\Gamma} p''}{p + q \xrightarrow{\Gamma} p''}$

par $\quad \dfrac{p \xrightarrow{\Gamma} p''}{p \mid q \xrightarrow{\Gamma} p'' \mid q} \quad , bn(\Gamma) \cap fn(q) = \emptyset$

ren $\quad \dfrac{p \xrightarrow{a?x} p''}{p[S] \xrightarrow{S(a)?x} p''[S]}$

$$\frac{p \xrightarrow{a!_B p'} p''}{p[S] \xrightarrow{S(a)!_B p'} p''[S]} \quad , B \cap (Dom(S) \cup Im(S)) = \emptyset$$

$$\frac{p \xrightarrow{\tau} p''}{p[S] \xrightarrow{\tau} p''[S]}$$

res $$\frac{p \xrightarrow{a?x} p''}{p\backslash b \xrightarrow{a?x} p''\backslash b} \quad , a \neq b$$

$$\frac{p \xrightarrow{a!_B p'} p''}{p\backslash b \xrightarrow{a!_B p'\backslash b} p''\backslash b} \quad , a \neq b, b \notin (fn(p') \cap fn(p''))$$

$$\frac{p \xrightarrow{\tau} p''}{p\backslash b \xrightarrow{\tau} p''\backslash b}$$

open $$\frac{p \xrightarrow{a!_B p'} p''}{p\backslash c \xrightarrow{a!_{B\cup\{d\}}(\{d/c\}p')} \{d/c\}p''} \quad , a \neq c, d \notin fn(p\backslash c), c \in (fn(p') \cap fn(p'')) \setminus B$$

com-close $$\frac{p \xrightarrow{a?x} p' \; q \xrightarrow{a!_B q'} q''}{p \mid q \xrightarrow{\tau} (p'[q'/x] \mid q'')\backslash B} \quad , B \cap fn(p') = \emptyset$$

non-struct $$\frac{p \xrightarrow{a!_B p'} p''}{p \xrightarrow{a!_{B'} p'} p''} \quad , B \cap (fn(p') \cup fn(p'')) = B' \cap (fn(p') \cup fn(p''))$$

The choice, par, com-close rules have symmetric counterparts.
Table 5.1.1: Operational semantics for Plain CHOCS

The structure of this transition system is tailored to cater for the behaviour we have in mind for systems like those described by (5.3) and (5.4) in the introduction to this chapter, but it also carries some philosophy of its own. The three kinds of actions yield the following types of transitions or observations:

Input action $p \xrightarrow{a?x} p'$, this kind of transitions may be interpreted as, "the process p is capable of receiving on channel a". We only allow transitions of this kind where $p \in CPr$ and $p' \in CPr \rightarrow CPr$. We want to model input transitions in such a way that no further observations are possible until a value is supplied. The reason for this is both technical and philosophical. Technically it ensures that we do not "rewrite" to open terms which, without care, could lead to confusion of free variables e.g.: $a?x.x \mid b?x.x \xrightarrow{b?x} a?x.x \mid x \xrightarrow{a?x} x \mid x$. Philosophically it follows a point of view of only observing systems by atomic observations or combinations of atomic observations. The input observations

consist of observing that input on channel a is possible and the systems readiness to accept a value. To make further observations about this process we have to supply a value say $q \in CPr$ and observe the system $p'[q/x]$ with this value. A more suggestive notation would perhaps be $p \xrightarrow{a?} \lambda x.p'$, but it is not essential in the present calculus since x only acts as a place holder. We have chosen the notation $p \xrightarrow{a?x} p'$ since p' is describable in the Plain CHOCS syntax. We could extend the above transition system to open expressions. To avoid confusion of variables introduced by the input-rule we would have to ascribe the par-rule by the additional constraint $FV(p'') \cap FV(q) = \emptyset$. We have not done this since the theory of equivalence will be defined in terms of closed expressions and extended to open expressions using the definition for closed expressions.

Output actions (with scope extrusion) $p \xrightarrow{a!_B p'} p''$, if $B = \emptyset$ this kind of transitions may be interpreted as, "the process p can output the process p' on channel a and in doing so become p''". To observe this action we observe that output on channel a is possible, to make further observations we have to observe both the value p' and the resulting state p''. If p' and p'' share some private channels these will be in the set B and a scope extrusion is necessary. We observe this by the combined observation as for normal output actions together with the additional observation of the scope extrusion. A more suggestive notation for output transitions might be $p \xrightarrow{a!} (B, p', p'')$. We refer to p' as the emitted process and p as the emitting process or rather p'' since this is the state of the system after emitting p'.

Silent actions $p \xrightarrow{\tau} p'$, this kind of transitions may be interpreted as, "the process p can do an internal or silent move and in doing so become the process p'". Silent actions arise from communications between two processes. Since communications are the only computations in our calculus these are in a sense the real computations of the system. τ-transitions may of course arise from processes of the form $\tau.p$ as well.

The input, output and com-close rules form the basis for inferring a communication between two agents. In the rules of table 5.1.1 all transitions of the form $p \xrightarrow{a?x} p'$ have the property that $p \in CPr$ and $p' \in CPr \rightarrow CPr$ and all transitions of the form $p \xrightarrow{a!_B p'} p''$ have the property $p, p', p'' \in CPr$, therefore $p'[q'/x] \in CPr$ in the com-close rule. This set of rules gives an operational description where input is modelled as a function and communication acts as a generalized application. This is very different from the nature of inferring communication in dynamic CHOCS

(or in CCS with value passing [Mil80]). In dynamic CHOCS we have the following three rules as the basis for inferring communication:

$$a?x.p \xrightarrow{a?p'} p[p'/x] \quad a!p'.p \xrightarrow{a!p'} p \quad \frac{p \xrightarrow{a?p'} p'' \; q \xrightarrow{a!p'} q''}{p \mid q \xrightarrow{\tau} p'' \mid q''}$$

Note that in these rules the transition relation is always between elements of CPr. One way of interpreting the above rules is to say that the process with input prefix knows all the possible values it can receive. What it does is to offer a (an infinite) choice between all the possible new states and when the communication takes place it is only a signal from the output process to the input process telling which value to use (choose). The value is not really transmitted. This viewpoint is further strengthened by the (elegant) way of encoding value passing in SCCS as described in [Mil83]. In [MilParWal89] a scheme similar to the above for inferring communication has been termed early instantiation, referring to the fact that the instantiation of the free variable takes place in the axiom for input prefix as opposed to the scheme used in table 5.1.1. The scheme we are using has been termed late instantiation, though there is a difference since processes are allowed to offer new transitions after an input transition. This calls for some machinery to ensure that free variables are not confused. We have chosen the late instantiation scheme with the restriction that $p' \in CPr \rightarrow CPr$ in $p \xrightarrow{a?x} p'$ for the reasons given above; late instantiation also seems necessary for the scope opening and closing rules for the restriction operator. The rules concerning the restriction operator have several alternatives, e.g. in dynamic CHOCS this operator does not bind names in the process emitted but only in the emitting process as the examples in the introduction show. Another possibility would be the following rule

$$\frac{p \xrightarrow{a!p'} p''}{p\backslash b \xrightarrow{a!(p'\backslash b)} p''\backslash b} \quad a \neq b$$

i.e. the res-rule without the side condition $b \notin fn(p') \cap fn(p'')$. This approach would ensure that bound names would be bound both in the emitted process and in the emitting process, but it is too restrictive since they can not use the local channel to communicate with one another since the $\backslash b$ encapsulates the process. To elaborate on this we follow the ideas of [EngNie86, MilParWal89] and adopt the restriction rule above, but with the mentioned side condition. We also introduce two new rules; open and com-close. The opening rule signals that in the emitted process there are some bound names, names which are shared with the emitting process. The com-close rule ensures that exported restrictions are reintroduced

upon reception. The condition on this rule ensures that we do not bind free names in the receiving processes. When $B = \emptyset$ this rule is just a communication rule.

We conclude this section by listing a few useful properties of the transition system defined in table 5.1.1

Proposition 5.1.7

1. *If* $p \xrightarrow{a!_{B}p'} p''$ *and* $b \notin fn(p) \cup B$ *then* $p \xrightarrow{a!_{((B \setminus \{c\}) \cup \{b\})}\{b/c\}p'} \{b/c\}p''$ *for any* $c \in B$.
2. *If* $p \xrightarrow{a!_{B}p'} p''$ *then* $p \xrightarrow{a!_{B'}p'} p''$ *for some* B' *with* $B \cap (fn(p') \cup fn(p'')) = B' \cap (fn(p') \cup fn(p''))$ *and* $B' \subseteq fn(p') \cap fn(p'')$ *and* $B' \cap fn(p) = \emptyset$.
3. *If* $p \xrightarrow{a?x} p'$ *then* $fn(p') \subseteq fn(p)$.
4. *If* $p \xrightarrow{a!_{B}p'} p''$ *then* $fn(p') \subseteq fn(p) \cup B$ *and* $fn(p'') \subseteq fn(p) \cup B$
5. *If* $p \xrightarrow{\tau} p'$ *then* $fn(p') \subseteq fn(p)$.

PROOF: By induction on the length of the inference used to establish the transition and cases of the structure of p. □

5.2 Bisimulation and Equivalence

In the previous section we presented the operational semantics for Plain CHOCS in terms of a labelled transition system. The structure of this transition system resembles a merge between the applicative transition systems of [Abr90] and the higher order communication trees used in the semantics for CHOCS in chapter 2. The transition relation $\rightarrow$ forms the basis for the observations we can make about processes, but it is in itself too shallow to use as a distinguishing equivalence. Instead we use the notion of (bi)simulation [Par81, Mil83] redefined to the kind of observations the transition allows:

Definition 5.2.1 *An* applicative higher order simulation R *is a binary relation on CPr such that whenever* pRq *and* $a \in Names$ *then:*

(i) Whenever $p \xrightarrow{a?x} p'$, *then* $q \xrightarrow{a?y} q'$ *for some* q', y *and* $p'[r/x]Rq'[r/y]$ *for all* $r \in CPr$

(ii) Whenever $p \xrightarrow{a!_{B}p'} p''$, *then* $q \xrightarrow{a!_{B}q'} q''$ *for some* q', q'' *with* $B \cap (fn(p) \cup fn(q)) = \emptyset$ *and* $p'Rq'$ *and* $p''Rq''$

(iii) Whenever $p \xrightarrow{\tau} p'$, *then* $q \xrightarrow{\tau} q'$ *for some* q' *with* $p'Rq'$

A relation R *is an* applicative higher order bisimulation *if both it and its inverse are applicative higher order simulations.*
Two processes p *and* q *are said to be bisimulation equivalent iff there exists an applicative higher order bisimulation* R *containing* (p,q). *In this case we write* $p \dot{\sim} q$.

The first clause of this predicate is essentially the clause for applicative (bi)simulation in the Lazy-λ-Calculus as defined in [Abr90]. It can be interpreted as saying that if p can do an input on channel a and become the function p', then q must match this by being able to input on channel a and become the function q' and for all values (arguments) we can receive on this channel the resulting process together with this value should continue to simulate each other. The second clause with $B = \emptyset$ and the third clause are similar to the clauses of higher order bisimulation defined in definition 2.3.2. The second clause supports a kind of black box view of the processes being sent. If p can output a process p' on channel a and in doing so become p'', then q should be able to output some q' on channel a and in doing so become q'' and p' and q', as well as p'' and q'' should be equivalent. The second clause with $B \neq \emptyset$ is a generalization of the clause for scope extrusion in the strong ground bisimulation defined in [MilParWal89]. B is a set of private channels between p' and p''. These channels are exported from their original scope and are intended to become restricted upon reception.

Now for $R \subseteq Pr^2$ we can define $\mathcal{AB}(R)$ as the set of pairs (p,q) satisfying for all $a \in Act$ the clauses (i) to (iii) and their symmetric counterparts above. From this definition it follows immediately that R is a bisimulation just in the case $R \subseteq \mathcal{AB}(R)$. Also, $\mathcal{AB}$ is easily seen to be a monotone endofunction on the complete lattice of binary relations (over CPr) under subset inclusion. Standard fixed point results, due to Tarski [Tar55], yield that a maximal fixed point for $\mathcal{AB}$ exists and is defined as $\bigcup\{R : R \subseteq \mathcal{AB}(R)\}$. This maximal fixed point actually equals $\dot{\sim}$.

Proposition 5.2.2 $\dot{\sim}$ *is an equivalence*

PROOF: Reflexivity and symmetry are straightforward since the relation $Id = \{(p,p) : p \in CPr\}$ is an applicative higher order bisimulation and the relation $R^T = \{(q,p) : (p,q)\}$ is an applicative higher order bisimulation if R is an applicative higher order bisimulation. Transitivity follows from the fact that if R and S are applicative higher order bisimulations then $R \circ S$ is an applicative higher order bisimulation. To see this observe that if $(p_1,p_3) \in R \circ S$ and $p_1 \xrightarrow{a?x} p_1'$ then for

some p_2 such that $(p_1, p_2) \in R$ and $(p_2, p_3) \in S$ we have $p_2 \xrightarrow{a?y} p_2'$ for some p_2', y and $(p_1'[r/x], p_2'[r/y]) \in R$ for all r and we have $p_3 \xrightarrow{a?z} p_3'$ for some p_3', z and $(p_2'[r/y], p_3'[r/z]) \in S$ for all r. Thus $(p_1'[r/x], p_3'[r/z]) \in R \circ S$ for all r.
If $p_1 \xrightarrow{a!_B p_1'} p_1''$ then for some p_2 such that $(p_1, p_2) \in R$ and $(p_2, p_3) \in S$ and $B \cap (fn(p_1) \cup fn(p_2)) = \emptyset$ and $B \cap (fn(p_2) \cup fn(p_3)) = \emptyset$ which implies $B \cap (fn(p_1) \cup fn(p_3)) = \emptyset$ we have $p_2 \xrightarrow{a!_B p_2'} p_2''$ for some p_2', p_2'' such that $(p_1', p_2') \in R$ and $(p_1'', p_2'') \in R$ and we have $p_3 \xrightarrow{a!_B p_3'} p_3''$ for some p_3', p_3'' such that $(p_2', p_3') \in S$ and $(p_2'', p_3'') \in S$. Thus $(p_1', p_3') \in R \circ S$ and $(p_1'', p_3'') \in R \circ S$.
If $p_1 \xrightarrow{\tau} p_1''$ then for some p_2 such that $(p_1, p_2) \in R$ and $(p_2, p_3) \in S$ we have $p_2 \xrightarrow{\tau} p_2''$ for some p_2'' such that $(p_1'', p_2'') \in R$ and we have $p_3 \xrightarrow{\tau} p_3''$ for some p_3'' such that $(p_2'', p_3'') \in S$. Thus $(p_1'', p_3'') \in R \circ S$. □

Lemma 5.2.3 *If $p \dot{\sim} q$ and $b \notin fn(p) \cup fn(q)$ then $\{b/a\}p \dot{\sim} \{b/a\}q$.*

Before relating the process constructions of Plain CHOCS to the underlying semantic equivalence $\dot{\sim}$ we present a technical construction called an applicative higher order bisimulation up to restriction. This construction resembles the higher order bisimulation up to $\sim$ presented for CHOCS in section 2.6.

Definition 5.2.4 *An* applicative higher order simulation up to restriction *R is a binary relation on CPr such that whenever pRq and $a \in$ Names then:*

(i) If $b \notin fn(p) \cup fn(q)$ then $\{b/a\}pR\{b/a\}q$

(ii) Whenever $p \xrightarrow{a?x} p'$, then $q \xrightarrow{a?y} q'$ for some q', y and $p'[r/x]Rq'[r/y]$ for all $r \in CPr$

(iii) Whenever $p \xrightarrow{a!_B p'} p''$, then $q \xrightarrow{a_B!q'} q''$ for some q', q'' with $B \cap (fn(p) \cup fn(q)) = \emptyset$, $p'Rq'$ and $p''Rq''$

(iv) Whenever $p \xrightarrow{\tau} p'$, then $q \xrightarrow{\tau} q'$ for some q' and either $p'Rq'$ or for some p'', q'' and b: $p' \equiv p''\backslash b$, $q' \equiv q''\backslash b$ and $p''Rq''$

A relation R is an applicative higher order bisimulation up to restriction *if both it and its inverse are applicative higher order simulations up to restriction.*

Lemma 5.2.5 *If R is an applicative higher order bisimulation up to restriction then $R \subseteq \dot{\sim}$.*

PROOF: We show that the relation $R^{\backslash} = \bigcup_{n\in\omega} R_n$ where

$$\begin{aligned} R_0 &= R \\ R_{n+1} &= \{(p\backslash b, q\backslash b) : (p,q) \in R_n, b \in Names\} \end{aligned}$$

is an applicative higher order bisimulation.
First we show by induction on n that if pR_nq and $c \notin fn(p) \cup fn(q)$ then $\{c/a\}pR_n\{c/a\}q$.
For $n = 0$ this is immediate from the definition of applicative higher order bisimulation up to restriction. Suppose $n > 0$ and $p\backslash bR_nq\backslash b$ where $pR_{n-1}q$ and $c \notin fn(p\backslash b) \cup fn(q\backslash b)$. If $a = b$ then $\{c/a\}(p\backslash b) \equiv p\backslash bR^{\backslash}q\backslash b \equiv \{c/a\}(q\backslash b)$. If $a \neq b$ then $\{c/a\}(p\backslash b) \equiv (\{c/a\}(\{b_1/b\}p))\backslash b_1R^{\backslash}(\{c/a\}(\{b_1/b\}q))\backslash b_1 \equiv \{c/a\}(q\backslash b)$
Next we show by induction on n that if pR_nq then

(i) Whenever $p \xrightarrow{a?x} p'$, then $q \xrightarrow{a?y} q'$ for some q', y and $p'[r/x]R^{\backslash}q'[r/y]$ for all $r \in CPr$

(ii) Whenever $p \xrightarrow{a!_Bp'} p''$, then $q \xrightarrow{a!_Bq'} q''$ for some q', q'' with $B \cap (fn(p) \cup fn(q)) = \emptyset$, $p'R^{\backslash}q'$ and $p''R^{\backslash}q''$

(iii) Whenever $p \xrightarrow{\tau} p'$, then $q \xrightarrow{\tau} q'$ for some q' and $p'R^{\backslash}q'$

$\underline{n = 0}$ This case is immediate from the fact that R_0 is an applicative higher order bisimulation up to restriction and from the definition of $R^{\backslash}$.

$\underline{n > 0}$ Suppose pR_nq where $p \equiv p_1\backslash b$ and $q \equiv q_1\backslash b$.

1. If $p \xrightarrow{a?x} p'$ this must have been inferred by the res-rule and $p_1 \xrightarrow{a?x} p_1'$ with $a \neq b$ and $p' \equiv p_1'\backslash b$. Then for some q_1', y we have $q_1 \xrightarrow{a?y} q_1'$ and $p_1'[r/x]R^{\backslash}q_1'[r/y]$ for all $r \in CPr$. Then $q \xrightarrow{a?y} q' \equiv q_1'\backslash b$ and for all $r \in CPr$ and some $c \notin fn(p_1'\backslash b) \cup fn(q_1'\backslash b) \cup fn(r)$ we have $p'[r/x] \equiv (\{c/b\}p_1')[r/x])\backslash cR^{\backslash}((\{c/b\}q_1')[r/x])\backslash c \equiv q'[r/x]$.

2. Suppose $B \cap (fn(p) \cup fn(q)) = \emptyset$. If $p \xrightarrow{a!_Bp'} p''$ and this has been inferred by the res-rule then $p_1 \xrightarrow{a!_Bp_1'} p_1''$ with $a \neq b$ and $b \notin B \cup (fn(p_1') \cap fn(p_1''))$ and $p' \equiv p_1'\backslash b$ and $p'' \equiv p_1''\backslash b$. So for some q_1', q_1'' we have $q_1 \xrightarrow{a!_Bq_1'} q_1''$ and $p_1'R^{\backslash}q_1'$ and $p_1''R^{\backslash}q_1''$. Thus $q \xrightarrow{a!_Bq'} q''$ with $q' \equiv q_1'\backslash b$ and $q'' \equiv q_1''\backslash b$ and $p'R^{\backslash}q'$ and $p''R^{\backslash}q''$.
If $p \xrightarrow{a!_Bp'} p''$ and this has been inferred by the open-rule then $p_1 \xrightarrow{a!_{B'}p_1'} p_1''$

with $a \neq b$, $B = B' \cup \{c\}$ and $b \in fn(p_1') \cup fn(p_1'')$ and $c \notin fn(p_1 \backslash b) \cup B'$ and $p' \equiv \{c/b\}p_1'$ and $p'' \equiv \{c/b\}p_1''$. So for some q_1', q_1'' we have $q_1 \xrightarrow{a!_{B'}q_1'} q_1''$ and $p_1' R \backslash q_1'$ and $p_1'' R \backslash q_1''$. Thus $q \xrightarrow{a!_B q'} q''$ with $q' \equiv \{c/b\}q_1'$ and $q'' \equiv \{c/b\}q_1''$ and $p' R \backslash q'$ and $p'' R \backslash q''$.
If $p \xrightarrow{a!_B p'} p''$ and this has been inferred by the non-struct-rule then $p \xrightarrow{a!_{B'}p'} p''$ with $B' \cap (fn(p') \cup fn(p'')) = B \cap (fn(p') \cup fn(p''))$. So for some q', q'' we have $q \xrightarrow{a!_{B'}q'} q''$ and $p' R \backslash q'$ and $p'' R \backslash q''$. Thus $B' \cap (fn(q') \cup fn(q'')) = B \cap (fn(q') \cup fn(q''))$ and $q \xrightarrow{a!_B q'} q''$ and we already know $p' R \backslash q'$ and $p'' R \backslash q''$.

3. If $p \xrightarrow{\tau} p'$ then $p_1 \xrightarrow{\tau} p_1'$ and $p' \equiv p_1' \backslash b$. Then $q_1 \xrightarrow{\tau} q_1'$ and $p_1' R \backslash q_1'$. Thus $q \xrightarrow{\tau} q' \equiv q_1'$ and $p' R \backslash q'$.

□

Let $\overline{x} = (x_1, \ldots, x_n)$ be a vector of variables of length n and $x_i \neq x_j$ if $i \neq j$. We also consider $\overline{x}$ as a set of variables $\{x_1, \ldots, x_n\}$ and we write $\overline{x} \subseteq FV(p)$ which means that the set $\overline{x}$ is a subset of $FV(p)$. Let $p[\overline{q}/\overline{x}]$ mean $(\ldots(p[q_1/x_1])\ldots)[q_n/x_n]$. We only consider substitutions of compatible vectors, i.e. of vectors of the same length. Let $\overline{q}_1 \dot{\sim} \overline{q}_2$ mean $q_{1_j} \dot{\sim} q_{2_j}$ for all $q_{i_j} \in \overline{q}_i$, $i \in 1,2$ and let $\overline{q}_i \in CPr$ mean $q_{i_j} \in CPr$ for all $q_{i_j} \in \overline{q}_i$.

Proposition 5.2.6 *$\dot{\sim}$ is a congruence relation on processes (closed expressions).*

1. $p[\overline{q}_1/\overline{x}] \dot{\sim} p[\overline{q}_2/\overline{x}]$ *if* $\overline{q}_1 \dot{\sim} \overline{q}_2$ *and* $\overline{x} \subseteq FV(p)$
2. $a?x.p \dot{\sim} a?x.q$ *if* $p[r/x] \dot{\sim} q[r/x]$ *for all* r
3. $a!p'.p \dot{\sim} a!q'.q$ *if* $p \dot{\sim} q$ *and* $p' \dot{\sim} q'$
4. $\tau.p \dot{\sim} \tau.q$ *if* $p \dot{\sim} q$
5. $p + p' \dot{\sim} q + q'$ *if* $p \dot{\sim} q$ *and* $p' \dot{\sim} q'$
6. $p \mid p' \dot{\sim} q \mid q'$ *if* $p \dot{\sim} q$ *and* $p' \dot{\sim} q'$
7. $p\backslash a \dot{\sim} q\backslash a$ *if* $p \dot{\sim} q$
8. $p[S] \dot{\sim} q[S]$ *if* $p \dot{\sim} q$

PROOF:

1. We prove this by showing that the relation ACR^*, the reflexive and transitive closure of ACR, where

$$ACR = \{(p[\overline{q}_1/\overline{x}], p[\overline{q}_2/\overline{x}]) \ : \ p \in Pr \ \& \ \overline{x} \subseteq FV(p) \ \& \ \overline{q}_1 \dot{\sim} \overline{q}_2 \ \& \ \overline{q}_i \in CPr\},$$

is an applicative higher order bisimulation up to restriction.

Note if $q_1 \dot{\sim} q_2$ then $(x[q_1/x], x[q_2/x]) \in ACR^*$ and we write $(q_1, q_2) \in ACR^*$.

We only show that ACR^* is an applicative higher order simulation up to restriction, symmetry of ACR^* then yields the result. To see that ACR^* is an applicative higher order simulation up to restriction we show that if $(p_1, p_2) \in ACR$ then $p_i \equiv p[\overline{q}_i/\overline{x}]$ and:

(i) If $b \notin fn(p[\overline{q}_1/\overline{x}]) \cup fn(p[\overline{q}_2/\overline{x}])$ then $\{b/a\}(p[\overline{q}_1/\overline{x}])ACR^*\{b/a\}(p[\overline{q}_2/\overline{x}])$

(ii) Whenever $p[\overline{q}_1/\overline{x}] \xrightarrow{a?x} p'$, then $p[\overline{q}_2/\overline{x}] \xrightarrow{a?y} q'$ for some q', y and $p'[r/x]ACR^*q'[r/y]$ for all $r \in CPr$

(iii) Whenever $p[\overline{q}_1/\overline{x}] \xrightarrow{a!_B p'} p''$, then $p[\overline{q}_2/\overline{x}] \xrightarrow{a!_B q'} q''$ for some q', q'' with $B \cap (fn(p) \cup fn(q)) = \emptyset$, $p'ACR^*q'$ and $p''ACR^*q''$

(iv) Whenever $p[\overline{q}_1/\overline{x}] \xrightarrow{\tau} p'$, then $p[\overline{q}_2/\overline{x}] \xrightarrow{\tau} q'$ for some q' and either $p'ACR^*q'$ or for some p'', q'' and b: $p' \equiv p''\backslash b$, $q' \equiv q''\backslash b$ and $p''ACR^*q''$

If $(p, q) \in ACR^*$ then there is a sequence $p_1 \ldots p_n$ such that $(p, p_1) \in ACR$, $(p_i, p_{i+1}) \in ACR$ for $1 \leq i < n$ and $(p_n, q) \in ACR$. The result then follows by induction on the length of the transitive sequence $p_1 \ldots p_n$ of ACR^*.
First (i) is easily proved by structural induction on p using lemma 5.2.3 in the case $p \equiv y$.

Next we show (ii)-(iv) simultaneously. We proceed by induction on the length of the inference used to establish the transitions of $p[\overline{q}_1/\overline{x}]$ and cases of the structure of p. We only need to consider transitions inferred by use of the structural rules since we may transform any derivation of a transitions into an equivalent one where we use the non-struct-rule excatly once after each application of a structural rule.

$\underline{p \equiv nil}$ Trivial since $p[\overline{q}_i/\overline{x}] \not\rightarrow$.

$\underline{p \equiv a?y.p_1}$ Assume $y \notin \overline{x}$ (otherwise use α-conversion on y). Then $p[\overline{q}_i/\overline{x}] \equiv a?y.(p_1[\overline{q}_i/\overline{x}])$ and $p[\overline{q}_i/\overline{x}] \xrightarrow{a?y} p_1[\overline{q}_i/\overline{x}]$. Since $FV(p_1) \subseteq (\overline{x} \cup \{y\})$ and $y \notin \overline{x}$ we have $(p_1[\overline{q}_1/\overline{x}])[r/y] \equiv p_1[\overline{q}_1, r/\overline{x}, y]ACR^*p_1[\overline{q}_2, r/\overline{x}, y] \equiv (p_1[\overline{q}_2/\overline{x}])[r/y]$ for all $r \in CPr$, since $r \dot{\sim} r$ and $\overline{q}_i$ are closed.

$\underline{p \equiv a!p_1.p_2}$ Then $p[\overline{q}_i/\overline{x}] \xrightarrow{a!_\emptyset(p_1[\overline{q}_i/\overline{x}])} (p_2[\overline{q}_i/\overline{x}])$ and $p_2[\overline{q}_1/\overline{x}]ACR^*p_2[\overline{q}_2/\overline{x}]$ and $p_1[\overline{q}_1/\overline{x}]ACR^*p_1[\overline{q}_2/\overline{x}]$

$\underline{p \equiv \tau.p_1}$ An argument similar to the argument given in the case above yields this case.

$\underline{p \equiv p_1 + p_2}$ If $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then

either $p_1[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ by a shorter inference. There are three cases depending on the structure of Γ. We show the case when $\Gamma = a?x$: By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?z} p''$ and $p'[r/x]ACR^*p''[r/z]$ for all $r \in CPr$. By the operational semantics for choice we have $(p_1+p_2)[\overline{q}_2/\overline{x}] \xrightarrow{a?z} p''$ which is a matching move.

or $p_2[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ and we may argue as above.

$\underline{p \equiv p_1 \mid p_2}$ If $p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p'$ then

either $p_1[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p_1'$ by a shorter inference and $p' \equiv p_1' \mid p_2[\overline{q}_1/\overline{x}]$. There are three cases depending on the structure of Γ:

$\Gamma = a?x$ Then by induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?z} p_1''$ and $(p_1'[r/x], p_1''[r/z]) \in ACR^*$ for all $r \in CPr$. Then by the operational semantics for parallel we have $(p_1 \mid p_2)[\overline{q}_2/\overline{x}] \equiv (p_1[\overline{q}_2/\overline{x}]) \mid (p_2[\overline{q}_2/\overline{x}]) \xrightarrow{a?z} p_1'' \mid p_2[\overline{q}_2/\overline{x}]$. Since $(p_1'[r/x], p_1''[r/z]) \in ACR^*$ for all $r \in CPr$ there exist p_3, $\overline{q}_1^{1'}$, $\overline{q}_2^{1'}$ and $\overline{x}'$ for each $r \in CPr$ such that $p_1'[r/x] \equiv p_3[\overline{q}_1^{1'}/\overline{x}']$ and $p_1''[r/z] \equiv p_3[\overline{q}_2^{1'}/\overline{x}']$ with $FV(p_3) \subseteq \overline{x}'$ and $\overline{q}_1^{1'} \dot{\sim} \overline{q}_2^{1'}$. We may assume $\overline{x}' \cap \overline{x} = \emptyset$ since if $\overline{x}' \cap \overline{x} \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup \overline{x}' \cup \overline{x}) = \emptyset$ and we have $p_3[\overline{q}_i^{1'}/\overline{x}'] \equiv (p_3[\overline{y}/\overline{x}'])[\overline{q}_i^{1'}/\overline{y}]$ by proposition 5.1.5. Thus $(p_1' \mid p_2[\overline{q}_1^2/\overline{x}])[r/x] \equiv p_1'[r/x] \mid p_2[\overline{q}_1^2/\overline{x}] \equiv (p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}' \cup \overline{x}]$ and $(p_1'' \mid p_2[\overline{q}_2^2/\overline{x}])[r/z] \equiv p_1''[r/z] \mid p_2[\overline{q}_2^2/\overline{x}] \equiv (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}' \cup \overline{x}]$ and $((p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}' \cup \overline{x}], (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}' \cup \overline{x}]) \in ACR^*$ for each $r \in CPr$ by proposition 5.1.5 since r and $q_i^{1'}$ and q_i^2 are all closed.

$\Gamma = a!_B p'$ Then $B \cap fn(p_2[\overline{q}_1/\overline{x}]) = \emptyset$. By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a!_B p''} p_1''$ with $(p', p'') \in ACR^*$ and $(p_1', p_1'') \in ACR^*$ and $B \cap (fn(p_1[\overline{q}_1/\overline{x}]) \cup fn(p_1[\overline{q}_2/\overline{x}])) = \emptyset$. Thus $B \cap fn(p_2[\overline{q}_2/\overline{x}]) = \emptyset$ and by the operational semantics for parallel $p_1[\overline{q}_2/\overline{x}] \mid p_2[\overline{q}_2/\overline{x}] \xrightarrow{a!_B p''} p_1'' \mid p_2[\overline{q}_2/\overline{x}]$.

Since $(p_1', p_1'') \in ACR^*$ there exist p_3, $\overline{q}_1^{1'}$, $\overline{q}_2^{1'}$ and $\overline{x}'$ such that $p_1' \equiv p_3[\overline{q}_1^{1'}/\overline{x}']$ and $p_1'' \equiv p_3[\overline{q}_2^{1'}/\overline{x}']$ with $FV(p_3) \subseteq \overline{x}'$ and $\overline{q}_1^{1'} \dot{\sim} \overline{q}_2^{1'}$. We may assume $\overline{x}' \cap \overline{x} = \emptyset$ since if $\overline{x}' \cap \overline{x} \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup \overline{x}' \cup \overline{x}) = \emptyset$ and we have $p_3[\overline{q}_i^{1'}/\overline{x}'] \equiv (p_3[\overline{y}/\overline{x}'])[\overline{q}_i^{1'}/\overline{y}]$ by proposition 5.1.5. Thus $(p_1' \mid p_2[\overline{q}_1^2/\overline{x}]) \equiv (p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}' \cup \overline{x}]$ and $(p_1'' \mid p_2[\overline{q}_2^2/\overline{x}]) \equiv (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}' \cup \overline{x}]$ and $((p_3 \mid p_2)[\overline{q}_1^{1'} \cup \overline{q}_1^2/\overline{x}' \cup \overline{x}], (p_3 \mid p_2)[\overline{q}_2^{1'} \cup \overline{q}_2^2/\overline{x}' \cup \overline{x}]) \in ACR^*$.

$\Gamma = \tau$ and we may argue as above.

or $p_2[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p_2'$ and we may argue as above.

or $\Gamma = \tau$ and w.l.o.g. $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a?x} p_1'$ and $p_2[\overline{q}_1/\overline{x}] \xrightarrow{a!_B r'} p_2'$ by shorter inferences and $p' \equiv (p_1'[r'/x] \mid p_2')\backslash B$ and $B \cap fn(p_1') = \emptyset$. By induction $p_2[\overline{q}_2/\overline{x}] \xrightarrow{a!_B r''} p_2''$ with $(r', r'') \in ACR^*$ and $(p_2', p_2'') \in ACR^*$ and $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?z} p_1''$ with $(p_1'[r/x], p_1''[r/z]) \in ACR^*$ for all $r \in CPr$. By proposition 5.1.7 we may assume that $B \cap fn(p_1'') = \emptyset$. By the operational semantics for parallel $(p_1 \mid p_2)[\overline{q}_2/\overline{x}] \xrightarrow{\tau} (p_1''[r''/z] \mid p_2'')\backslash B$. To see that $p_1'[r'/x] \mid p_2' ACR^* p_1''[r''/z] \mid p_2''$ and thus showing that ACR^* is an applicative higher order bisimulation up to restriction we observe that $(p_1'[r/x], p_1''[r/z]) \in ACR^*$ for all $r \in CPr$, in particular this is true for r'. Clearly $p_1''[r'/z] ACR^* p_1''[r''/z]$ since $r' ACR^* r''$ and if $(r', r'') \in ACR^*$ then $r' \equiv r_1[\overline{q}_1/\overline{x}]$ and $r'' \equiv r_1[\overline{q}_2/\overline{x}]$ for some r_1 with $FV(r_1) \subseteq \overline{x}$ and $\overline{q}_1 \dot{\sim} \overline{q}_2$ for some closed $\overline{q}_i$. Then $p_1''[r'/z] \equiv p_1''[r_1[\overline{q}_1/\overline{x}]/z] \equiv (p_1''[r_1/z])[\overline{q}_1/\overline{x}]$ and $p_1''[r''/z] \equiv p_1''[r_1[\overline{q}_2/\overline{x}]/z] \equiv (p_1''[r_1/z])[\overline{q}_2/\overline{x}]$ since $FV(p_1'') = \{z\}$ and $FV(r_1) = \overline{x}$ we have $FV(p_1''[r_1/z]) = \overline{x}$ and $((p_1''[r_1/z])[\overline{q}_1/\overline{x}], (p_1''[r_1/y])[\overline{q}_2/\overline{x}]) \in ACR^*$. Thus $(p_1'[r'/x], p_1''[r''/z]) \in ACR^*$. Therefore there exist p_3, $\overline{q}_1^1$, $\overline{q}_2^1$ and $\overline{x}^1$ such that $p_1'[r'/x] \equiv p_3[\overline{q}_1^1/\overline{x}^1]$ and $p_1''[r''/z] \equiv p_3[\overline{q}_2^1/\overline{x}^1]$ with $FV(p_3) \subseteq \overline{x}^1$ and $\overline{q}_1^1 \dot{\sim} \overline{q}_2^1$. Also, since $(p_2', p_2'') \in ACR^*$ there exist p_4, $\overline{q}_1^2$, $\overline{q}_2^2$ and $\overline{x}^2$ such that $p_2' \equiv p_4[\overline{q}_1^2/\overline{x}^2]$ and $p_2'' \equiv p_4[\overline{q}_2^2/\overline{x}^2]$ with $FV(p_4) = \overline{x}^2$ and $\overline{q}_1^2 \dot{\sim} \overline{q}_2^2$. We may assume $\overline{x}^1 \cap \overline{x}^2 = \emptyset$ since if $\overline{x}^1 \cap \overline{x}^2 \neq \emptyset$ we proceed by choosing $\overline{y}$ such that $\overline{y} \cap (FV(p_3) \cup FV(p_4) \cup \overline{x}^1 \cup \overline{x}^2) = \emptyset$ and we have $p_3[\overline{q}_i^1/\overline{x}^1] \equiv (p_3[\overline{y}/\overline{x}^1])[\overline{q}_i^1/\overline{y}]$ by proposition 5.1.5. Therefore we have $p_1'[r'/x] \mid p_2' \equiv (p_3[\overline{q}_1^1/\overline{x}^1]) \mid (p_4[\overline{q}_1^2/\overline{x}^2]) \equiv (p_3 \mid p_4)[\overline{q}_1^1 \cup \overline{q}_1^2/\overline{x}^1 \cup \overline{x}^2]$ and $p_1''[r'/z] \mid p_2'' \equiv (p_3[\overline{q}_2^1/\overline{x}^1]) \mid (p_4[\overline{q}_2^2/\overline{x}^2]) \equiv (p_3 \mid p_4)[\overline{q}_2^1 \cup \overline{q}_2^2/\overline{x}^1 \cup \overline{x}^2]$ and $(p_1'[r/x] \mid p_2', p_1''[r/z] \mid p_2'') \in ACR^*$. (Note that if we have to introduce a "new" $\overline{y}$ it is because two or more occurrences of the same x_i refer to different q_i's after the transition.)

$\underline{p \equiv p_1\backslash b}$ Then $p[\overline{q}_i/\overline{x}] \equiv (((\{d_i/b\}p_1)[\overline{q}_i/\overline{x}])\backslash d_i$ for some $d_i \notin fn(p_1\backslash b) \cup fn(\overline{q}_i)$. By (i) we may assume $b = d_1 = d_2 \notin fn(p_1\backslash b) \cup fn(\overline{q}_1) \cup fn(\overline{q}_2)$. If

$p[\overline{q}_1/\overline{x}] \xrightarrow{\Gamma} p''$ then if

$\Gamma = a?x$ Then $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a?x} p_1'$ by a shorter inference and $p' \equiv p_1'\backslash b$ and $a \neq b$. By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a?z} p_2'$ $(p_1'[r/x], p_2'[r/z]) \in ACR^*$ for all $r \in CPr$. By the operational semantics for restriction $(p_1\backslash b)[\overline{q}_2/\overline{x}] \xrightarrow{a?z} p_2'\backslash b$. Since $(p_1'[r/x], p_2'[r/z]) \in ACR^*$ for all $r \in CPr$ there exist p_3, $\overline{q}_1$, $\overline{q}_2$ and $\overline{x}$ for each r such that $p_1' \equiv p_3[\overline{q}_1/\overline{x}]$ and $p_2' \equiv p_3[\overline{q}_2/\overline{x}]$ with $FV(p_3) \subseteq \overline{x}$ and $\overline{q}_1 \dot{\sim} \overline{q}_2$ and $(p_1'\backslash b) \equiv (p_3\backslash b)[\overline{q}_1/\overline{x}]$ and $(p_2'\backslash b) \equiv (p_3\backslash b)[\overline{q}_2/\overline{x}]$ thus $(p_1'\backslash b, p_2'\backslash b) \in ACR^*$.

$\Gamma = a!_B p'$ Then

either $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a!_B p_1'} p_1''$ by a shorter inference and $p' \equiv p_1'\backslash b$, $p'' \equiv p_1''\backslash b$, $b \neq a$, $b \notin B$, $b \notin fn(p_1') \cap fn(p_1'')$. Then by induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a!_B p_2'} p_2''$ with $(p_1', p_2') \in ACR^*$ and $(p_1'', p_2'') \in ACR^*$ and $B \cap (fn(p_1[\overline{q}_1/\overline{x}]) \cup fn(p_1[\overline{q}_2/\overline{x}])) = \emptyset$. Then by the res-rule we have $(p_1[\overline{q}_2/\overline{x}])\backslash b \xrightarrow{a!_B p_2'\backslash b} p_2''\backslash b$ and we may argue as above that $(p_1'\backslash b, p_2'\backslash b) \in ACR^*$ and $(p_1''\backslash b, p_2''\backslash b) \in ACR^*$.

or $p_1[\overline{q}_1/\overline{x}] \xrightarrow{a!'_B p_1'} p_1''$ by a shorter inference and $p' \equiv \{d/b\}p_1'$, $p'' \equiv \{d/b\}p_1''$, $b \neq a$, $b \notin B'$, $b \in fn(p_1') \cap fn(p_1'')$, $B = B' \cup \{d\}$, $d \notin B' \cup fn((p_1[\overline{q}_i/\overline{x}])\backslash b)$. Then by induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{a!'_B p_2'} p_2''$ with $(p_1', p_2') \in ACR^*$ and $(p_1'', p_2'') \in ACR^*$ and $B' \cap (fn(p_1[\overline{q}_1/\overline{x}]) \cup fn(p_1[\overline{q}_2/\overline{x}])) = \emptyset$. If $b \in fn(p_2') \cap fn(p_2'')$ then by the open-rule we have $(p_1[\overline{q}_2/\overline{x}])\backslash b \xrightarrow{a!_B \{d/b\}p_2'} \{d/b\}p_2''$ and by (i) we have $(\{d/b\}p_1', \{d/b\}p_2') \in ACR^*$ and $(\{d/b\}p_1'', \{d/b\}p_2'') \in ACR^*$. If $b \notin fn(p_2') \cap fn(p_2'')$ then by the non-struct-rule we have $(p_1[\overline{q}_2/\overline{x}])\backslash b \xrightarrow{a!_B p_2'} p_2''$ and $p_2' \equiv \{d/b\}p_2'$ and $p_2'' \equiv \{d/b\}p_2''$ and by (i) we have $(\{d/b\}p_1', \{d/b\}p_2') \in ACR^*$ and $(\{d/b\}p_1'', \{d/b\}p_2'') \in ACR^*$.

$\Gamma = \tau$ and we may argue as above.

<u>$p \equiv p_1[S]$</u> If $p[\overline{q}_1/\overline{x}] \equiv (p_1[\overline{q}_1/\overline{x}])[S] \xrightarrow{\Gamma} p''$ then if

$\Gamma = a?x$ we have $p_1[\overline{q}_1/\overline{x}] \xrightarrow{b?x} p_1''$ by a shorter inference and $a = S(b)$ and $p'' \equiv p_1''[S]$. By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{b?z} p_2''$ and $p_1''[r/x]ACR^*p_2''[r/z]$ for all $r \in CPr$. Then $(p_1[\overline{q}_1/\overline{x}])[S] \xrightarrow{a?z} p_2''[S]$ with $(p_1''[r/x])[S] \equiv (p_1''[S])[r/x]ACR^*(p_2''[S])[r/z] \equiv (p_2''[r/z])[S]$ for all $r \in CPr$.

$\Gamma = a!_B p'$ we have $p_1[\overline{q}_1/\overline{x}] \xrightarrow{b!_B p_1'} p_1''$ by a shorter inference and $a = S(b)$ and $B \cap (Dom(S) \cup Im(S)) = \emptyset$ and $p' \equiv p_1'$ and $p'' \equiv p_1''[S]$. By induction $p_1[\overline{q}_2/\overline{x}] \xrightarrow{b!_B p_2'} p_2''$ and $p_1'ACR^*p_2'$ and $p_1''ACR^*p_2''$. Then $(p_1[\overline{q}_1/\overline{x}])[S] \xrightarrow{a!_B p_2'} p_2''[S]$ with $p_1'ACR^*p_2'$ and $p_1''[S]ACR^*p_2''[S]$.

$\Gamma = \tau$ this case is similar to the above.

$\underline{p \equiv y}$ By assumption $FV(p) \subseteq \overline{x}$ thus $\overline{x} = (y)$ and if $p[\overline{q}_1/\overline{x}] \equiv q_1 \xrightarrow{\Gamma} q_1'$ then if

$\Gamma = a?x$ we have $p[\overline{q_2}/\overline{x}] \equiv q_2 \xrightarrow{a?z} q_2'$ for some q_2' and z. Since $q_1 \dot{\sim} q_2$ we have $(q_1'[r/x], q_2'[r/z]) \in \dot{\sim}$ for all $r \in CPr$ and thus $(q_1'[r/x], q_2'[r/z]) \in ACR^*$ for all $r \in CPr$

$\Gamma = a!_B p'$ we have $p[\overline{q_2}/\overline{x}] \equiv q_2 \xrightarrow{a!_B q_2''} q_2'$ for some q_2' and q_2''. Since $q_1 \dot{\sim} q_2$ we have $(q_1', q_2') \in \dot{\sim}$ and $(q_1'', q_2'') \in \dot{\sim}$ and thus $(q_1', q_2') \in ACR^*$ and $(q_1'', q_2'') \in ACR^*$

$\Gamma = \tau$ A similar argument as above applies.

Thus in each case we have a matching move for $p[\overline{q}_2/\overline{x}]$.

2. This is proved by showing that the relation $R_1 = R \cup \dot{\sim}$, where:

$$R = \{(a?x.p, a?x.q) \ : \ FV(p) = FV(q) \subseteq \{x\} \& \forall r \in CPr. p[r/x] \dot{\sim} q[r/x]\}$$

is an applicative higher order bisimulation Note that the relation R_1 consists of two parts; one part covers the structure we are interested in and the second component is a kind of closure to cover the processes sent and received. The second component is necessary since the processes sent and received do not necessarily have the structure of the first part.
That the above relation is indeed an applicative higher order bisimulation is easily established.

Assume $(p, q) \in R_1$. Then

either $p \dot{\sim} q$ and we are done since if $p \xrightarrow{\Gamma} p'$ then $q \xrightarrow{\Gamma'} q'$ for some q', Γ'. If $\Gamma = a?x$ then $\Gamma' = a?y$ and for all $r \in CPr$ we have $(p'[r/x], q'[r/y]) \in \dot{\sim} \subseteq R_1$. If $\Gamma = a!_B p''$ then $\Gamma' = a!_B q''$ and we have $B \cap (fn(p) \cup fn(q)) = \emptyset$ and $(p'', q'') \dot{\sim} \subseteq R_1$ and $(p', q') \dot{\sim} \subseteq R_1$. If $\Gamma = \tau$ then $\Gamma' = \tau$ and we have $(p', q') \dot{\sim} \subseteq R_1$.

or $p \equiv a?x.p'$ and $q \equiv a?x.q'$. If $a?x.p' \xrightarrow{\Gamma} p'$ then $\Gamma = a?x$. Then $a?x.q' \xrightarrow{a?x} q'$ and by assumption $p'[r/x] \dot{\sim} q'[r/x]$ for all $r \in CPr$ which implies $(p'[r/x], q'[r/x]) \in R_1$.

3. follows from $((a!x.y)[(p, p')/(x, y)], (a!x.y)[(q, q')/(x, y)]) \in ACR$ if $p \dot{\sim} q$ and $p' \dot{\sim} q'$ and $x \neq y$.

4. follows from $((\tau.x)[p/x], (\tau.x)[q/x]) \in ACR$ if $p \dot{\sim} q$.

5. follows from $((x + y)[(p, p')/(x, y)], (x + y)[(q, q')/(x, y)]) \in ACR$ if $p \dot{\sim} q$ and $p' \dot{\sim} q'$ and $x \neq y$.

6. follows from $((x \mid y)[(p,p')/(x,y)], (x \mid y)[(q,q')/(x,y)]) \in ACR$ if $p \dot{\sim} q$ and $p' \dot{\sim} q'$ and $x \neq y$.
7. follows from $(x[p/x], x[q/x]) \in ACR$ if $p \dot{\sim} q$ and the fact that ACR^* is an applicative bisimulation up to restriction.
8. follows from $((x[S])[p/x], (x[S])[q/x]) \in ACR$ if $p \dot{\sim} q$.

□

The congruence result easily generalizes to open terms by standard techniques by defining $p \dot{\sim} q$ iff $\forall r_1 \ldots r_n . p[r_1 \ldots r_n / x_1 \ldots x_n] \dot{\sim} q[r_1 \ldots r_n / x_1 \ldots x_n]$ where $x_1 \ldots x_n$ are the free variables of p and q and $r_1 \ldots r_n$ are closed terms. This is equivalent to the following definition: $p \dot{\sim} q$ iff $a?x_1 \ldots . a?x_n . p \dot{\sim} a?x_1 \ldots . a?x_n . q$.

5.3 Algebraic Laws

From establishing bisimulations between Plain CHOCS processes we may show that two processes are equivalent, but this technique often involves quite an amount of ingenuity in the construction of a bisimulation relation. Instead we may prefer the more well known techniques of algebraic reasoning. A lot of interesting properties of Plain CHOCS may be inferred from equational reasoning. This kind of reasoning may of course be combined with establishing bisimulations directly.

The first set of laws concerns the choice operator and shows that *nil* is a zero for + and that + is idempotent, commutative and associative.

Proposition 5.3.1

$$\begin{aligned} p + nil &\dot{\sim} p \\ p + p &\dot{\sim} p \\ p + p' &\dot{\sim} p' + p \\ p + (p' + p'') &\dot{\sim} (p + p') + p'' \end{aligned}$$

PROOF: This follows from showing that the following relations are higher order applicative bisimulations:

$$\begin{aligned} R_1 &= \{(p + nil, p)\} \cup Id \\ R_2 &= \{(p + p, p)\} \cup Id \\ R_3 &= \{(p + p', p' + p)\} \cup Id \\ R_4 &= \{(p + (p' + p''), (p + p') + p'')\} \cup Id \end{aligned}$$

To see this observe that for $(r,q) \in R_i, i \in \{1,2,3,4\}$ we have either $(r,q) \in Id$ and if $r \xrightarrow{\Gamma} r'$ then $r = q \xrightarrow{\Gamma} q' = r'$ and we have a matching move or (r,q) belongs to the first part of R_i and if $r \xrightarrow{\Gamma} r'$ then this must have been inferred by the rules for choice. Then also $q \xrightarrow{\Gamma} r'$ which is a matching move. □

We now proceed with some properties of the restriction operator and its interplay with the other operators. To smooth the presentation of equations we introduce a fourth (derived) prefix; an output prefix with scope extrusion: $a!_B p'$. Thus $a!_B p'.p$ is shorthand notation for $(a!p'.p)\backslash B$ with the obvious operational semantics: $a!_B p'.p \xrightarrow{a!_B p'} p$. We shall always assume that $B \subseteq fn(p') \cap fn(p)$.

Proposition 5.3.2

$$
\begin{aligned}
p\backslash a &\dot{\sim} p \ \ if\ a \notin fn(p)\\
p\backslash a\backslash b &\dot{\sim} p\backslash b\backslash a\\
(p+p')\backslash a &\dot{\sim} p\backslash a + p'\backslash a\\
(a?x.p)\backslash b &\dot{\sim} a?x.(p\backslash b) \ \ if\ a \neq b\\
(a?x.p)\backslash b &\dot{\sim} nil \ \ if\ a = b\\
(\tau.p)\backslash b &\dot{\sim} \tau.(p\backslash b)\\
(a!_B p'.p)\backslash b &\dot{\sim} a!_B(p'\backslash b).(p\backslash b) \ \ if\ a \neq b\ and\ b \notin fn(p') \cap fn(p)\\
(a!_B p'.p)\backslash b &\dot{\sim} a!_{B\cup\{b\}} p'.p \ \ if\ a \neq b\ and\ b \in fn(p') \cap fn(p)\\
(a!_B p'.p)\backslash b &\dot{\sim} nil \ \ if\ a = b
\end{aligned}
$$

PROOF: The proposition follows from showing that the following relations are applicative higher order bisimulations:

$$
\begin{aligned}
R_1 &= \{(p\backslash a, p) \ : \ p \in CPr, a \notin fn(p)\}\\
R_2 &= \{(p\backslash a\backslash b, p\backslash b\backslash a) \ : \ p \in CPr\} \cup Id\\
R_3 &= \{((p+p')\backslash a, p\backslash a + p'\backslash a) \ : \ p_i \in CPr\} \cup Id\\
R_4 &= \{((a?x.p)\backslash b, a?x.(p\backslash b)) \ : \ a?x.p \in CPr, a \neq b\} \cup Id\\
R_5 &= \{((a?x.p)\backslash b, nil) \ : \ a?x.p \in CPr, a = b\}\\
R_6 &= \{((\tau.p)\backslash b, \tau.(p\backslash b)) \ : \ p \in CPr\} \cup Id\\
R_7 &= \{((a!_B p'.p)\backslash b, a!_B(p'\backslash b).(p\backslash b)) \ : \ p, p' \in CPr, a \neq b, b \notin fn(p') \cap fn(p)\} \cup Id\\
R_8 &= \{((a!_B p'.p)\backslash b, a!_{B\cup\{b\}} p'.p) \ : \ p, p' \in CPr, a \neq b, b \in fn(p') \cap fn(p)\} \cup Id\\
R_9 &= \{((a!_B p'.p)\backslash b, nil) \ : \ p, p' \in CPr, a = b\}
\end{aligned}
$$

We must include Id in relation R_2 to R_4 and R_6 to R_8. For relation R_3, R_4 and R_6 to R_8 this is clear since if $(p,q) \in R_i, i \in \{3,4,6,7,8\}$ then after the first transition $p \xrightarrow{\Gamma} p'$ and a first matching transition $q \xrightarrow{\Gamma'} q'$ we will have $(p',q') \in Id$. For R_2 it is necessary to include Id since the restrictions may disappear due to applications of the open-rule. □

The following theorem states an expected property of restriction, namely that the restricted name may be α-converted without affecting the behaviour of the process involved.

Theorem 5.3.3 $p\backslash a \dot{\sim} (\{b/a\}p)\backslash b$ *if* $b \notin fn(p)$

PROOF: This theorem follows by showing that the relation

$$R = \{(p\backslash a, (\{b/a\}p)\backslash b) \ : \ p \in CPr, b \notin fn(p)\} \cup Id$$

is an applicative higher order bisimulation.
The Id component of this relation is necessary in case of scope extrusion due to an application of the open-rule in which case the restrictions will disappear and a respectively b will be substituted with a new name $c \notin fn(p) \cup \{b\}$. The matching moves are easily established by appealing to proposition 5.1.7. □

Before presenting any additional laws we need to introduce a concept related to the concept of an applicative higher order bisimulation up to restriction. The new concept is called an applicative higher order bisimulation up to $\dot{\sim}$ and allows a relaxation of applicative higher order bisimulation in the sense that the relation only has to satisfy the applicative higher order bisimulation properties up to the closure property of $\dot{\sim}$:

Definition 5.3.4 *An* applicative higher order simulation up to $\dot{\sim}$ *is a binary relation R on CPr such that whenever pRq and $a \in Names$ then:*

(i) Whenever $p \xrightarrow{a?x} p'$, *then* $q \xrightarrow{a?y} q'$ *for some* q', y *and* $p'[r/x] \dot{\sim} R \dot{\sim} q'[r/y]$ *for all* $r \in CPr$

(ii) Whenever $p \xrightarrow{a!_B p'} p''$, *then* $q \xrightarrow{a!_B q'} q''$ *for some* q', q'' *with* $B \cap (fn(p) \cup fn(q)) = \emptyset$ *and* $p' \dot{\sim} R \dot{\sim} q'$ *and* $p'' \dot{\sim} R \dot{\sim} q''$

(iii) Whenever $p \xrightarrow{\tau} p'$, *then* $q \xrightarrow{\tau} q'$ *for some* q' *with* $p' \dot{\sim} R \dot{\sim} q'$

A relation R is an applicative higher order bisimulation up to $\dot{\sim}$ *if both it and its inverse are applicative higher order simulations up to* $\dot{\sim}$.

Lemma 5.3.5 *If R is an applicative higher order bisimulation up to $\dot{\sim}$ then $R \subseteq \dot{\sim}$.*

PROOF: Follows by arguments very similar to the arguments given for lemma 5.2.5 □

Definition 5.3.6 *An* applicative higher order simulation up to $\dot{\sim}$ and restriction *R is a binary relation on CPr such that whenever pRq and $a \in Names$ then:*

(i) If $b \notin fn(p) \cup fn(q)$ then $\{b/a\}p \dot{\sim} R \dot{\sim} \{b/a\}q$

(ii) Whenever $p \xrightarrow{a?x} p'$, then $q \xrightarrow{a?y} q'$ for some q', y and $p'[r/x] \dot{\sim} R \dot{\sim} q'[r/y]$ for all $r \in CPr$

(iii) Whenever $p \xrightarrow{a!_B p'} p''$, then $q \xrightarrow{a_B!q'} q''$ for some q', q'' with $B \cap (fn(p) \cup fn(q)) = \emptyset$, $p' \dot{\sim} R \dot{\sim} q'$ and $p'' \dot{\sim} R \dot{\sim} q''$

(iv) Whenever $p \xrightarrow{\tau} p'$, then $q \xrightarrow{\tau} q'$ for some q' and either $p' \dot{\sim} R \dot{\sim} q'$ or for some p'', q'' and b: $p' \dot{\sim} p''\backslash b$, $q' \dot{\sim} q''\backslash b$ and $p''Rq''$

A relation R is an applicative higher order bisimulation up to $\dot{\sim}$ and restriction *if both it and its inverse are applicative higher order simulations up to restriction.*

Lemma 5.3.7 *If R is an applicative higher order bisimulation up to $\dot{\sim}$ and restriction then $R \subseteq \dot{\sim}$.*

PROOF: Let $R^{\backslash\dot{\sim}} = \bigcup_{n\in\omega} R_n$ where

$$\begin{aligned} R_0 &= \dot{\sim} R \dot{\sim} \\ R_{n+1} &= \dot{\sim} \{(p\backslash a, q\backslash a) : (p,q) \in R_n, a \in Names\} \dot{\sim} \end{aligned}$$

The argument that $R^{\backslash\dot{\sim}}$ is an applicative higher order bisimulation follows the same pattern as the proof of lemma 5.2.5 □

With this machinery in hand we may now prove the following interplay between the restriction operator and parallel composition:

Proposition 5.3.8 $p_1\backslash a \mid p_2 \dot{\sim} (p_1 \mid p_2)\backslash a$ *if* $a \notin fn(p_2)$

PROOF: This proposition is proved by showing that the relation

$$R = \{(p_1\backslash a \mid p_2, (p_1 \mid p_2)\backslash a) \; : \; p_i \in CPr, a \notin fn(p_2)\} \cup Id$$

is an applicative higher order bisimulation up to $\dot{\sim}$ and restriction. To see this we show that when $(p,q) \in R$ and $p \xrightarrow{\Gamma} p'$ then $q \xrightarrow{\Gamma'} q'$ with a move which satisfies the conditions of applicative higher order bisimulation up to $\dot{\sim}$ and restriction. If $(p,q) \in Id$ the case is obvious so assume that $p \equiv p_1\backslash a \mid p_2$ and $q \equiv (p_1 \mid p_2)\backslash a$ and $a \notin fn(p_2)$.
If $p \xrightarrow{\Gamma} p'$ this transition must have been inferred in the following way:

either this has been inferred from the par-rule and $p_2 \xrightarrow{\Gamma} p_2''$ and $p' \equiv p_1\backslash a \mid p_2''$. There are three cases:

- $\Gamma = b?x$ then $b \neq a$ since $a \notin fn(p_2)$. Then by the par-rule and the res-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{b?x} (p_1 \mid p_2'')\backslash a$ and for all $r \in CPr$ we have $(p_1\backslash a \mid p_2'')[r/x] \dot{\sim} (\{d/a\}p_1)\backslash b \mid p_2''[r/x] R (\{d/a\}p_1 \mid p_2''[r/x])\backslash d \dot{\sim} ((p_1 \mid p_2'')\backslash a)[r/x]$ for some $d \notin fn(p_1) \cup fn(p_2) \cup fn(r)$.
- $\Gamma = b!_B p_2'$ then $b \neq a$ and we may assume $B \cap (\{a\} \cup fn(p_1)) = \emptyset$. Then by the par-rule and the res-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{b!_B p_2'\backslash a} (p_1 \mid p_2'')\backslash a$ which is a matching move since $a \notin fn(p_2) \cup B$ and therefore $p_2'\backslash a \dot{\sim} p_2'$.
- $\Gamma = \tau$ and we may argue as in the above case.

or this transition has been inferred by the par-rule and $p_1\backslash a \xrightarrow{\Gamma} p_1''$ and this has been inferred from the res-rule and $p_1 \xrightarrow{\Gamma} p_1'''$ and $p' \equiv p_1'' \mid p_2$. There are three cases:

- $\Gamma = b?x$ then $p_1'' \equiv p_1'''\backslash a$ and $b \neq a$. Then by the par-rule and the res-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{b?x} (p_1''' \mid p_2)\backslash a$. This is a matching move since for all $r \in CPr$ we have $(p_1'''\backslash a \mid p_2)[r/x] \dot{\sim} (\{d/a\}p_1''')[r/x])\backslash b \mid p_2 R (\{d/a\}p_1''')[r/x] \mid p_2)\backslash d \dot{\sim} ((p_1''' \mid p_2)\backslash a)[r/x]$ for some $d \notin fn(p_1) \cup fn(p_2) \cup fn(r)$.
- $\Gamma = b!_B p_1'$ then $p_1 \xrightarrow{b!_B p_1''''} p_1'''$ and $b \neq a$ and
  - **either** $a \notin fn(p_1'''') \cap fn(p_1''')$ in which case $p_1' \equiv p_1''''\backslash a$ and $p_1'' \equiv p_1'''\backslash a$. Then by the par-rule and the res-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{b!_B p_1'} (p_1''' \mid p_2)\backslash a$ which is a matching move.

or $a \in (fn(p_1'''') \cap fn(p_1''')) \setminus B$ in which case $p_1' \equiv p_1''''$ and $p_1'' \equiv p_1'''$ and $B = B' \cup \{a\}$ for some B' with $a \notin B'$. We may assume $B' \cap fn(p_2) = \emptyset$. Then by the par-rule and the open-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{b!_{Bp_1'}} p_1''' \mid p_2$ which is a matching move.

$\Gamma = \tau$ and we may argue as in the above case.

or $\Gamma = \tau$ and the transition has been inferred by the com-close-rule and $p_1\backslash a \xrightarrow{b?x} p_1'\backslash a$ which has been inferred by the res-rule and $p_1 \xrightarrow{b?x} p_1'$ with $b \neq a$ and $p_2 \xrightarrow{b!_{Bp_2'}} p_2''$ and $p' \equiv ((p_1'\backslash a)[p_2'/x] \mid p_2'')\backslash B$. We may assume $B \cup fn(p_1) = \emptyset$ and $a \notin fn(p_2')$. Thus by the com-close-rule and the res-rule we may infer that $(p_1 \mid p_2)\backslash a \xrightarrow{\tau} (p_1'[p_2'/x] \mid p_2'')\backslash B\backslash a$ which is a matching move since $(p_1'[p_2'/x] \mid p_2'')\backslash B\backslash a \dot{\sim} (p_1'[p_2'/x] \mid p_2'')\backslash a\backslash B$ by proposition 5.3.2 and $((p_1'\backslash a)[p_2'/x] \mid p_2'')R(p_1'[p_2'/x] \mid p_2'')\backslash a$.

or $\Gamma = \tau$ and the transition has been inferred by the com-close-rule and $p_1\backslash a \xrightarrow{b!_{Bp_1'}} p_1''$ which has been inferred by the res-rule and $p_1 \xrightarrow{b!_{Bp_1''''}} p_1'''$ and $b \neq a$ and $a \notin fn(p_1'''') \cap fn(p_1''')$ and $p_1' \equiv p_1''''\backslash a$ and $p_1'' \equiv p_1'''\backslash a$ and $p_2 \xrightarrow{b?x} p_2''$ and $p' \equiv (p_1'' \mid p_2''[p_1'/x])\backslash B$. We assume $B \cap fn(p_2'') = \emptyset$. Then by the com-close-rule and the res-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{\tau} (p_1''' \mid p_2''[p_1''''/x])\backslash B\backslash a$ which is a matching move since $(p_1''' \mid p_2''[p_1''''/x])\backslash B\backslash a \dot{\sim} (p_1''' \mid p_2''[(p_1'''')\backslash a/x])\backslash a\backslash B$ by proposition 5.3.2 and proposition 5.2.6 (and an argument by structural induction on p_2'' which is straightforward since by the assumptions either $p_1''''\backslash a \dot{\sim} p_1'''$ or $p_1''''\backslash a \dot{\sim} p_1''''$) and $p_1'' \mid p_2''[p_1'/x]R(p_1''' \mid p_2''[(p_1'''')\backslash a/x])\backslash a$.

or $\Gamma = \tau$ and the transition has been inferred by the com-close-rule and $p_1\backslash a \xrightarrow{b!_{Bp_1'}} p_1''$ which has been inferred by the open-rule and $p_1 \xrightarrow{b!_{B'p_1''''}} p_1'''$ and $b \neq a$ and $a \in fn(p_1'''') \cap fn(p_1''')$ and $p_1' \equiv p_1''''$ and $p_1'' \equiv p_1'''$ and $B = B' \cup \{a\}$ for some B' with $a \notin B'$ and $p_2 \xrightarrow{b?x} p_2''$ and $p' \equiv (p_1'' \mid p_2''[p_1'/x])\backslash B$. We assume $B' \cap fn(p_2'') = \emptyset$. Then by the com-close-rule and the res-rule we have $(p_1 \mid p_2)\backslash a \xrightarrow{\tau} (p_1''' \mid p_2''[p_1''''/x])\backslash B'\backslash a$ which is a matching move since $(p_1''' \mid p_2''[p_1''''/x])\backslash B'\backslash a \dot{\sim} (p_1''' \mid p_2''[p_1''''/x])\backslash B$ by proposition 5.3.2 and $p_1'' \mid p_2''[p_1'/x]Rp_1''' \mid p_2''[p_1''''/x]$.

We omit the proof for the cases showing R^{-1} is an applicative higher order simulation up to $\dot{\sim}$ and restriction. The arguments in these cases are very similar to the above and follow almost from symmetry. □

The next set of laws shows some expected properties of the parallel operator. It would perhaps have been more natural to present these laws before the laws of restriction and its interplay with other operators, but to prove the law of associativity for the parallel operator we need some of the above properties.

Proposition 5.3.9

$$\begin{aligned} p \mid nil &\dot{\sim} p \\ p_1 \mid p_2 &\dot{\sim} p_2 \mid p_1 \\ p_1 \mid (p_2 \mid p_3) &\dot{\sim} (p_1 \mid p_2) \mid p_3 \end{aligned}$$

PROOF: This proposition is proved by showing that the first two of the following relations are applicative higher order bisimulations and that the last relation is an applicative higher order bisimulation up to $\dot{\sim}$ and restriction:

$$\begin{aligned} R_1 &= \{(p \mid nil, p) : p \in CPr\} \cup Id \\ R_2 &= \{(p_1 \mid p_2, p_2 \mid p_1) : p_i \in CPr\} \cup Id \\ R_3 &= \{(p_1 \mid (p_2 \mid p_3), (p_1 \mid p_2) \mid p_3) : p_i \in CPr\} \cup Id \end{aligned}$$

The *Id* component in each of the above relations is necessary to cover the cases when processes are communicated since these processes might not have the structure of the first part of the relation. To see that the above relations are indeed applicative higher order bisimulations respectively applicative higher order bisimulations up to $\dot{\sim}$ and restriction we analyze each relation in turn. (The *Id* part of the above relations is obvious.)

R_1 Any transitions of $p \mid nil$ must have been inferred from a transition of p and the rule for parallel composition since *nil* has no transitions, thus p has a matching move for each move of $p \mid nil$ and vice versa.

R_2 This is easily established by noting that both rules (par and com-close) involving the parallel operator are symmetric.

R_3 The proof that this relation is an applicative higher order bisimulation up to $\dot{\sim}$ and restriction is surprisingly complicated. This is due to the fact that the communication of processes may introduce restrictions and thus alter the structure of the term. To illustrate this point we show the case when $p_1 \mid (p_2 \mid p_3) \xrightarrow{\tau} p'$ and this transition has been inferred by the com-close-rule and $p_1 \xrightarrow{b?x} p_1'$ and $(p_2 \mid p_3) \xrightarrow{b!_B p''} p'''$ and this is due to an application of the par-rule and $p_2 \xrightarrow{b!_B p''} p_2'$ with $B \cap fn(p_3) = \emptyset$ and $p''' \equiv p_2' \mid p_3$ and $p' \equiv (p_1'[p''/x] \mid (p_2' \mid p_3))\backslash B$. Then by the com-close-rule $p_1 \mid p_2 \xrightarrow{\tau} (p_1'[p''/x] \mid p_2')\backslash B$ and by the par-rule $(p_1 \mid p_2) \mid p_3 \xrightarrow{\tau} (p_1'[p''/x] \mid p_2')\backslash B \mid p_3$. Since $B \cap fn(p_3)$ we can apply proposition 5.3.2 and $(p_1'[p''/x] \mid p_2')\backslash B \mid p_3 \dot{\sim} ((p_1'[p''/x] \mid p_2') \mid p_3)\backslash B$

and we have established a matching move which satisfies the conditions of an applicative higher order bisimulation up to $\dot{\sim}$ and restriction. There are five other similar cases: one when p_1 does an output transition and p_2 does an input transition, two when p_1 and p_3 communicate and two when p_2 and p_3 communicate. These cases follow the same pattern of argument as above. The only three remaining cases are when either of the three components does a transition on its own but in each case a matching move can be established by two applications of the par-rule.

□

Using the above properties we may now present a law of interplay between parallel composition and restriction which will look more familiar to readers with knowledge of CCS.

Theorem 5.3.10 $(p_1 \mid p_2)\backslash a \dot{\sim} p_1\backslash a \mid p_2\backslash a$ *if* $a \notin fn(p_1) \cap fn(p_2)$

PROOF: If $a \notin fn(p_1) \cap fn(p_2)$ then a can not be a free name in both p_1 and p_2. Suppose $a \notin fn(p_2)$. Then by proposition 5.3.8 and proposition 5.3.2 we have $(p_1 \mid p_2)\backslash a \dot{\sim} p_1\backslash a \mid p_2 \dot{\sim} p_1\backslash a \mid p_2\backslash a$. The other case where $a \notin fn(p_1)$ follows by a similar argument after commuting p_1 and p_2 using proposition 5.3.9. □

We now present some expected properties of renaming:

Proposition 5.3.11

$$\begin{aligned}
nil[S] &\dot{\sim} nil \\
p[S] &\dot{\sim} p[S][S] \\
p[S]\backslash b &\dot{\sim} p\backslash b[S] \quad \text{if } b \notin Dom(S) \cup Im(S) \\
(p_1 + p_2)[S] &\dot{\sim} p_1[S] + p_2[S] \\
(p_1 \mid p_2)[S] &\dot{\sim} p_1[S] \mid p_2[S] \\
(a?x.p)[S] &\dot{\sim} S(a)?x.(p[S]) \\
(\tau.p)[S] &\dot{\sim} \tau.(p[S]) \\
(a!_B p'.p)[S] &\dot{\sim} S(a)!_B p'.(p[S]) \quad \text{if } B \cap (Dom(S) \cup Im(S)) = \emptyset
\end{aligned}$$

PROOF: The proposition follows from showing that the following relations are applicative higher order bisimulations:

$$\begin{aligned}
R_1 &= \{(nil[S], nil)\} \\
R_2 &= \{(p[S], p[S][S]) \,:\, p \in CPr\}
\end{aligned}$$

$$
\begin{aligned}
R_3 &= \{(p[S]\backslash b, p\backslash b[S] \ : \ p \in CPr, b \notin Dom(S) \cup Im(S)\} \cup Id \\
R_4 &= \{((p_1 + p_2)[S], p_1[S] + p_2[S]) \ : \ p_i \in CPr\} \cup Id \\
R_5 &= \{((p_1 \mid p_2)[S], p_1[S] \mid p_2[S]) \ : \ p_i \in CPr\} \cup Id \\
R_6 &= \{((a?x.p)[S], S(a)?x.(p[S])) \ : \ a?x.p \in CPr\} \cup Id \\
R_7 &= \{((\tau.p)[S], \tau.(p[S])) \ : \ p_i \in CPr\} \cup Id \\
R_8 &= \{((a!_B p'.p)[S], S(a)!_B p'.(p[S])) \ : \\
&\qquad p, p' \in CPr, B \cap (Dom(S) \cup Im(S)) = \emptyset\} \cup Id
\end{aligned}
$$

The Id component in relations R_3 to R_8 serves to cover processes being sent. In addition the Id component of relation R_3 covers the case when the restriction disappears due to an application of the open-rule. It is relatively straightforward to find matching moves for each relation and we omit the details. (The proof for relation R_5 relies on the fact that $p[S] \dot{\sim} p[S][S]$ and this is easily established since $S = [a \mapsto b]$ and either $a = b$ in which case $p[S] \dot{\sim} p$ or $a \neq b$ in which case the second renaming will have no effect.) □

We have not listed any immediate interplay between (nondeterministic) choice and parallel composition. This is due to the fact that the two operators in general do not commute, but there is a restricted interplay between them:

Proposition 5.3.12 *Let* $\overline{x} = \{x_1 \ldots x_n\}$, $\overline{y} = \{y_1 \ldots y_n\}$ *and* $\overline{x} \cap \overline{y} \neq \emptyset$ *and* $A_j \cap fn(q) = \emptyset$ *and* $B_l \cap fn(p) = \emptyset$ *then*

$$
\begin{array}{rrcl}
\textit{if} & p & = & \Sigma_i a_i?x_i.p_i + \Sigma_j a_j!_{A_j} p'_j.p_j \\
\textit{and} & q & = & \Sigma_k b_k?y_k.q_k + \Sigma_l b_l!_{B_l} q'_l.q_l \\
\textit{then} & p \mid q & \dot{\sim} & \Sigma_i a_i?x_i.(p_i \mid q) + \Sigma_j a_j!_{A_j} p'_j.(p_j \mid q) + \\
 & & & \Sigma_k b_k?y_k.(p \mid q_k) + \Sigma_l b_l!_{B_l} q'_l.(p \mid q_l) + \\
 & & & \Sigma_{(i,l) \in \{(i,l) \,:\, a_i = b_l\}} \tau.(p_i[q'_l/x_i] \mid q_l)\backslash B_l + \\
 & & & \Sigma_{(j,k) \in \{(j,k) \,:\, a_j = b_k\}} \tau.(p_j \mid q_k[p'_j/y_k])\backslash A_j
\end{array}
$$

where $\Sigma_i \Gamma_i.p_i$ *describes the sum* $\Gamma_1.p_1 + \ldots + \Gamma_n.p_n$ *when* $n > 0$ *and nil if* $n = 0$, *knowing this notation is unambiguous because of proposition 5.3.1.*

PROOF: Assume the premisses of the proposition. Let rhs denote the right hand side of the above equation. Let

$$R = \{(p \mid q, rhs)\} \cup Id$$

Then R is an applicative higher order bisimulation. For each transition of $p \mid q$ we may find a matching transition of rhs and vice versa.
If $p \mid q \xrightarrow{\Gamma} r$ then

either $p \xrightarrow{\Gamma} p'$ and $r \equiv p' \mid q$. If $\Gamma = a_i?x_i$ then $p' \equiv p_i$ for some i and $rhs \xrightarrow{\Gamma} p_i \mid q$ which is a matching move since $x_i \notin FV(q)$.
If $\Gamma = a_j!_{A_j}p'_j$ then $p' \equiv p_j$ for some j and $rhs \xrightarrow{\Gamma} p_j \mid q$ which is a matching move.

or $q \xrightarrow{\Gamma} q'$ and $r \equiv p \mid q'$. Then similar arguments as above apply.

or $\Gamma = \tau$. Then

either $p \xrightarrow{a_i?x_i} p_i$ and $q \xrightarrow{b_l!_{B_l}q'_l} q_l$ and $r \equiv (p_i[q'_l/x_i] \mid q_l)\backslash B_l$ and $a_i = b_l$. Then $rhs \xrightarrow{\tau} r$ which is a matching move.

or $q \xrightarrow{b_k?x_k} q_k$ and $p \xrightarrow{a_j!_{A_j}p'_j} p_j$ and $r \equiv (p_j \mid q_k[p'_j/y_k] \mid q_k)\backslash A_j$ and $a_j = b_k$. Again $rhs \xrightarrow{\tau} r$ which is a matching move.

If $rhs \xrightarrow{\Gamma} r$ then a similar case analysis as above will yield matching moves for $p \mid q$. □

We can not hope that these equations form a basis for a sound and complete proof system for Plain CHOCS. One reason for this is hinted in the translation given in the next section from Plain CHOCS into Mobile Processes [MilParWal89]. This translation needs parallel composition under the scope of recursion to work. In [Mil83] Milner shows how this combination could be used to simulate a Turing machine. Another reason is that we may encode recursion using the constructs of Plain CHOCS. In fact the protocol we use is the one defined in [Tho89]:

Definition 5.3.13 *Let $Y_x[\,]$ be the following context:*

$$(a?x.([\,] \mid a!x.nil) \mid a!(a?x.([\,] \mid a!x.nil)).nil)\backslash a$$

To see how this construction works consider the following example also presented in [Tho89]:

Example 5.3.14 *Let $p \equiv b!.x$ then according to the inference rules of definition 5.1.6 $Y_x[p]$ has the following derivations:*

$$\begin{array}{c} Y_x[p] \equiv (a?x.(b!.x \mid a!x.nil) \mid a!(a?x.(b!.x \mid a!x.nil)).nil)\backslash a \\ \downarrow_\tau \\ (b!.(a?x.(b!.x \mid a!x.nil)) \mid a!(a?x.(b!.x \mid a!x.nil)).nil \mid nil)\backslash a \\ \downarrow_{b!} \\ (a?x.(b!.x \mid a!x.nil) \mid a!(a?x.(b!.x \mid a!x.nil)).nil \mid nil)\backslash a \\ \downarrow_\tau \\ (b!.(a?x.(b!.x \mid a!x.nil) \mid a!(a?x.(b!.x \mid a!x.nil)).nil) \mid nil \mid nil \mid nil)\backslash a \end{array}$$

$$\downarrow_{b!}$$
$$\vdots$$

Note how $Y_x[\ ]$ needs a τ–transition to unwind the "recursion". This resembles the unwinding of recursion in the inference rule of recursion in TCCS [HenNic87]: $\mathtt{rec}\,x.p \rightsquigarrow p[\mathtt{rec}\,x.p/x]$, where $\rightsquigarrow$ may be read as $\xrightarrow{\tau}$.

As mentioned in section 2.6 this protocol only simulates recursion in "dynamic" CHOCS when x is not free in a sending position. But because of the static nature of the restriction operator in Plain CHOCS we may use the above construct to program systems which recursively send out copies of themselves:

Let $p \equiv b!x.x$ then according to the inference rules of definition 5.1.6 $Y_x[p]$ has the following derivations:

$$Y_x[p] \equiv (a?x.(b!x.x \mid a!x.nil) \mid a!(a?x.(b!x.x \mid a!x.nil)).nil)\backslash a$$
$$\downarrow_\tau$$
$$(b!(a?x.(b!x.x \mid a!x.nil)).(a?x.(b!x.x \mid a!x.nil)) \mid a!(a?x.(b!x.x \mid a!x.nil)).nil \mid nil)\backslash a$$
$$\downarrow_{b!_{\{a\}}(a?x.(b!x.x \mid a!x.nil))}$$
$$(a?x.(b!.x \mid a!x.nil) \mid a!(a?x.(b!.x \mid a!x.nil)).nil \mid nil)$$

After this transition we have a scope extrusion on a but when the "copy" $(a?x.(b!x.x \mid a!x.nil))$ is received the com-close-rule will ensure that this "copy" can communicate with $a!(a?x.(b!.x \mid a!x.nil)).nil$ and thus continue the "recursive unfolding" of p.

As in section 2.6 we may introduce a recursion operator $\mathtt{rec}\,x.p$ with the following operational semantics:

$$\frac{p[\mathtt{rec}\,x.p/x] \xrightarrow{\Gamma} p'}{\mathtt{rec}\,x.p \xrightarrow{\Gamma} p'}$$

$\mathtt{rec}\,x.$ is a variable binder and fn, $\{\ /\ \}$, FV and $[\ /\ ]$ have to be extended to cater for the new operator.

We cannot prove a simulation of recursion theorem for Plain CHOCS as directly as theorem 2.6.2 for "dynamic" CHOCS. This is because when we send out copies of the recursive process we have to do a scope extrusion for a in the Y construct to keep a connection to the remaining part and keep the "recursion" going, whereas the recursion construct does not need to do a scope extrusion and the two terms are incomparable until they are received and we have closed the scope in the Y construct.

However, we conjecture the following relationship:

Conjecture 5.3.15 *If x is not free in a sending position in p and $a \notin fn(p)$ then*

$$Y_x[p] \dot{\sim} \mathtt{rec}\, x.\tau.p$$

It would be interesting to formulate an equivalence theory where the kind of distributed property of a system linked by internal channels such as the above Y construct is taken into account. I imagine that such a theory would have to use the ideas of context dependent bisimulation described by Larsen in [Lar 86].

5.4 Plain CHOCS and Mobile Processes

In this section we compare the approach taken in this thesis of sending processes to that of sending labels as described in [EngNie86, MilParWal89]. We shall not embark on a discussion of which is the best or the correct way of expressing mobility in concurrent systems, since we feel that both approaches have their justifications. This is further strengthened by showing that the calculi may simulate each other.

The description of Plain CHOCS in Mobile Processes uses the capability of changing the interconnection structure of processes describable in Mobile Processes in a very disciplined way. Whenever a process is sent in Plain CHOCS a link to a trigger construct (which provides copies of the process to be sent) is sent in the Mobile Processes translation. To a certain extent this resembles invocations of procedures in conventional programming languages. The triggering of a copy of the process to be sent and the instantiation of its names could correspond to a new activation record for a procedure and instantiations of its parameters.

The description of Mobile Processes in Plain CHOCS is done by passing very small processes around. These small processes are essentially one element buffers which simulate the behaviour of channels.

We briefly review the π–Calculus as presented in [MilParWal89]. This calculus is a description tool for Mobile Processes with link passing as a means for expressing process networks with dynamically changing interconnection structure.

Processes are built from the following range of constructs: The inactive process 0, three types of prefixes; input prefix $x(y)$, output prefix $\overline{x}y$ and τ prefix, (non-deterministic) choice, parallel composition, restriction, match and recursion.

This is summarized by the syntax of the π-Calculus:

$$p ::= 0 \mid x(z).p \mid \overline{x}y.p \mid \tau.p \mid p + p' \mid p \mid p' \mid (y)p \mid [x = y]p \mid \mathtt{rec}\, X.p \mid X$$

Here $X \in Var$ (a set of variables to be bound by the recursion construct). In [MilParWal89] agent identifiers are used to express recursion, but we prefer the equivalent but more explicit recursion construct above.

In the π–Calculus the communicable values are links or rather names of links, thus x, y above belong to the set *Names* of port names. The constructs of input prefix and restriction bind port names in their scope. The set of free names of a process is denoted by $fn(p)$, the set of bound names of a process is denoted by $bn(p)$ and the set of names of a process is $n(p) = bn(p) \cup fn(p)$.

We may substitute one label for another and label substitution in the π–Calculus follows the pattern of label substitution in Plain CHOCS. We have to take care not to bind free names by input prefix or restriction. If the names coincide we do α-conversion:

$$\begin{aligned} \{z'/z\}(x(y).p) &\equiv \{z'/z\}x(y').(\{z'/z\}(\{y'/y\}p)) \text{ where } y' \notin fn((y)p) \cup \{z'\} \\ \{z'/z\}((y)p) &\equiv (y')(\{z'/z\}(\{y'/y\}p)) \text{ where } y' \notin fn((y)p) \cup \{z'\} \end{aligned}$$

Free and bound (recursion) variables are defined as usual and substitution of processes is the usual one taking care of not accidentally binding free names by restriction and free recursion variables by the recursion construct.

The dynamic behaviour of processes is defined in terms of an operational semantics given as a labelled transition system. Processes may evolve by performing actions of the following kind: input actions $x(y)$, free output actions $\overline{a}y$, τ actions and bound output actions $\overline{x}(y)$. Actions are ranged over by α. A name occurring in brackets in an action is said to be a bound name and the set of bound names of an action is denoted by $bn(\alpha)$. $fn(\alpha)$ denotes the set of free names of an action and $n(\alpha)$ denotes the set of all names of an action. $c(\alpha)$ denotes x in $\alpha = x(y)$ and $\alpha = \overline{x}(y)$.

In the following we give the operational semantics for the π-Calculus as presented in [MilParWal89]. Formally the operational semantics is given as the smallest relation $\xrightarrow{\alpha}$ satisfying the following rules:

TAU-ACT: $\tau.p \xrightarrow{\tau} p$

OUTPUT-ACT: $\overline{x}y.p \xrightarrow{\overline{x}y} p$

INPUT-ACT: $x(z).p \xrightarrow{x(w)} \{w/z\}p \quad , w \notin fv((z)p)$

SUM: $$\frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'}$$

$$\text{MATCH:}\quad \frac{p \xrightarrow{\alpha} p'}{[x = x]p \xrightarrow{\alpha} p'}$$

$$\text{REC:}\quad \frac{p[\texttt{rec}\ X.p/X] \xrightarrow{\alpha} p'}{\texttt{rec}\ X.p \xrightarrow{\alpha} p'}$$

$$\text{PAR:}\quad \frac{p \xrightarrow{\alpha} p'}{p \mid q \xrightarrow{\alpha} p' \mid q} \quad , bv(\alpha) \cap fv(q) = \emptyset$$

$$\text{COM:}\quad \frac{p \xrightarrow{\overline{x}y} p' \quad q \xrightarrow{x(z)} q'}{p \mid q \xrightarrow{\tau} p' \mid q'\{y/z\}}$$

$$\text{CLOSE:}\quad \frac{p \xrightarrow{\overline{x}(w)} p' \quad q \xrightarrow{x(w)} q'}{p \mid q \xrightarrow{\tau} (w)(p' \mid q')}$$

$$\text{RES:}\quad \frac{p \xrightarrow{\alpha} p'}{(y)p \xrightarrow{\alpha} (y)p'} \quad , y \notin v(\alpha)$$

$$\text{OPEN:}\quad \frac{p \xrightarrow{\overline{x}y} p'}{(y)p \xrightarrow{\overline{x}w} \{w/y\}p'} \quad , y \neq x, w \notin fv((y)p')$$

Table 5.4.1. Operational semantics for the π-Calculus. Rules involving the binary operators + and | additionally have symmetric forms.

To compare terms in the π–Calculus we use a generalization of the notion of bisimulation called strong ground bisimulation:

Definition 5.4.1 *A* strong ground simulation *R is a binary relation on CPr such that whenever pRq then:*

(i) Whenever $p \xrightarrow{x(y)} p'$ and $y \notin n(p) \cup n(q)$, then $q \xrightarrow{x(y)} q'$ for some q' and $\{w/y\}p'R\{w/y\}q'$ for all $w \in Names$

(ii) Whenever $p \xrightarrow{\overline{x}y} p'$, then $q \xrightarrow{\overline{x}y} q'$ for some q' and $p'Rq'$

(iii) Whenever $p \xrightarrow{\overline{x}(y)} p'$ and $y \notin (n(p) \cup n(q))$, then $q \xrightarrow{\overline{x}(y)} q'$ for some q' with $p'Rq'$

(iv) Whenever $p \xrightarrow{\tau} p'$, then $q \xrightarrow{\tau} q'$ for some q' with $p'Rq'$

A relation R is a strong ground bisimulation *if both it and its inverse are strong ground simulations.*
Two processes p and q are said to be strong ground bisimulation equivalent iff there exists a strong ground bisimulation R containing (p, q). In this case we write $p \dot{\sim} q$.

In [MilParWal89] the relation $\dot{\sim}$ is shown to be an equivalence relation and it has the expected congruence properties with respect to the constructs of the π–Calculus. It also satisfies a set of expected properties:

$$\begin{aligned}
p+0 &\dot{\sim} p \\
p+p &\dot{\sim} p \\
p+q &\dot{\sim} q+p \\
p+(q+r) &\dot{\sim} (p+q)+r \\
(x)p &\dot{\sim} p \text{ if } x \notin fn(p) \\
(x)(y)p &\dot{\sim} (y)(x)p \\
(x)(p+q) &\dot{\sim} (x)p+(x)q \\
(x)\alpha.p &\dot{\sim} \alpha.(x)p \text{ if } x \notin n(\alpha) \\
(x)\alpha.p &\dot{\sim} 0 \text{ if } x = c(\alpha) \\
p \mid 0 &\dot{\sim} p \\
p \mid q &\dot{\sim} q \mid p \\
(x)(p \mid q) &\dot{\sim} (x)p \mid q \text{ if } x \notin fn(q) \\
p \mid (q \mid r) &\dot{\sim} (p \mid q) \mid r
\end{aligned}$$

The relation $\dot{\sim}$ is however not preserved by arbitrary label substitutions. A notion of strong bisimulation equivalence $\sim$ is introduced in [MilParWal89] as $p \sim q$ iff $\{a/b\}p \dot{\sim} \{a/b\}q$ for all label substitutions $\{a/b\}$. We shall not concern ourselves with this relation since the strong ground bisimulation relation suffices for the presentation in this section.

Before turning to translations between the π–Calculus and Plain CHOCS we present a useful construct and show a few facts about this. We shall need communications which carry no parameters. This could be modelled by presupposing a special name ε which is never bound and we write $\overline{x}.p$ in place of $\overline{x}\varepsilon.p$ and $x.p$ in place of $x(y).p$ where y is not free in p.

Definition 5.4.2 *Let*

$$b \Rightarrow p = \mathtt{rec}\, X.b.(p \mid X)$$

where $b \notin n(p)$ *and* $X \notin FV(p)$.

This construction is intended to provide copies of p when triggered by $\overline{b}$ actions e.g:

$$(b)(\overline{b}.nil \mid \overline{b}.nil \mid b \Rightarrow p) \xrightarrow{\tau}\xrightarrow{\tau} (b)(nil \mid nil \mid p \mid p \mid b \Rightarrow p) \dot{\sim} p \mid p$$

This construct satisfies several interesting properties:

Lemma 5.4.3 *if* $p_i \overset{b}{\not\longrightarrow}$, $i \in \{1,2\}$ *and* $b \notin n(q)$ *then*

$$LHS = (b)(p_1 \mid b \Rightarrow q) + (b)(p_2 \mid b \Rightarrow q) \dot{\sim} (b)((p_1 + p_2) \mid b \Rightarrow q) = RHS$$

PROOF: First note that the p_i's are allowed to trigger $b \Rightarrow q$, but we assume that only $b \Rightarrow q \overset{b}{\longrightarrow} q \mid b \Rightarrow q$. We use b as a private name in both summands of LHS and in RHS, this is convenient and obtainable by a suitable α-conversion on the private names.
To prove the lemma we show that the relation:

$$R = \{(LHS, RHS)\} \cup Id$$

is a strong ground bisimulation.
To see this observe that if $LHS \overset{\alpha}{\longrightarrow} r$ then

either $(b)(p_1 \mid b \Rightarrow q) \overset{\alpha}{\longrightarrow} r$ and this is because

> **either** $p_1 \overset{\alpha}{\longrightarrow} p_1'$ with $\alpha \neq \overline{b}$ and $r \equiv (b)(p_1' \mid b \Rightarrow q)$.
> Then $RHS \overset{\alpha}{\longrightarrow} (b)(p_1' \mid b \Rightarrow q)$ which is a matching move.
>
> **or** $p_1 \overset{\overline{b}}{\longrightarrow} p_1'$ with $\alpha = \tau$ and $r \equiv (b)(p_1' \mid q \mid b \Rightarrow q)$.
> Then $RHS \overset{\tau}{\longrightarrow} (b)(p_1' \mid q \mid b \Rightarrow q)$ which is a matching move.

or $(b)(p_2 \mid b \Rightarrow q) \overset{\alpha}{\longrightarrow} r$ and an argument as above applies.

Also if $RHS \overset{\alpha}{\longrightarrow} r$ then

either $p_1 \overset{\alpha'}{\longrightarrow} p_1'$ with

> **either** $\alpha' = \alpha \neq \overline{b}$ and $r \equiv (b)(p_1' \mid b \Rightarrow q)$.
> Then $LHS \overset{\alpha}{\longrightarrow} (b)(p_1' \mid b \Rightarrow q)$ which is a matching move.
>
> **or** $\alpha' = \overline{b}$ and $\alpha = \tau$ and $r \equiv (b)(p_1' \mid q \mid b \Rightarrow q)$.
> Then $LHS \overset{\tau}{\longrightarrow} (b)(p_1' \mid q \mid b \Rightarrow q)$ which is a matching move.

or $p_2 \overset{\alpha'}{\longrightarrow} p_2'$ and an argument as above applies.

We have abused the notation slightly when $\alpha \neq \overline{b}$ in the above proof since we should analyze each case of α: $a(x)$, $\overline{a}b$, $\overline{a}(c)$ or τ. We shall not do so since it is not hard (only elaborate) and each case follows the general pattern. □

Lemma 5.4.4 *if* $p_i' \overset{b}{\not\longrightarrow}$ *for all derivatives* p_i' *of* p_i, $i \in \{1,2\}$ *and* $b \notin fn(q)$ *then*

$$LHS = (b)(p_1 \mid b \Rightarrow q) \mid (b)(p_2 \mid b \Rightarrow q) \dot{\sim} (b)((p_1 \mid p_2) \mid b \Rightarrow q) = RHS$$

PROOF: To prove the lemma we show that the relation:

$$R = \{(LHS, RHS)\}$$

is a strong ground bisimulation.
To see this observe that if $LHS \xrightarrow{\alpha} r$ then

either $(b)(p_1 \mid b \Rightarrow q) \xrightarrow{\alpha} r'$ and $r \equiv r' \mid (b)(p_2 \mid b \Rightarrow q)$ and this is because

either $p_1 \xrightarrow{\alpha} p_1'$ with $\alpha \neq \overline{b}$ and $r' \equiv (b)(p_1' \mid b \Rightarrow q)$.
Then $RHS \xrightarrow{\alpha} (b)(p_1' \mid p_2 \mid b \Rightarrow q)$ which is a matching move.

or $p_1 \xrightarrow{\overline{b}} p_1'$ with $\alpha = \tau$ and $r' \equiv (b)(p_1' \mid q \mid b \Rightarrow q)$.
Then $RHS \xrightarrow{\tau} (b)(p_1' \mid p_2 \mid q \mid b \Rightarrow q)$ which is a matching move.

or $(b)(p_2 \mid b \Rightarrow q) \xrightarrow{\alpha} r'$ and an argument as above applies.

or $p_1 \xrightarrow{\alpha} p_1'$ and $p_2 \xrightarrow{\overline{\alpha}} p_2'$ and $\alpha = \tau$ and $r \equiv (b)(p_1' \mid b \Rightarrow q) \mid (b)(p_2' \mid b \Rightarrow q)$, where $\overline{\alpha}$ is an action with opposite polarity of α [MilParWal89]. Then $RHS \xrightarrow{\tau} (b)(p_1' \mid p_2' \mid b \Rightarrow q)$ which is a matching move.

Also if $RHS \xrightarrow{\alpha} r$ we may argue in a similar way as above. □

Lemma 5.4.5 *if* $p_i' \overset{b}{\not\longrightarrow}$ *for all derivatives* p_i' *of* p_i, $i \in \{1,2\}$ *and* $b \notin fn(q)$ *and* $c \notin fn(p_1) \cup fn(p_2) \cup fn(q)$ *then*

$$LHS = (c \Rightarrow (b)(p_1 \mid b \Rightarrow q)) \mid (b)(p_2 \mid b \Rightarrow q) \dot{\sim} (b)(c \Rightarrow p_1 \mid p_2 \mid b \Rightarrow q) = RHS$$

PROOF: To prove the lemma we show that the relation:

$$R = \{(LHS, RHS)\}$$

is a strong ground bisimulation up to $\dot{\sim}$ (strong ground bisimulation up to $\dot{\sim}$ is defined similarly to the definition of bisimulation up to $\sim$ in [Mil89]).
To see this observe that if $LHS \xrightarrow{\alpha} r$ then

either $\alpha = c$ and $r \equiv (b)(p_1 \mid b \Rightarrow q) \mid (c \Rightarrow (b)(p_1 \mid b \Rightarrow q)) \mid (b)(p_2 \mid b \Rightarrow q) \dot{\sim} (c \Rightarrow (b)(p_1 \mid b \Rightarrow q)) \mid (b)(p_1 \mid p_2 \mid b \Rightarrow q)$ which follows by lemma 5.4.4.
Then $RHS \xrightarrow{c} (b)(p_1 \mid c \Rightarrow p_1 \mid p_2 \mid b \Rightarrow q)$ which is a matching move.

or $\alpha \neq c$ and $(b)(p_2 \mid b \Rightarrow q) \xrightarrow{\alpha} r'$ and $r \equiv c \Rightarrow (b)(p_1, b \Rightarrow q) \mid r'$ and this is because

either $p_2 \xrightarrow{\alpha'} p_2'$ with $\alpha = \alpha' \neq \overline{b}$ and $r' \equiv (b)(p_2' \mid b \Rightarrow q)$. Then $RHS \xrightarrow{\alpha} (b)(c \Rightarrow p_1 \mid p_2' \mid b \Rightarrow q)$ which is a matching move.

or $p_2 \xrightarrow{\overline{b}} p_2'$ with $\alpha = \tau$ and $r' \equiv (b)(p_2 \mid q \mid b \Rightarrow q)$. Then $RHS \xrightarrow{\tau} (b)(c \Rightarrow p_1 \mid p_2 \mid q \mid b \Rightarrow q)$ which is a matching move.

Also if $RHS \xrightarrow{\alpha} r$ we may argue in a similar way as above. □

We now turn to the question of translations between Plain CHOCS and Mobile Processes. First we give a translation of Plain CHOCS without the renaming construct into Mobile Processes. This subset of Plain CHOCS corresponds very closely to the informal idea of encoding process passing in Mobile Processes described in [MilParWal89]. This translation carries no additional parameters which shows that Plain CHOCS programs can be viewed as a set of derived operators in Mobile Processes.

Definition 5.4.6 $[\![\,]\!] : PlainCHOCS \rightarrow MP$

$$
\begin{aligned}
[\![nil]\!] &= 0 \\
[\![a?x.p]\!] &= a(x).[\![p]\!] \\
[\![a!p'.p]\!] &= (b)(\overline{a}b.([\![p]\!] \mid b \Rightarrow [\![p']\!])),\ b \notin fn(p) \cup fn(p') \cup \{a\} \\
[\![\tau.p]\!] &= \tau.[\![p]\!] \\
[\![p + p']\!] &= [\![p]\!] + [\![p']\!] \\
[\![p \mid p']\!] &= [\![p]\!] \mid [\![p']\!] \\
[\![p\backslash a]\!] &= (a)[\![p]\!] \\
[\![x]\!] &= \overline{x}.0
\end{aligned}
$$

Note how a process variable in Plain CHOCS is translated into a process which is only capable of synchronizing on the x channel and then stop. This is exactly the idea described in [MilParWal89] of an executor to trigger the start of the process.

An interesting point to note about the above translation is that only a rather special kind of recursion is needed. We only need a construction which provides "copies" of the process to be sent. This construction resembles a Kleene-star operator. Combining this with conjecture 5.3.15 (which would show that general recursion may be simulated in Plain CHOCS) we see that using this Kleene-star operator and

the dynamic interconnection mechanism provided by Mobile Processes we may simulate recursion in e.g. CCS. In fact we do not need to appeal to conjecture 5.3.15 to show this; The lemmas above suffice to prove $(z)(\overline{z}.0 \mid z \Rightarrow p[\overline{z}.0/X]) \dot{\sim} \mathtt{rec}\,\tau.p$ if $z \notin fn(p)$.

Note that this translation ensures static scope for the restriction operator since the process p' being "sent" stays in the "sending" environment e.g:

$$[\![a?x.(x \mid x) \mid (a!p'.p)\backslash c]\!] = a(x).(\overline{x}.0 \mid \overline{x}.0) \mid (c)(\overline{a}(b).((b \Rightarrow [\![p']\!]) \mid [\![p]\!]))$$
$$\downarrow_\tau$$
$$(b)(\overline{b}.0 \mid \overline{b}.0 \mid (c)((b \Rightarrow [\![p']\!]) \mid [\![p]\!]))$$
$$\downarrow_\tau$$
$$(b)(\overline{b}.0 \mid 0 \mid (c)([\![p']\!] \mid (b \Rightarrow [\![p']\!]) \mid [\![p]\!]))$$
$$\downarrow_\tau$$
$$(b)(0 \mid 0 \mid (c)([\![p']\!] \mid [\![p']\!] \mid (b \Rightarrow [\![p']\!]) \mid [\![p]\!]))$$
$$\dot{\sim}$$
$$(c)([\![p']\!] \mid [\![p']\!] \mid [\![p]\!])$$

In this example we see how the recursion in the translation of the output prefix ensures that a sufficient number of copies of the process to be passed is provided.

As we can see from the above example the translated terms need an additional τ–move to simulate the substitution. Let us specify this at the Plain CHOCS level by introducing a notion we call τ–substitution $[\ /\ ]_\tau$. This substitution is defined as $[p/x]_\tau = [\tau.p/x]$. In the following two propositions let $\longrightarrow$ be a transition relation defined as the transition relation of definition 5.1.6, but with $[\ /\ ]_\tau$ instead of $[\ /\ ]$ in the com-close-rule. Let $\dot{\sim}_\tau$ be the applicative higher order bisimulation equivalence defined as in definition 5.2.1 relative to the new transition system with τ–substitution instead of the usual substitution in clause (i). Using these definitions we can now formally relate the two calculi. In the following $\dot{\sim}$ is the strong ground bisimulation defined in [MilParWal89].

Proposition 5.4.7 $[\![p[q/x]_\tau]\!] \dot{\sim} (b)([\![p]\!]\{b/x\} \mid b \Rightarrow [\![q]\!])$ *where* $b \notin fn(p) \cup fn(q)$

PROOF: By structural induction on p using lemma 5.4.3 to lemma 5.4.5.

$p \equiv nil$	$[\![nil[q/x]_\tau]\!]$	$\equiv$	by definition of $[\ /\ ]_\tau$
	$[\![nil]\!]$	$=$	by definition of $[\![\ ]\!]$
	0	$\dot{\sim}$	by algebraic laws
	$(b)(0\{b/x\} \mid (b \Rightarrow [\![q]\!]))$	$\dot{\sim}$	by definition of $[\![\ ]\!]$
	$(b)([\![nil]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!]))$		

$p \equiv a?y.p_1$ Assume $y \neq x$ and $y \notin FV(q)$ (otherwise use α-conversion)

$[\![(a?y.p_1)[q/x]_\tau]\!] \equiv$ by definition of $[\ /\ ]_\tau$

$[\![a?y.(p_1[q/x]_\tau)]\!] =$ by definition of $[\![\ ]\!]$

$a(y).[\![(p_1[q/x]_\tau)]\!] \dot{\sim}$ by I.H.

$a(y).(b)([\![p_1]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!])) \dot{\sim}$ since $b \notin fn(p)$

$(b)(a(y).([\![p_1]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!]))) \dot{\sim}$ since $y \notin FV(q)$ and $y \neq x$

$(b)((a(y).[\![p_1]\!])\{b/x\} \mid (b \Rightarrow [\![q]\!])) =$ by definition of $[\![\ ]\!]$

$(b)(([\![a?y.p_1]\!])\{b/x\} \mid (b \Rightarrow [\![q]\!]))$

$p \equiv a!p_1.p_2$ $[\![(a!p_1.p_2)[q/x]_\tau]\!]$

$\equiv$ by definition of $[\ /\ ]_\tau$

$[\![a!(p_1[q/x]_\tau).(p_2[q/x]_\tau)]\!]$

$=$ by definition of $[\![\ ]\!]$

$(b)(\overline{a}b.((b \Rightarrow [\![p_1[q/x]_\tau]\!]) \mid [\![p_2[q/x]_\tau]\!]))$

$\dot{\sim}$ by I.H. $(b \notin fn(p_1) \cup fn(p_2) \cup fn(q))$

$(b)(\overline{a}b.((c)((b \Rightarrow (c)([\![p_1]\!]\{c/x\} \mid (c \Rightarrow [\![q]\!])))$

$\mid (c)([\![p_2]\!]\{c/x\} \mid (c \Rightarrow [\![q]\!])))))$

$\dot{\sim}$ by lemma 5.4.5.

$(b)(\overline{a}b.(c)((b \Rightarrow [\![p_1]\!]\{c/x\}) \mid [\![p_2]\!]\{c/x\} \mid (c \Rightarrow [\![q]\!])))$

$\dot{\sim}$ since $c \neq a$ and $c \neq b$, otherwise use α-conversion on c

$(c)(b)(\overline{a}b.((b \Rightarrow [\![p_1]\!]\{c/x\}) \mid [\![p_2]\!]\{c/x\} \mid (c \Rightarrow [\![q]\!])))$

$\equiv$ by definition of $\{\ /\ \}$

$(c)(b)(\overline{a}b.(((b \Rightarrow [\![p_1]\!]) \mid [\![p_2]\!])\{c/x\} \mid (c \Rightarrow [\![q]\!])))$

$\dot{\sim}$

$(c)((b)(\overline{a}b.(((b \Rightarrow [\![p_1]\!]) \mid [\![p_2]\!])\{c/x\}) \mid (c \Rightarrow [\![q]\!])))$

$\dot{\sim}$ since $b \notin fn(q)$

$(c)([\![a!p_1.p_2]\!]\{c/x\} \mid (c \Rightarrow [\![q]\!]))$

$p \equiv p_1 + p_2$ $[\![(p_1 + p_2)[q/x]_\tau]\!]$

$\equiv$ by definition of $[\ /\ ]_\tau$ and by definition of $[\![\ ]\!]$

$[\![p_1[q/x]_\tau]\!] + [\![p_2[q/x]_\tau]\!]$

$\dot{\sim}$ by I.H.

$(b)([\![p_1]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!])) + (b)([\![p_2]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!]))$

$\dot{\sim}$ by lemma 5.4.3.

$(b)(([\![p_1]\!]\{b/x\} + [\![p_2]\!]\{b/x\}) \mid (b \Rightarrow [\![q]\!]))$

$\equiv$ by definition of $\{\ /\ \}$

$(b)([\![p_1 + p_2]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!]))$

$p \equiv p_1 \mid p_2$ $[\![(p_1 \mid p_2)[q/x]_\tau]\!]$

$\equiv$ by definition of $[\ /\ ]_\tau$ and by definition of $[\![\ ]\!]$

$[\![p_1[q/x]_\tau]\!] \mid [\![p_2[q/x]_\tau]\!]$

$\dot{\sim}$ by I.H.

$(b)([\![p_1]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!])) \mid (b)([\![p_2]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!]))$

$\dot{\sim}$ by lemma 5.4.4.

$(b)(([\![p_1]\!]\{b/x\} \mid [\![p_2]\!]\{b/x\}) \mid (b \Rightarrow [\![q]\!]))$

$\equiv$ by definition of $\{\ /\ \}$

$(b)([\![p_1 \mid p_2]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!]))$

$p \equiv p_1\backslash a$ Assume $a \notin fn(q)$ otherwise use α-conversion.

$$
\begin{array}{lll}
[\![(p_1\backslash a)[q/x]_\tau]\!] & \equiv & \text{by definition of } [\ /\ ]_\tau \\
[\![(p_1[q/x]_\tau)\backslash a]\!] & = & \text{by definition of } [\![\ ]\!] \\
(a)([\![p_1[q/x]_\tau]\!]) & \dot{\sim} & \text{by I.H.} \\
(a)(b)([\![p_1]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!])) & \dot{\sim} & \text{since } a \notin fn(q) \\
(b)((a)([\![p_1]\!]\{b/x\}) \mid (b \Rightarrow [\![q]\!])) & \equiv & \text{by definition of } \{\ /\ \} \\
(b)([\![p_1\backslash a]\!]\{b/x\} \mid (b \Rightarrow [\![q]\!])) & &
\end{array}
$$

$p \equiv y$ if $y \neq x$ then

$$
\begin{array}{lll}
[\![y[q/x]_\tau]\!] & \equiv & \\
[\![y]\!] & = & \\
\overline{y}.0 & \dot{\sim} & \\
(b)((\overline{y}.0)\{b/x\} \mid (b \Rightarrow [\![q]\!])) & &
\end{array}
$$

if $y = x$ then

$$
\begin{array}{lll}
[\![y[q/x]_\tau]\!] & \equiv & \\
[\![\tau.q]\!] & = & \\
\tau.[\![q]\!] & \dot{\sim} & \\
(b)((\overline{y}.0)\{b/x\} \mid (b \Rightarrow [\![q]\!])) & &
\end{array}
$$

□

Proposition 5.4.8

1. *if* $p \xrightarrow{a?x} p'$ *then* $[\![p]\!] \xrightarrow{a(x)} [\![p']\!]$
2. *if* $p \xrightarrow{a!_B p'} p''$ *then* $[\![p]\!] \xrightarrow{\overline{a}(b)} q \dot{\sim} (b_1)\ldots(b_n)(b \Rightarrow [\![p']\!] \mid [\![p'']\!])$ *where* $B = \{b_1, \ldots, b_n\}$ *for some* q.
3. *if* $p \xrightarrow{\tau} p'$ *then* $[\![p]\!] \xrightarrow{\tau} [\![p']\!]$
4. *if* $q \dot{\sim} [\![p]\!]$ *and* $q \xrightarrow{a(x)} q'$ *then* $p \xrightarrow{a?x} p'$ *for some* p' *with* $q'\{b/x\} \dot{\sim} [\![p']\!]\{b/x\}$ *for all* $b \in Names$.
5. *if* $q \dot{\sim} [\![p]\!]$ *then* $q \overset{\overline{a}b}{\not\rightarrow}$.
6. *if* $q \dot{\sim} [\![p]\!]$ *and* $q \xrightarrow{\overline{a}(b)} q'$ *then* $p \xrightarrow{a!_B p'} p''$ *with* $q' \dot{\sim} (b_1)\ldots(b_n)(b \Rightarrow [\![p']\!] \mid [\![p'']\!])$ *for some* B, p', p'' *where* $B = \{b_1, \ldots, b_n\}$.
7. *if* $q \dot{\sim} [\![p]\!]$ *and* $q \xrightarrow{\tau} q'$ *then* $p \xrightarrow{\tau} p'$ *with* $q' \dot{\sim} [\![p']\!]$ *for some* p'.

PROOF:

1. By induction on the length of the inference used to establish $p \xrightarrow{a?x} p'$ observing the structure of the process p. The cases when $p \equiv nil$, $p \equiv a!p_1.p_2$ and $p \equiv \tau.p_1$ are trivial since $p \overset{a?x}{\not\rightarrow}$.

$p \equiv a?x.p_1$ Then $a?x.p_1 \xrightarrow{a?x} p_1$ by the input-rule and $p' \equiv p_1$. Also $[\![a?x.p_1]\!] = a(x).[\![p_1]\!] \xrightarrow{a(x)} [\![p_1]\!]$ by the INPUT-ACT-rule.

$p \equiv p_1 + p_2$ If $p \xrightarrow{a?x} p'$ then

either $p_1 \xrightarrow{a?x} p'$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{a(x)} [\![p']\!]$ and by the SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{a(x)} [\![p']\!]$.

or $p_2 \xrightarrow{a?x} p'$ by a shorter inference, and by induction we have $[\![p_2]\!] \xrightarrow{a(x)} [\![p']\!]$ and by the SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{a(x)} [\![p']\!]$.

$p \equiv p_1 \mid p_2$ If $p \xrightarrow{a?x} p'$ then

either $p_1 \xrightarrow{a?x} p_1'$ and $p' \equiv p_1' \mid p_2$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{a(x)} [\![p_1']\!]$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{a(x)} [\![p']\!]$.

or $p_2 \xrightarrow{a?x} p_2'$ and $p' \equiv p_1 \mid p_2'$ by a shorter inference, and by induction we have $[\![p_2]\!] \xrightarrow{a(x)} [\![p_2']\!]$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{a(x)} [\![p']\!]$.

$p \equiv p_1 \backslash b$ If $p \xrightarrow{a?x} p'$ then $p_1 \xrightarrow{a?x} p_1'$ with $a \neq b$ and $p' \equiv p_1' \backslash b$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{a(x)} [\![p_1']\!]$ and by the RES-rule we have $[\![p_1 \backslash b]\!] \xrightarrow{a(x)} [\![p']\!]$.

2. By induction on the length of the inference used to establish $p \xrightarrow{a!_{BP'}} p''$ observing the structure of the process p. The cases when $p \equiv nil$, $p \equiv a?x.p_1$ and $p \equiv \tau.p_1$ are trivial since $p \overset{a!_{BP'}}{\not\longrightarrow}$.

$p \equiv a!p_1.p_2$ Then $a!p_1.p_2 \xrightarrow{a!_\emptyset p_1} p_2$ by the output-rule. Also $[\![a!p_1.p_2]\!] \xrightarrow{\overline{a}(b)} (b \Rightarrow [\![p_1]\!]) \mid [\![p_2]\!]$ by the INPUT-ACT-rule.

$p \equiv p_1 + p_2$ If $p \xrightarrow{a!_{BP'}} p''$ then

either $p_1 \xrightarrow{a!_{BP'}} p''$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p'']\!])$ and by the SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p'']\!])$

or $p_2 \xrightarrow{a!_{BP'}} p''$ by a shorter inference, and by induction we have $[\![p_2]\!] \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p'']\!])$ and by the SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p'']\!])$

$p \equiv p_1 \mid p_2$ If $p \xrightarrow{a!_{BP'}} p''$ then

either $p_1 \xrightarrow{a!_{BP'}} p_1''$ and $p'' \equiv p_1'' \mid p_2$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{\overline{a}(b)} p_1''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p_1'']\!])$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p_1'']\!]) \mid [\![p_2]\!] \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p'']\!])$ (using a suitable α-conversion).

or $p_2 \xrightarrow{a!_B p'} p_2''$ and $p'' \equiv p_1 \mid p_2''$ by a shorter inference, and by induction we have $[\![p_2]\!] \xrightarrow{\overline{a}(b)} p_2''' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p_2'']\!])$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\overline{a}(b)} p''' \dot{\sim} [\![p_1]\!] \mid (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p_2'']\!]) \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p'']\!])$ (using a suitable α-conversion).

$p \equiv p_1\backslash d$ If $p \xrightarrow{a!_B p'} p''$ then

either $p_1 \xrightarrow{a!_{B'} p'} p_1''$ by a shorter inference and $d \in (fn(p') \cap fn(p_1'')) \setminus B$ and $B = B' \cup \{d\}$ and $a \neq d$ and $p'' \equiv p_1''\backslash d$. By induction we have $[\![p_1]\!] \xrightarrow{\overline{a}(b)} p_1''' \dot{\sim} (b_1)\ldots(b_k)((b \Rightarrow [\![p']\!]) \mid [\![p_1'']\!])$ and by the RES-rule we have $[\![p_1\backslash d]\!] = (d)([\![p_1]\!]) \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (d)(b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p_1'']\!])$.

or $p_1 \xrightarrow{a!_B p_1'} p_1''$ by a shorter inference and $d \notin fn(p') \cap fn(p_1'')$ and $a \neq d$ and $p' \equiv p_1'\backslash d$. and $p'' \equiv p_1''\backslash d$. By induction we have $[\![p_1]\!] \xrightarrow{\overline{a}(b)} p_1''' \dot{\sim} (b_1)\ldots(b_k)((b \Rightarrow [\![p']\!]) \mid [\![p_1'']\!])$ and by the RES-rule we have $[\![p_1\backslash d]\!] = (d)([\![p_1]\!]) \xrightarrow{\overline{a}(b)} p''' \dot{\sim} (d)(b_1)\ldots(b_n)((b \Rightarrow [\![p']\!]) \mid [\![p_1'']\!]) \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow (d)[\![p_1']\!]) \mid (d)[\![p_1'']\!])$.

3. By induction on the length of the inference used to establish $p \xrightarrow{\tau} p'$ observing the structure of the process p. The cases when $p \equiv nil$, $p \equiv a?x.p_1$ and $p \equiv a!p_1.p_2$ are trivial since $p \not\xrightarrow{\tau}$.

$p \equiv \tau.p_1$ Then $p \xrightarrow{\tau} p_1$ by the tau-rule and $p' \equiv p_1$. Also $[\![p]\!] = \tau.[\![p_1]\!] \xrightarrow{\tau} [\![p_1]\!]$ by the TAU-ACT-rule.

$p \equiv p_1 + p_2$ If $p \xrightarrow{\tau} p'$ then

either $p_1 \xrightarrow{\tau} p'$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{\tau} [\![p']\!]$ and by the SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{\tau} [\![p']\!]$.

or $p_2 \xrightarrow{\tau} p'$ by a shorter inference, and by induction we have $[\![p_2]\!] \xrightarrow{\tau} [\![p']\!]$ and by the SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{\tau} [\![p']\!]$.

$p \equiv p_1 \mid p_2$ If $p \xrightarrow{\tau} p'$ then

either $p_1 \xrightarrow{\tau} p_1'$ and $p' \equiv p_1' \mid p_2$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{\tau} [\![p_1']\!]$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\tau} [\![p']\!]$.

or $p_2 \xrightarrow{\tau} p_2'$ and $p' \equiv p_1 \mid p_2'$ by a shorter inference, and by induction we have $[\![p_2]\!] \xrightarrow{\tau} [\![p_2']\!]$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\tau} [\![p']\!]$.

or $p_1 \xrightarrow{a?x} p_1'$ and $p_2 \xrightarrow{a!_B p_2'} p_2''$ by shorter inferences and $p' \equiv (p_1'[p_2'/x]_\tau \mid p_2'')\backslash B$. By induction and propositions 5.4.8.1 and 5.4.8.2 we have $[\![p_1]\!] \xrightarrow{a(x)} [\![p_1']\!]$ and $[\![p_2]\!] \xrightarrow{\overline{a}(b)} (b_1)\ldots(b_n)((b \Rightarrow [\![p_2']\!]) \mid [\![p_2'']\!])$. Then by the CLOSE-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\tau} (b)([\![p']\!]\{b/x\} \mid (b_1)\ldots(b_n)((b \Rightarrow [\![p_2']\!]) \mid [\![p_2'']\!])) \dot{\sim} (b_1)\ldots(b_n)((b)([\![p']\!]\{b/x\} \mid (b \Rightarrow [\![p_2']\!]) \mid [\![p_2'']\!])) =$

$[\![(p_1'[p_2'/x]_\tau \mid p_2'')\backslash B]\!]$ by proposition 5.4.7 and assuming $B \cap fn(p') = \emptyset$ (otherwise use α-conversion).

or $p_2 \xrightarrow{a?x} p_2'$ and $p_1 \xrightarrow{a!_B p_1'} p_1''$ and we may argue as above.

$p \equiv p_1\backslash b$ If $p \xrightarrow{\tau} p'$ then $p_1 \xrightarrow{\tau} p_1'$ by a shorter inference, and by induction we have $[\![p_1]\!] \xrightarrow{\tau} [\![p_1']\!]$ and by the RES-rule we have $[\![p_1\backslash b]\!] = (b)([\![p_1]\!]) \xrightarrow{\tau} (b)([\![p_1]\!]) = [\![p']\!]$.

4. Assume $q \dot{\sim} [\![p]\!]$ and $q \xrightarrow{a(x)} q'$. Then $[\![p]\!] \xrightarrow{a(x)} q''$ for some q'' with $q'\{b/x\} \dot{\sim} q''\{b/x\}$ for all $b \in Names$ since $q \dot{\sim} [\![p]\!]$.
We proceed by induction on the length of the inference used to establish $[\![p]\!] \xrightarrow{a(x)} q''$ observing the structure of p.
If $[\![p]\!] \xrightarrow{a(x)} q''$ then p must have one of the following forms:

$p \equiv a?x.p_1$ In this case $[\![p]\!] \xrightarrow{a(x)} [\![p_1]\!]$. By the input-rule we have $a?x.p_1 \xrightarrow{a?x} p_1$ which proves the lemma in this case.

$p \equiv p_1 + p_2$ In this case

either $[\![p_1]\!] \xrightarrow{a(x)} q''$ by a shorter inference and by induction $p_1 \xrightarrow{a?x} p_1'$ and $q''\{b/x\} \dot{\sim} [\![p_1']\!]\{b/x\}$ for all $b \in Names$. By the sum-rule we have $p_1 + p_2 \xrightarrow{a?x} p_1'$ and $q''\{b/x\} \dot{\sim} [\![p_1']\!]\{b/x\}$ for all $b \in Names$.

or $[\![p_2]\!] \xrightarrow{a(x)} q''$ by a shorter inference and by induction $p_2 \xrightarrow{a?x} p_2'$ and $q''\{b/x\} \dot{\sim} [\![p_2']\!]\{b/x\}$ for all $b \in Names$. By the sum-rule we have $p_1 + p_2 \xrightarrow{a?x} p_2'$ and $q''\{b/x\} \dot{\sim} [\![p_2']\!]\{b/x\}$ for all $b \in Names$.

$p \equiv p_1 \mid p_2$ In this case

either $[\![p_1]\!] \xrightarrow{a(x)} q_1''$ by a shorter inference and $q'' \equiv q_1'' \mid [\![p_2]\!]$. By induction $p_1 \xrightarrow{a?x} p_1'$ and $q_1'\{b/x\} \dot{\sim} [\![p_1']\!]\{b/x\}$ for all $b \in Names$. By the par-rule we have $p_1 \mid p_2 \xrightarrow{a?x} p_1' \mid p_2$ and $q''\{b/x\} \dot{\sim} [\![p_1' \mid p_2]\!]\{b/x\}$ for all $b \in Names$.

or $[\![p_2]\!] \xrightarrow{a(x)} q_2''$ by a shorter inference and $q'' \equiv [\![p_1]\!] \mid q_2''$. By induction $p_2 \xrightarrow{a?x} p_2'$ and $q_2' \dot{\sim} [\![p_2']\!]$. By the par-rule we have $p_1 \mid p_2 \xrightarrow{a?x} p_1 \mid p_2'$ and $q''\{b/x\} \dot{\sim} [\![p_1 \mid p_2']\!]\{b/x\}$ for all $b \in Names$.

$p \equiv p_1\backslash c$ In this case $[\![p_1]\!] \xrightarrow{a(x)} q_1''$ and $a \neq c$. By induction $p_1 \xrightarrow{a?x} p_1'$ and $[\![p_1]\!]\{b/x\} \dot{\sim} q_1''\{b/x\}$ for all $b \in Names$. By the res-rule we have $p_1\backslash c \xrightarrow{a?x} p_1'\backslash c$ and $q''\{b/x\} \dot{\sim} [\![p_1'\backslash c]\!]\{b/x\}$ for all $b \in Names$.

5. From the definition of $[\![\ ]\!]$ it is easy to see that $[\![p]\!] \overset{\overline{a}b}{\not\rightarrow}$. Since $q \dot{\sim} [\![p]\!]$ this must be true for q.

6. Assume $q \dot{\sim} [\![p]\!]$ and $q \xrightarrow{\overline{a}(b)} q'$. Then $[\![p]\!] \xrightarrow{\overline{a}(b)} q''$ with $q' \dot{\sim} q''$, since $q \dot{\sim} [\![p]\!]$.
We proceed by induction on the length of the inference used to establish $[\![p]\!] \xrightarrow{\overline{a}(b)} q''$ observing the structure of p.
If $[\![p]\!] \xrightarrow{\overline{a}(b)} q''$ then p must have one of the following forms:

   $p \equiv a!p_1.p_2$ From the output-rule we have $p \xrightarrow{a!_\emptyset p_1} p_2$ and from the OUTPUT-ACT-rule we have $[\![p]\!] \xrightarrow{\overline{a}(b)} (b \Rightarrow [\![p_1]\!]) \mid [\![p_2]\!]$ which proves the lemma in this case.

   $p \equiv p_1 + p_2$ **either** $[\![p_1]\!] \xrightarrow{\overline{a}(b)} q''$ by a shorter inference and by induction we have $p_1 \xrightarrow{a!_B p_1'} p_1''$ and $q'' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p_1']\!]) \mid [\![p_1'']\!])$. Then $p_1 + p_2 \xrightarrow{a!_B p_1'} p_1''$ by the sum-rule and by the SUM-rule we have $[\![p]\!] \xrightarrow{\overline{a}(b)} q''$ which proves the lemma in this case.
   or $[\![p_2]\!] \xrightarrow{\overline{a}(b)} q''$ and an argument as above applies.

   $p \equiv p_1 \mid p_2$ **either** $[\![p_1]\!] \xrightarrow{\overline{a}(b)} q_1''$ by a shorter inference and $q'' \dot{\sim} q_1'' \mid [\![p_2]\!]$. By induction we have $p_1 \xrightarrow{a!_B p_1'} p_1''$ with $q_1'' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p_1']\!]) \mid [\![p_1'']\!])$. By the par-rule we have $p_1 \mid p_2 \xrightarrow{a!_B p_1'} p_1'' \mid p_2$ and by the PAR-rule and RES-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\overline{a}(b)} q'' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p_1']\!]) \mid [\![p_1'' \mid p_2]\!])$ by a suitable α-conversion such that $B \cap fn(p_2) = \emptyset$.
   or $[\![p_2]\!] \xrightarrow{\overline{a}(b)} q_2''$ and symmetric arguments as above yield the result.

   $p \equiv p_1 \backslash d$ Then $[\![p_1]\!] \xrightarrow{\overline{a}(b)} q_1''$ by a shorter inference and $a \neq d$ and $q'' \equiv (d)(q_1'')$. By induction we have $p_1 \xrightarrow{a!_B p_1'} p_1''$ with $q_1'' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p_1']\!]) \mid [\![p_1'']\!])$. If $d \notin fn(p_1') \cup fn(p_1'')$ then by the res-rule we have $p_1 \backslash d \xrightarrow{a!_B p_1' \backslash d} p_1'' \backslash d$ and by the RES-rule we have $[\![p]\!] \xrightarrow{\overline{a}(b)} q'' \dot{\sim} (b_1)\ldots(b_n)(d)((b \Rightarrow [\![p_1']\!]) \mid [\![p_1'']\!]) \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow (d)[\![p_1']\!]) \mid (d)[\![p_1'']\!])$.
   If $d \in (fn(p_1') \cap fn(p_1'')) \setminus B$ then by the open-rule we have $p_1 \backslash d \xrightarrow{a!_{B \cup \{d\}} p_1'} p_1''$ and by the RES-rule we have $[\![p]\!] \xrightarrow{\overline{a}(b)} q'' \dot{\sim} (b_1)\ldots(b_n)(d)((b \Rightarrow [\![p_1']\!]) \mid [\![p_1'']\!])$.

7. Assume $q \dot{\sim} [\![p]\!]$ and $q \xrightarrow{\tau} q'$. Then $[\![p]\!] \xrightarrow{\tau} q''$ with $q' \dot{\sim} q''$ since $q \dot{\sim} [\![p]\!]$.
We proceed by induction on the length of the inference used to establish $[\![p]\!] \xrightarrow{\tau} q''$ observing the structure of the process p.
If $[\![p]\!] \xrightarrow{\tau} q''$ then p must have one of the following forms:

   $p \equiv \tau.p_1$ Then by the tau-rule we have $p \xrightarrow{\tau} p_1$ and by the TAU-rule $[\![p]\!] \xrightarrow{\tau} [\![p_1]\!]$ which proves the lemma in this case.

   $p \equiv p_1 + p_2$ **either** $[\![p_1]\!] \xrightarrow{\tau} q''$ by a shorter inference. By induction we have $p_1 \xrightarrow{\tau} p_1'$ with $q'' \dot{\sim} [\![p_1']\!]$. By the sum-rule $p_1 + p_2 \xrightarrow{\tau} p_1'$ and by the

SUM-rule we have $[\![p_1 + p_2]\!] \xrightarrow{\tau} q''$

or $[\![p_2]\!] \xrightarrow{\tau} q''$ and a similar argument as above applies.

$p \equiv p_1 \mid p_2$ **either** $[\![p_1]\!] \xrightarrow{\tau} q_1''$ by a shorter inference and $q'' \equiv q_1'' \mid [\![p_2]\!]$. By induction we have $p_1 \xrightarrow{\tau} p_1'$ with $q_1'' \dot{\sim} [\![p_1']\!]$. By the par-rule $p_1 \mid p_2 \xrightarrow{\tau} p_1' \mid p_2$ and by the PAR-rule we have $[\![p_1 \mid p_2]\!] \xrightarrow{\tau} q_1'' \mid [\![p_2]\!] \dot{\sim} [\![p_1' \mid p_2]\!]$

or $[\![p_2]\!] \xrightarrow{\tau} q_2''$ and an argument as above applies.

or $[\![p_1]\!] \xrightarrow{a(x)} q_1'$ and $[\![p_2]\!] \xrightarrow{\overline{a}(b)} q_2'$ by shorter inferences and $q'' \dot{\sim} (b)(q_1'\{b/x\} \mid q_2'')$ modulo the appropriate α-conversions. By proposition 5.4.8.4 we have $p_1 \xrightarrow{a?x} p_1'$ with $q_1'\{b/x\} \dot{\sim} [\![p_1']\!]\{b/x\}$ for all $b \in Names$ and by proposition 5.4.8.6 we have $p_2 \xrightarrow{a!_B p_2'} p_2''$ with $q_2' \dot{\sim} (b_1)\ldots(b_n)((b \Rightarrow [\![p_2']\!]) \mid [\![p_2'']\!])$. By the com-close-rule we have $p_1 \mid p_2 \xrightarrow{\tau} (p_1'[p_2'/x]_\tau \mid p_2'')\backslash B$ assuming $B \cap fn(p_1') = \emptyset$ (otherwise use a suitable α-conversion). By the COM-rule we have
$[\![p_1 \mid p_2]\!] \xrightarrow{\tau} q'' \dot{\sim} (b)([\![p_1']\!]\{b/x\} \mid (b_1)\ldots(b_n)((b \Rightarrow [\![p_2']\!]) \mid [\![p_2'']\!]))$
$\dot{\sim} [\![(p_1'[p_2'/x]_\tau \mid p_2'')\backslash B]\!]$ according to proposition 5.4.7.

or $[\![p_1]\!] \xrightarrow{\overline{a}(b)} q_1'$ and $[\![p_2]\!] \xrightarrow{a(x)} q_2'$ which is a symmetric case to the above.

$p \equiv p_1\backslash b$ Then $[\![p_1]\!] \xrightarrow{\tau} q_1''$ with $q'' \equiv (b)(q_1'')$ by a shorter inference. By induction $p_1 \xrightarrow{\tau} p_1'$ with $q_1' \dot{\sim} [\![p_1']\!]$. By the res-rule we have $p_1\backslash b \xrightarrow{\tau} p_1'\backslash b$ and by the RES-rule we have $[\![p_1\backslash b]\!] \xrightarrow{\tau} [\![p_1'\backslash b]\!]$.

□

The above proposition shows a strong connection between transitions of processes in Plain CHOCS and their translations into Mobile Processes. We have so far been unsuccessful in proving that the translation preserves equivalence but we conjecture that this holds under certain restrictions on the observations we allow ourselves i.e.:

Conjecture 5.4.9 $p \dot{\sim}_\tau q \Leftrightarrow [\![p]\!] \dot{\sim} [\![q]\!]$

An immediate attempt to prove the above conjecture is to show that the realtion:

$$R_1 = \{(q_1, q_2) : \exists p_1, p_2 . q_1 \dot{\sim} [\![p_1]\!], q_2 \dot{\sim} [\![p_2]\!], p_1 \dot{\sim}_\tau p_2\}$$

is a strong ground bisimulation and that the relation

$$R_2 = \{(p_1, p_2) : [\![p_1]\!] \dot{\sim} [\![p_2]\!]\}$$

is an applicative higher order bisimulation w.r.t. $[\ /\ ]_\tau$. Unfortunately this attempt has so far been unsuccessful. The reason for this is that for relation R_1 I have

been unable to prove that if $q_1 \xrightarrow{a(x)} q_1'$ then $q_2 \xrightarrow{a(x)} q_2'$ and $q_1'\{b/x\} \dot{\sim} q_2'\{b/x\}$ for all $b \in Names$ from $p_1 \xrightarrow{a?x} p_1'$ and $p_2 \xrightarrow{a?x} p_2'$ and $p_1'[r/x]_\tau \dot{\sim}_\tau p_2'[r/x]_\tau$ for all r. $[\![p_1'[r/x]_\tau]\!] \dot{\sim} [\![p_2'[r/x]_\tau]\!]$ does not seem to imply $[\![p_1']\!]\{b/x\} \dot{\sim} [\![p_2']\!]\{b/x\}$ for all $b \in Names$. For relation R_2 I have been unable to prove that if $p_1 \xrightarrow{a!_{BP_1'}} p_1''$ then $p_2 \xrightarrow{a!_{BP_2'}} p_2''$ and $p_1' \dot{\sim}_\tau p_2'$ and $p_1'' \dot{\sim}_\tau p_2''$ from $q_1 \xrightarrow{\overline{a}(b)} q_1' \dot{\sim} (b_1)\ldots(b_n)(b \Rightarrow [\![p_1']\!] \mid [\![p_1'']\!])$ and $q_2 \xrightarrow{\overline{a}(b)} q_2' \dot{\sim} (b_1)\ldots(b_n)(b \Rightarrow [\![p_2']\!] \mid [\![p_2'']\!])$ and $q_1' \dot{\sim} q_2'$. It does not seem to be possible to infer from $(b \Rightarrow [\![p_1']\!] \mid [\![p_1'']\!]) \dot{\sim} (b \Rightarrow [\![p_2']\!] \mid [\![p_2'']\!])$ that $[\![p_1']\!] \dot{\sim} [\![p_2']\!]$ and $[\![p_1'']\!] \dot{\sim} [\![p_2'']\!]$.

The above only applies to the sublanguage of Plain CHOCS where the renaming operator has been omitted. The type of systems we can describe in this language is limited in the sense that there is no real need for passing the process in the communication since the receiving process can do no more than copy it and start each copy at different times. This is reflected in the above translation in the sense that the process to be "sent" stays in the sending environment and the "receiving" process only receives a link which can be used to trigger copies of the "received" process. The renaming construct allows us to change the way we communicate with each copy by renaming some of the free names to locally bound names. This may be incorporated into the translation by extending the translation function by a list of names L i.e.:

Definition 5.4.10 $[\![\,]\!] : Plain\,CHOCS \rightarrow Names^* \rightarrow MP$

$$
\begin{aligned}
[\![nil]\!]L &= 0 \\
[\![a?x.p]\!]L &= a(x).[\![p]\!]L \\
[\![a!p'.p]\!]L &= (b)(\overline{a}b.([\![p]\!]L \mid b(L) \Rightarrow [\![p']\!]L)),\ b \notin fn(p) \cup fn(p') \cup \{a\} \cup L \\
[\![\tau.p]\!]L &= \tau.[\![p]\!]L \\
[\![p + p']\!]L &= [\![p]\!]L + [\![p']\!]L \\
[\![p \mid p']\!]L &= [\![p]\!]L \mid [\![p']\!]L \\
[\![p\backslash a]\!]L &= (d)[\![\{d/a\}p]\!]L \ \ \textit{where } d \notin fn(p\backslash a) \cup L \\
[\![p[a \mapsto b]]\!]L &= \{b/a\}([\![p]\!]L) \\
[\![x]\!]L &= \overline{x}L.0
\end{aligned}
$$

where $b(L).p$ *means* $b(l_1).\ldots.b(l_n).p$ *and* $\overline{b}L.p$ *means* $\overline{b}l_1.\ldots.\overline{b}l_n.p$ *for* $L = \{l_1, \ldots, l_n\}$.

When translating a Plain CHOCS expression p we then instantiate L to a list consisting of the elements of $fn(p)$ to obtain the desired effect.

Let us consider the following small example to give an idea about how the above translation works:

Assume $\{a,b\} = fn(p) \cup fn(p') \cup fn(p'')$ and $b' \notin \{a,b\}$.

$$[\![a?x.(x[b \mapsto b'] \mid b'?x.p)\backslash b' \mid a!p'.p'']\!]_{[a,b]}$$
$$=$$
$$a(x).(b')(\overline{x}ab'.0 \mid b'(x).[\![p]\!]_{[a,b]}) \mid (c)(\overline{a}c.((c(a)(b) \Rightarrow [\![p']\!]_{[a,b]}) \mid [\![p'']\!]_{[a,b]}))$$
$$\downarrow_\tau$$
$$(c)((b')(\overline{c}ab'.0 \mid b'(x).[\![p]\!]_{[a,b]}) \mid (c(a)(b) \Rightarrow [\![p']\!]_{[a,b]}) \mid [\![p'']\!]_{[a,b]})$$
$$\downarrow_\tau$$
$$\downarrow_\tau$$
$$(c)((b')(0 \mid b'(x).[\![p]\!]_{[a,b]} \mid (\{a/a\}\{b'/b\}([\![p']\!]_{[a,b]}))) \mid (c(a)(b) \Rightarrow [\![p']\!]_{[a,b]}) \mid [\![p'']\!]_{[a,b]})$$
$$\dot{\sim}$$
$$(b')(b'(x).[\![p]\!]_{[a,b]} \mid (\{a/a\}\{b'/b\}([\![p']\!]_{[a,b]}))) \mid [\![p'']\!]_{[a,b]}$$

Note that if p' has any b-ports they will be renamed to b' and thus be private between $b'(x).[\![p]\!]_{[a,b]}$ and $[\![p']\!]_{[a,b]}$. The translated terms need a sequence of τ-transitions to establish the connection between the "receiving" process and the "copy" it is "receiving". This sequence has the same length as the parameter L. In the above example we needed two τ-transitions and in general we will need as many τ-transitions as the cardinality of the set $fn(p)$.

We now turn to the question of encoding label passing using process passing. This may seem as an artificial question, but as a theoretical result it is of interest since it will provide a basis for discussion of the expressive power of the two approaches.

The idea in the translation below is that instead of sending a channel a we send a wire $(a - chan)$ defined as $i?.a?x.c!.nil + o?.c?x.a!x.nil$. This wire has a multipurpose plug c and a switch to indicate in which direction the wire is to be used. We assume c, i, o are distinct names not used in the Mobile Processes expression being translated. When this wire is received it is plugged into the receiving process by the localizing constructions: $(\ldots[c \mapsto c'][i \mapsto i'][o \mapsto o']\ldots)\backslash c'\backslash i'\backslash o'$. The receiving process will choose in which direction to use this wire by sending an o' signal for output or an i' signal for input. The wire will be private to the sending and receiving processes in the case of a bound name in the Mobile Processes expression. This is ensured by a scope extrusion caused by the static restriction operator.

Mobile Processes [MilParWal89] was developed from ECCS [EngNie86] by simplifying the notions of values, labels and variables into one concept called names. This, however, presents a problem when translating Mobile Processes into Plain CHOCS since a name in a process p may act as a name of a link (as e.g. y in

$y(x).p$) or it may act as a variable (as e.g. x in $y(x).p$) or it may act as a local link name (as e.g. x in $(x).p$). To overcome this difficulty we first translate all free names and all names bound by input prefix into process variables. Then we instantiate the process variables corresponding to free names in the Mobile Processes expression to names in Plain CHOCS. Names bound by restriction will be allocated names in Plain CHOCS in the first translation step.

Definition 5.4.11 $[\![\ ]\!]_1 : MP \rightarrow$ *Plain CHOCS is defined structurally:*

$$
\begin{aligned}
[\![0]\!]_1 &= nil \\
[\![x(y).p]\!]_1 &= (x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid i'!.c'?y.[\![p]\!]_1)\backslash c'\backslash i'\backslash o' \\
[\![\overline{x}y.p]\!]_1 &= (x[c \mapsto c'][i \mapsto i'][o \mapsto o'] \mid o'!.c'!y.[\![p]\!]_1)\backslash c'\backslash i'\backslash o' \\
[\![\tau.p]\!]_1 &= \tau.[\![p]\!]_1 \\
[\![p + p']\!]_1 &= [\![p]\!]_1 + [\![p']\!]_1 \\
[\![p \mid p']\!]_1 &= [\![p]\!]_1 \mid [\![p']\!]_1 \\
[\![(x)(p)]\!]_1 &= ([\![p]\!]_1[(a - chan)/x])\backslash a, \quad \textit{where } a \textit{ has not been used before}
\end{aligned}
$$

$[\![\ ]\!]_2 : MP \rightarrow$ *Plain CHOCS is defined as:*

$$[\![p]\!]_2 = (\ldots([\![p]\!]_1[(a_1 - chan)/x_1])\ldots)[(a_n - chan)/x_n]$$

where $FV([\![p]\!]_1) = \{x_1, \ldots, x_n\}$ *and* $a_1 \ldots a_n$ *are allocated by some* $1-1$ *mapping between* V *and Names (usually established by the* $1-1$ *mapping between* $fn(p)$ *and* $FV([\![p]\!]_1)$*).*

We have omitted the match construct of Mobile Processes. This can be eliminated in the Mobile Processes expression according to [MilParWal89]. Recursion could be translated using the $Y_x[\ ]$ construction from the previous section.

It is easy to see from the above definition that the label passing in the Mobile Processes is mimiced by the translation only requiring two additional communications for each use of the wire, i.e:

$$
\begin{aligned}
[\![a(x).p]\!]_2 &\dot{\sim} \tau.a?x.\tau.[\![p]\!]_2 \\
[\![\overline{a}b.p]\!]_2 &\dot{\sim} \tau.\tau.a!(b - chan).[\![p]\!]_2
\end{aligned}
$$

We may state this more precisely:

Proposition 5.4.12

1. *if* $p \xrightarrow{a(x)} p'$ *then* $[\![p]\!] \xrightarrow{\tau} \xrightarrow{a(x)} \xrightarrow{\tau} [\![p']\!]$
2. *if* $p \xrightarrow{\overline{a}b} p'$ *then* $[\![p]\!] \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{a!_{\emptyset}(b-chan)} [\![p']\!]$
3. *if* $p \xrightarrow{\overline{a}(b)} p'$ *then* $[\![p]\!] \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{a!_{\{b\}}(b-chan)} [\![p']\!]$
4. *if* $p \xrightarrow{\tau} p'$ *then* $[\![p]\!] \xrightarrow{\tau} [\![p']\!]$ *or* $[\![p]\!] \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} [\![p']\!]$

PROOF: By induction on the length of the inference used to establish the transition of p observing the structure of p. □

We conjecture the following relationship between Mobile Processes and their translations into Plain CHOCS:

Conjecture 5.4.13 *if* $p \sim q$ *then* $[\![p]\!]_2 \dot{\approx} [\![q]\!]_2$, *where* $\dot{\approx}$ *is a suitable formulation of weak higher order applicative bisimulation.*

We can not hope for the implication to hold in the opposite direction since the translation may introduce non-determinism not present in the original term e.g: Consider the following term $p = (a)(b)(a(x).c(x).0 + b(x).0 \mid \overline{a}c.0)$ then $p \dot{\sim} \tau.c(x).0 \not\dot{\sim} p + \tau.0$ whereas $[\![p]\!] \dot{\sim} \tau.\tau.\tau.\tau.\tau.\tau.c?x.\tau.nil + \tau.0 \dot{\sim} [\![p + \tau.0]\!]$.

To see how the translation works we study the following small system consisting of two components. Initially the first component is ready to receive a channel on a and the second component is ready to send the b-channel on a. Upon receiving a channel the first component is ready to send a bound channel d on the newly received channel. The second component is ready to receive this channel. The end result is that the second component receives a private d-channel from the first component.

$$[\![a(x).(d)(\overline{x}d.P) \mid \overline{a}b.b(x).Q]\!]$$
$$\dot{\sim}$$
$$\tau.a?x.\tau.[\![(d)(\overline{x}d.P)]\!] \mid \tau.\tau.a!(b-chan).[\![b(x).Q]\!]$$
$$\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}$$
$$([\![(d)(\overline{x}d.P)]\!][(b-chan)/x] \mid [\![b(x).Q]\!])$$
$$\dot{\sim}$$
$$((\tau.\tau.b!(d-chan).[\![P]\!][(b-chan)/x])\backslash d \mid \tau.b?x.\tau.[\![Q]\!])$$
$$\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}$$
$$(([\![P]\!][(b-chan)/x] \mid [\![Q]\!][(d-chan)/x])\backslash d)$$

Comparing the two translations presented in this section we see that the two calculi Mobile Processes and Plain CHOCS are equally expressive in the sense that they may simulate one another. However, the translations are rather ad hoc. It would be of interest if this comparison could be formulated in a more general framework for comparison. In a private communication Chen Liang has told me that he is working on such a generalized framework and he is using the translations between Plain CHOCS and Mobile Processes as an example. His framework is built on a category theoretical characterization of transition systems and the translations are expressed as functors.

5.5 Plain CHOCS Object Oriented Programming

Over the past two decades object oriented programming has grown into a strong discipline in the world of industrial programming. One reason for the success of this programming notion is the link with ideas of structured programming. Object oriented programming allows problems to be broken down into "objects" of manageable size. There is to date no unifying definition of what exactly an object is and what an object does although over the years much effort has been devoted to finding such definitions. It seems as if each object oriented programming language (and even each object oriented programmer) seems to have its (his/her) own definition of an object.

This having been said, there seems to be a consensus that an object is regarded as an encapsulating entity and there are strong analogies to the ideas of abstract data types. Thus objects encapsulate "things" and users access these "things" via "methods" which are the terms used for the diverse access strategies used in object oriented programming. The idea behind the method paradigm is to present the user with an interface through which objects can be accessed and at the same time hide the way the objects are implemented. Most present day object oriented programming languages have roots in ideas presented in the SIMULA language [DahMyhNyg68] designed in the late sixties and ideas presented in the Smalltalk language [GolRob83] have had substantial influence.

The object oriented approach has mostly grown out of an imperative sequential programming discipline as a structuring device for large scale programs, but recently it has been recognized as a useful tool in the description and construction of distributed and concurrent systems [Atk89]. As we shall see in this section there seems to be a strong analogy between the idea of objects and processes, encapsulation and restriction, method call and communication via named channels. We shall also see that it is possible to make connections between concurrency theory and

inheritance, which for many object oriented programmers seems to be a vital part of the definition of what can be characterized as object oriented programming.

Many object oriented programming languages do not have a formal semantics but rely on (thorough) verbal descriptions of the semantics. Recently some more thorough studies of semantics foundations of object oriented programming languages have emerged, POOL [Ame87] and Dragoon [Atk89] are very good examples of how far the current state of affairs for real life programming languages has reached.

In this section we study the connection between concurrency and object oriented programming in more details. We do this via a small toy language O. We may consider O as a prototype core of most imperative concurrent object oriented programming languages. In O we may define a class of objects and instantiate objects to be of a defined class. In each class we may define a number of methods and a thread of control. This thread of control is the primary means for concurrency since objects may be started and executed in parallel. The parallelism is asynchronous and synchronization is obtained by method calls. O was inspired by the toy language P studied in [Mil80] and in section 3.3, and the thread of control in each object is similar to the sequential part of P. The language O is untyped and we only consider type meaningful programs. We assume that objects are declared before they are created, that all objects are created before started and that all objects are started only once.

The semantics of O is described in Plain CHOCS in a phrase-by-phrase style resembling a denotational semantics. However, we do not give any semantic domains. Instead we may view the O semantics as a set of derived operators in Plain CHOCS since the translation carries no parameters. Plain CHOCS only caters for process values in communication. To allow for other values in Plain CHOCS than process values we use the technique of [Mil83] and introduce a $\mathcal{D}$–indexed family of actions $a?_d$, $a!_d$, $d \in \mathcal{D}$ for each value domain $\mathcal{D}$. Due to the fact that only finite sums of processes can be handled in the version of Plain CHOCS presented in this thesis we restrict our attention to finite value domains as e.g. the set of booleans and finite subsets of the integers. We let $\alpha?_x.p$ abbreviate $\Sigma_{d\in D}\alpha?_d.p\{d/x\}$ where $\{d/x\}$ means exchanging all occurrences of x in p by d as e.g. $\alpha?_x.\beta!_x.nil\{d/x\} \equiv \Sigma_{d\in D}\alpha?_d.\beta!_d.nil$. We shall use the following construct from [Mil83]: If b is a boolean valued expression in x then let $\alpha?_x.(if\ b\ then\ p\ else\ p')$ be encoded by $\Sigma_{x\in D\&b}\alpha?_x.p + \Sigma_{x\in D\&\neg b}\alpha?_x.p'$. We should not confuse $\alpha?_x.p$ with $\alpha?x.p$ since the first is a convenient shorthand notation and the latter is part of the Plain CHOCS syntax.

The language O

Programs in O are built from declarations D, expressions E and commands C, using assignments to program variables X. Variables Y always refer to objects and variables Z refer to classes. Some set of functions F is assumed and for the cause of simplicity we do not consider types of expressions. O has the following abstract syntax:

Declarations:	$D ::=$	$\texttt{var}\ X \mid \texttt{obj}\ Y \mid D; D \mid$ $\texttt{method}\ P(\texttt{ref}\ X)\ \texttt{is}\ C \mid \texttt{class}\ Z\ \texttt{is}\ D\ \texttt{body}\ C$
Expressions:	$E ::=$	$X \mid F(E_1, \ldots, E_n)$
Commands:	$C ::=$	$X := E \mid C; C \mid \texttt{if}\ E\ \texttt{then}\ C\ \texttt{else}\ C' \mid$ $\texttt{while}\ E\ \texttt{do}\ C \mid \texttt{skip} \mid \texttt{begin}\ D; C\ \texttt{end} \mid$ $Y.\texttt{create}\ Z \mid Y.\texttt{start} \mid Y.\texttt{call}\ P(X)$

Table 5.5.1: Syntax of O

To give a smooth definition of the semantics of O we need some auxiliary definitions.
To each variable X we associate a register Reg_X. Generally it follows the pattern:

$$Loc = \alpha?_x.Reg(x)$$

$$Reg(y) = \alpha?_x.Reg(x) + \gamma!_y.Reg(y)$$

and thus for X we will have $Loc_X = Loc[\alpha \mapsto \alpha_X][\gamma \mapsto \gamma_X]$. Initially we write in a value, thereupon we can read this value on γ or overwrite the contents of Loc via α. We have written the above definition in an equation style to make it more readable. The proper Plain CHOCS definition is: $Loc = (\alpha?_x.h!_x.nil \mid Reg)\backslash h$ where $Reg = Y_{Reg}[h?_x.(\alpha?_x.h!_x.Reg + \gamma!_x.h!_x.Reg)] \mid Y_{Keep}[h?_x.h!_x.Keep]$. The second component of this process takes care of the parameters in the recursion of the above equations. (This is in fact a general technique for simulating the parameterized recursion of [Mil83]). We also associate a register to each class Z, each object Y and each method P. It may be defined in the same way as above with x substituted with x.

To each n–ary function symbol F we associate a function f which is represented by:

$$b_f = \rho_1?_{x_1}.\ldots.\rho_n?_{x_n}.\rho!_{f(x_1\ldots x_n)}.nil$$

Constants will thus be represented as e.g. $b_{true} = \rho!_{true}.nil$. The result of evaluating an expression is always communicated via ρ. It is therefore useful to define:

$$p\ result\ p' = (p \mid p')\backslash\rho$$

We adopt the protocol of signaling successful termination of commands via δ and it is therefore convenient to define:

$$done = \delta!.nil$$

$$p\ before\ p' = (p[\delta \mapsto \beta] \mid \beta?.p')\backslash\beta \quad , \beta \notin fn(p) \cup fn(p')$$

We now give the semantics of O by the translation into Plain CHOCS shown in table 5.5.2.

Declarations:

$$\begin{aligned}
[\![\texttt{var}\ X]\!] &= Loc_X \\
[\![\texttt{obj}\ Y]\!] &= Loc_Y \\
[\![D; D']\!] &= [\![D]\!] \mid [\![D']\!] \\
[\![\texttt{method}\ P(\texttt{ref}\ X)\ \texttt{is}\ C]\!] &= ((Loc_P \mid \alpha_P!(\ method\ process\ ).nil)\backslash\alpha_P) \\
[\![\texttt{class}\ Z\ \texttt{is}\ D\ \texttt{body}\ C]\!] &= ((Loc_Z \mid \alpha_Z!(\ class\ process\ ).nil)\backslash\alpha_Z)
\end{aligned}$$

where $method\ process = [\![C]\!][\alpha_X \mapsto \alpha_{P_v}][\gamma_X \mapsto \gamma_{P_v}]$
and $class\ process = ([\![D]\!][\alpha_{P_v} \mapsto \alpha^Z_{P_v}][\gamma_{P_v} \mapsto \gamma^Z_{P_v}] \mid [\![C]\!])\backslash V_D$

Expressions:

$$\begin{aligned}
[\![X]\!] &= \gamma_X?_x.\rho!_x.nil \\
[\![F(E_1, \ldots, E_n)]\!] &= ([\![E_1]\!][\rho_1/\rho] \mid \ldots \mid [\![E_n]\!][\rho_n/\rho] \mid b_f)\backslash\rho_1 \ldots \backslash\rho_n
\end{aligned}$$

Commands:

$$\begin{aligned}
[\![X := E]\!] &= [\![E]\!]\ result\ (\rho?_x.\alpha_X!_x.done) \\
[\![C; C']\!] &= [\![C]\!]\ before\ [\![C']\!] \\
[\![\texttt{if}\ E\ \texttt{then}\ C\ \texttt{else}\ C']\!] &= [\![E]\!]\ result\ \rho?_x.(if\ x\ then\ [\![C]\!]\ else\ [\![C']\!]) \\
[\![\texttt{while}\ E\ \texttt{do}\ C]\!] &= Y_w[[\![E]\!]\ result\ \rho?_x.(if\ x\ then\ ([\![C]\!]\ before\ w)\ else\ done)] \\
[\![\texttt{skip}\ ]\!] &= done \\
[\![\texttt{begin}\ D; C\ \texttt{end}]\!] &= ([\![D]\!] \mid [\![C]\!])\backslash L_D \\
[\![Y.\texttt{create}\ Z]\!] &= \gamma_Z?x.\alpha_Y!(x[\alpha^Z_{P_v} \mapsto \alpha^Y_{P_v}][\gamma^Z_{P_v} \mapsto \gamma^Y_{P_v}]).done
\end{aligned}$$

$$
\begin{aligned}
[\![Y.\mathtt{start}\,]\!] &= \gamma_Y?x.(x[\delta \mapsto \beta] \mid \beta?.done)\backslash\beta \\
[\![Y.\mathtt{call}\ P(X)]\!] &= \gamma_P^Y?x.(x[\alpha_{P_v}^Y \mapsto \alpha_X][\gamma_{P_v}^Y \mapsto \gamma_X][\delta \mapsto \beta] \mid \beta?.done)\backslash\beta
\end{aligned}
$$

Table 5.5.2: Semantics of O

In the definition of *class process* we let $\backslash V_D$ abbreviate restrictions with respect to all variables and objects declared in D and in the equation for $[\![\mathtt{begin}\ D; C\ \mathtt{end}]\!]$ we let $\backslash L_D$ abbreviate restriction with respect to α and γ channels for all variables, objects, classes and methods declared in D. The method and class definitions each create a location to store the method process respectively the class process. The restrictions $\backslash\alpha_P$ respectively $\backslash\alpha_Z$ ensure that these processes cannot be overwritten after their definitions.

Note that if we disregard the object oriented part of O we have essentially a language definition similar to the definition of P from section 3.3. However, if we compare the semantic definition of procedures in P with the semantic definition of methods in O we note that the $Transform$ process needed to ensure static binding of variables in the P semantics is no longer present in the O semantics. This is not because we advocate dynamic binding for variables in the object oriented paradigm. It is because the static nature of the restriction operator in Plain CHOCS will ensure that static binding is obtained. The static nature will ensure (by a scope extrusion) that any variable reference is kept with the defining environment. Assignments to variables may be nondeterministic since two or more methods may refer to the same variable and we can have situations where one method reads the value currently stored then another method writes a new value before the first method overwrites the current value. As for the semantic description of P we can avoid this problem by surrounding each variable with a semaphore construct.

A class is defined as a process stored in a register. The class process behaves like a block except that we can invoke the methods defined in the declaration part. These will execute concurrently with the thread defined by the command part of the class process. A class is a passive entity in the sense that it is stored in a register. An object Y of class Z is just a copy of the class process stored in another register. It becomes active when started by the $Y.\mathtt{start}$ command which reads the register and activates the process by the $\gamma_Y?x.(x\ldots)$ construct. Each method is also just a process stored in a register. When a method is called the register

is read and a copy of the method process is activated. The renaming surrounding the variable x ensures a call-by-reference parameter mechanism in the method call. This parameter mechanism seems to be most in line with current trends in object oriented programming, but we can also define call-by-value, call-by-name and lazy parameter mechanisms for method calls in O using the same approach as in the definition of parameter mechanisms in P discussed in section 3.3.

The semantic definition of O has not taken the object oriented paradigm to its extreme where everything is an object. We have kept a distinction between objects, values and methods. We can go a bit further and describe how objects can be passed in method call. To some object oriented programmers this is the true spirit of the object oriented paradigm. Let us see how object passing in method call can be described semantically:

$$[\![\texttt{method } P(\texttt{obj } Y) \texttt{ is } C]\!] = ((Loc_P \mid \alpha_P!(\ \textit{method process}\ ).nil)\backslash\alpha_P)$$

where $\textit{method process} = [\![C]\!][\alpha^Y_{P_v} \mapsto \alpha_{P_v}][\gamma^Y_{P_v} \mapsto \gamma_{P_v}]$

$$[\![Y.\texttt{call } P(Y')]\!] = \gamma^Y_P?x.(x[\alpha_{P_v} \mapsto \alpha^{Y'}_{P_v}][\gamma_{P_v} \mapsto \gamma^{Y'}_{P_v}]\ \textit{before done})$$

Passing an object in a method call works very similar to the call-by-reference parameter mechanism for normal method calls. We simply rename the method calls of the formal parameter to method calls of the actual parameter.

Another phenomenon often connected with object oriented programming languages is the concept of inheritance. This is often considered the main structuring mechanism. We may describe this semantically as follows:

$$[\![\texttt{class } Z \texttt{ inherits } Z' \texttt{ is } D \texttt{ body } C]\!] = ((Loc_Z \mid \alpha_Z!(\ \textit{class process}\ ).nil)\backslash\alpha_P)$$

and $\textit{class process} = (\gamma_{Z'}?x.(x[\alpha^{Z'}_{P_v} \mapsto \alpha^Z_{P_v}][\gamma^{Z'}_{P_v} \mapsto \gamma^Z_{P_v}]\ \textit{before}\ ([\![D]\!][\alpha_{P_v} \mapsto \alpha^Z_{P_v}][\gamma_{P_v} \mapsto \gamma^Z_{P_v}] \mid [\![C]\!])\backslash V_D)$

This describes that the class Z inherits the methods and the thread of control of class Z'. All methods of Z' are renamed to methods of Z and the thread of control of Z' is sequentially composed with that of Z. It is easy to generalize this to multiple inheritance simply by sequentially composing each inheritance class. In some object oriented programming languages programmers are allowed to redefine inherited methods. This is easily obtained by restricting the α and γ channels of the redefined method from the inheritance class and redefining it in the declaration part of the class.

This section represents a small step towards a semantic description of object oriented programming in Plain CHOCS. Except for a few syntactic differences the core

of POOL [Ame87] is very similar to O. There are many interesting and challenging aspects in investigating a comparison of the semantics of POOL with the semantics of O more thoroughly and perhaps establishing a translation of POOL into Plain CHOCS and thus provide a basis for a formal comparison. These prospects are left for future studies.

Chapter 6

Conclusion

The aim of this thesis has been to provide a thorough investigation into the foundations of calculi for higher order communicating systems. I have aimed at putting this study on the same kind of principles as the studies of functions using the λ–Calculus. The achievements reported in this thesis do of course not compare to those achieved for the λ–Calculus but they are a first step towards exploring the expressive power of calculi for higher order communicating systems. I have tried to follow the path of simplicity and minimality and I have tried to carefully select a small set of operators (as small as possible in fact) which can be used to express complicated and sophisticated operators. This provides a minimal syntax (in the sense of [Mil86]) for the calculus and I have investigated several semantic models.

The first part of this thesis described the CHOCS calculus with an operational semantics given as an extension of the operational semantics for CCS with value passing. We have shown how the fundamental notions of bisimulation and observational equivalence may be extended to take processes sent and received in communication into account. We have shown that these equivalences satisfy almost the same set of algebraic laws known from CCS only needing some obvious new laws for process communication. As an interesting point to note we have shown how process communication may be used to simulate recursive behaviour. We have also shown how to define a denotational semantics for CHOCS and we have shown that this semantics, with mild restrictions, is fully abstract with respect to the operational semantics. These restrictions are due to the well known impossibility of modelling unbounded nondeterminism in the Plotkin Power Domain. It is a challenging task to see if the Plotkin Power Domain for countable nondeterminism [Plo82] could be used to resolve this problem for a denotational semantics for CHOCS.

6.1 Loose Ends

The study of calculi for higher order communicating systems presented in this thesis has touched on most of the usual subjects of process calculus and successfully extended these results to take processes sent and received in communication into account. Only one major study has been "neglected": most process calculi study the subject of process logic. One outstanding representative for this approach is the so-called Hennessy-Milner-Logic (HML) for CCS [HenMil85]. So far we have not presented any results about process logic, but HML may be extended to a version relevant for CHOCS. This logic takes processes sent and received in communication into account and we therefore introduce binary modal operators which correspond to the unary modal operators of HML.

Let the language $\mathcal{L}$ of formulae be the least set such that:

1. $T \in \mathcal{L}$
2. $F, F' \in \mathcal{L} \Rightarrow \langle a? \rangle (F, F'),\ \langle a! \rangle (F, F'),\ F \wedge F' \in \mathcal{L}$
3. $F \in \mathcal{L} \Rightarrow \langle \tau \rangle F,\ \neg F \in \mathcal{L}$

The satisfaction relation $\models \subseteq \mathcal{P} \times \mathcal{L}$ is the least relation such that:

1. $p \models T$ for all $p \in Pr$
2. $p \models F \wedge F'$ iff $p \models F$ and $p \models F'$
3. $p \models \neg F$ iff not $p \models F$
4. $p \models \langle a? \rangle (F, F')$ iff for some p', p'', $p \xrightarrow{a?p'} p''$ and $p' \models F$ and $p'' \models F'$
5. $p \models \langle a! \rangle (F, F')$ iff for some p', p'', $p \xrightarrow{a!p'} p''$ and $p' \models F$ and $p'' \models F'$
6. $p \models \langle \tau \rangle F$ iff for some p'', $p \xrightarrow{\tau} p''$ and $p'' \models F'$

Theorem 6.1.1 *If $\mathcal{P}$ is image finite then*

$$p \sim q \ \textit{iff}\ \mathcal{L}(p) = \mathcal{L}(q)$$

where $\mathcal{L}(p) = \{F \ : \ p \models F\}$.

PROOF: The proof of this theorem follows the corresponding proof for HML given in [HenMil85]. □

It is an interesting subject for further studies to see if this extension of HML can be reconstructed using the domain logic framework of [Abr87a] along the lines of the reconstruction of HML in [Abr90a]. From the domain equation: $D \cong P^0[\sum_{e \in Ev} D_e]$, where $D_{a?} \equiv D \times D$, $D_{a!} \equiv D \times D$ and $D_\tau \equiv D$, defined in Chapter 4, we may generate a domain logic using the framework based on Stone Duality presented in [Abr87a]. Once this study is completed it is a natural next step to investigate if and how this domain logic and the denotational semantics may be used to give a compositional proof system for CHOCS along the lines of [Sti87] and [Win85].

6.2 Technical Choices and Open Questions

Throughout this thesis several technical choices have been made during the development of the theory; often these choices have been motivated by technical necessity to make the theory work or they have been made from "gut" feeling about the intuition behind the theory. But wherever choices are made alternatives exist. I have tried to list the most obvious ones and explain their implications at the appropriate point in the text but one choice has been bigger and deserves more attention and may turn out to be a worth-while path for future studies.

The choice relates to the question of the observational theory for CHOCS. This theory is based on the definition of $p \stackrel{\Gamma}{\Longrightarrow} p''$ as $p \stackrel{\tau}{\longrightarrow}^* \stackrel{\Gamma}{\longrightarrow} p''$ motivated by technical necessity. I have not been able to prove congruence properties about weak higher order bisimulation with the more common definition of $p \stackrel{\Gamma}{\Longrightarrow} p''$ as $p \stackrel{\tau}{\longrightarrow}^* \stackrel{\Gamma}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* p''$. Of course this does not imply that it is not possible to do so and philosophically there should be no reason for not doing so. This is not the case for an observational theory for Plain CHOCS however. For technical reasons we would have to define $\stackrel{a?x}{\Longrightarrow} \equiv \stackrel{\tau}{\longrightarrow}^* \stackrel{a?x}{\longrightarrow}$ since $\stackrel{a?x}{\longrightarrow} \subseteq Pr \times [Pr \to Pr]$ and it would not make sense to write $\stackrel{a?x}{\Longrightarrow} \equiv \stackrel{\tau}{\longrightarrow}^* \stackrel{a?x}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^*$. We could define the output transitions as $\stackrel{a!p'}{\Longrightarrow} \equiv \stackrel{\tau}{\longrightarrow}^* \stackrel{a!p'}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^*$ but this would introduce an unnecessary asymmetry. In general the formulation of an observational theory for Plain CHOCS is an open and interesting question.

We have not solved the problem of finding an alternative description of the largest congruence containing higher order observational equivalence. What we have shown is that we can define an irrefiexive weak higher order bisimulation predicate and prove that this equivalence is a congruence. There are a few suggestions to how to proceed. One could define a recursive bisimulation-like version of the congruence:

Definition 6.2.1 *A* weak higher order context bisimulation *R is a binary relation on Pr such that whenever pRq and $\Gamma \in Act$ and C is a context then:*

(i) Whenever $C[p] \stackrel{\Gamma}{\Longrightarrow} p'$, then $C[q] \stackrel{\Gamma'}{\Longrightarrow} q'$ for some q', Γ' with $\tilde{\Gamma} \hat{\hat{R}} \tilde{\Gamma'}$ and $p' R q'$

(ii) Whenever $C[q] \stackrel{\Gamma}{\Longrightarrow} q'$, *then* $C[p] \stackrel{\Gamma'}{\Longrightarrow} p'$ *for some* p', Γ' *with* $\tilde{\Gamma}'\hat{\hat{R}}\tilde{\Gamma}$ *and* $p'Rq'$

If there exists a weak higher order context bisimulation R *containing* (p,q) *we write* $p \approx^C q$.

It is relatively easy to see that the relation $\approx^C$ is a congruence relation containing $\approx$. However, $\approx^C$ also contains the following relation:

Definition 6.2.2 *A* weak higher order plus bisimulation R *is a binary relation on* Pr *such that whenever* pRq *and* $\Gamma \in Act$ *and* $r \in Pr$ *then:*

(i) Whenever $p + r \stackrel{\Gamma}{\Longrightarrow} p'$, *then* $q + r \stackrel{\Gamma'}{\Longrightarrow} q'$ *for some* q', Γ' *with* $\tilde{\Gamma}\hat{\hat{R}}\tilde{\Gamma'}$ *and* $p'Rq'$

(ii) Whenever $q + r \stackrel{\Gamma}{\Longrightarrow} q'$, *then* $p + r \stackrel{\Gamma'}{\Longrightarrow} p'$ *for some* p', Γ' *with* $\tilde{\Gamma}'\hat{\hat{R}}\tilde{\Gamma}$ *and* $p'Rq'$

If there exists a weak higher order plus bisimulation R *containing* (p,q) *we write* $p \approx^+ q$.

This relation generalizes the usual definition of $\approx^+$ to a recursively defined predicate on Pr^2. Unfortunately this immediately refutes the law $pre.\tau.p \approx^c pre.p$, where pre is any of the prefixes $a?x$, $a!p'$ or τ and thus leaves open the question of the validity of this law for $\approx^c$.

It is interesting to see if the approach of denotational semantics may help answering this question. It seems possible to define a denotational semantics which is fully abstract with respect to an operational semantics built on the notion of irreflexive higher order bisimulation. Abramsky has recently discovered how this can be used in the context of SCCS to get to full abstraction with respect to the observational congruence by "factoring" out the denotational semantics by the equation: $pre.\tau.p \approx^c pre.p$ [Abr90b].

The second part of this thesis has provided an alternative operational semantics for the calculus of higher order communicating systems. We have called this study Plain CHOCS since it exhibits a static nature for the operators of restriction and renaming. The study of Plain CHOCS is still in a rather preliminary stage. There are various tasks to be continued. As mentioned above there is the immediate task of establishing an observational theory for Plain CHOCS. Another major challenge is to establish a denotational theory. The operational modelling of input suggests that input should be modelled by function space $D \rightarrow D$, but the behaviour is

also dependent on the set of bound names exported in scope extrusion, thus one suggestion for a denotation domain worth investigating is:

$$D \cong P^0[\Sigma_{a \in Names}[Names \to D] \to D + \Sigma_{a \in Names} Names \times D \times D + D]$$

In the context of Plain CHOCS there is a very interesting variant of the applicative higher order bisimulation which deserves to be explored:

Definition 6.2.3 *A* variant applicative higher order simulation R *is a binary relation on CPr such that whenever* pRq *and* $a \in Names$ *then:*

(i) Whenever $p \xrightarrow{a?x} p'$, *then for all* $r \in CPr$ *there is some* q', y *such that* $q \xrightarrow{a?y} q'$ *and* $p'[r/x]Rq'[r/y]$

(ii) Whenever $p \xrightarrow{a!p'_B} p''$, *then* $q \xrightarrow{a!q'_B} q''$ *for some* q', q'' *with* $B \cap (fn(p) \cup fn(q)) = \emptyset$ *and* $p'Rq'$ *and* $p''Rq''$

(iii) Whenever $p \xrightarrow{\tau} p'$, *then* $q \xrightarrow{\tau} q'$ *for some* q' *with* $p'Rq'$

A relation R *is a* variant applicative higher order bisimulation *if both it and its inverse are applicative higher order simulations.*
If there exists a variant applicative higher order bisimulation R *containing* (p, q) *we write* $p \stackrel{\cdot:'}{\sim} q$.

This relation is obtained by commuting the quantifiers in the first clause of definition 5.2.1. The relation $\stackrel{\cdot:'}{\sim}$ is interesting since it is stronger than the applicative higher order bisimulation relation. A similar variation of strong ground bisimulation was suggested in [MilParWal89] and it was shown that in the context of Mobile Processes the variant relation is strictly stronger. It is an open question if the inclusion is strict in the context of Plain CHOCS.

6.3 Applications

The calculi studied in this thesis are idealized cores for the study of process passing in communicating systems and we have shown how other values, including higher order functions, sequential programs and objects may be encoded. Several examples throughout this thesis have shown that there are systems which naturally may be described in terms of communicating systems passing processes in communication. Even so it is my opinion that we need other types of values as primitive constructs in the language for real programming languages and large scale specifications.

Both the formalisms of LOTOS [BolBri87] and SMoLCS [AstReg87] have non-process values as algebraic data types incorporated in their specification languages. This seems to be an appropriate way of specifying communicable values from the point of algebraic reasoning about values, but it offers little from the point of an operational description of their behaviour. Therefore in my opinion non-process values should be introduced together with an operational semantics and an appropriate integration of equivalences should be studied which could form a basis for algebraic reasoning along the ideas of laws presented in this thesis. These ideas have partly been pursued in TPL [Nie89] and more thoroughly studied in FACILE [GiaMisPra90]. It is my hope that the calculi studied in this thesis may serve as a foundational tool for future multi-paradigm programming languages. Already both TPL and FACILE use foundational models similar to the higher order communication trees studied in detail in the first part of this thesis.

The introduction of non-process values calls for a notion of types. Both FACILE and TPL are equipped with type structures, but in my opinion the FACILE type structure which assigns the type code to process values is too restrictive since there is no reference to the sort of the process, and the type structure of TPL which has no distinction between process expressions and other expressions is too flexible. A suggestion for a type structure is a merge between the FACILE type structure and the sort system described in section 2.4 which should be pursued in the future.

Design of a new multi-paradigm language with non-process values and processes as communicable values seems to be a major challenge. The theory presented in this thesis is hopefully a useful tool, but for real life programming languages there is also the inevitable question of how to implement them on real computers. The translation of Plain CHOCS into Mobile Processes presented in section 5.4 gives some ideas about how process passing could be implemented on a lower level using processes on dynamically reconfigurable networks. This again calls for a study of an implementation strategy for such processes. Some hints are given in [Mil90] where a Chemical Abstract Machine [BerBou90] for Mobile Processes is considered. A more thorough study of implementations of processes on dynamically reconfigurable networks will appear in [Let91]. Another suggestion is to use the framework of Chemical Abstract Machines directly as demonstrated for the γ-Calculus in [BerBou90]. All the above approaches towards implementations are relatively abstract. To get closer to a real implementation I think it is necessary to consider implementations in an occam-like[1] language on a Transputer-like machine architecture [INM88]. This is one of the tasks that the ProCos Project under the ESPRIT Basic Research Action 3104 [Bjø89] aims at doing for multi-paradigm languages

[1] occam is a trademark of INMOS Limited.

based on a CSP/occam/Transputer approach. However, occam does not seem to be quite adequate for implementing a multi-paradigm programming language built on the ideas of CHOCS because it does not allow one to describe dynamically reconfigurable networks; such networks can only be simulated using large static networks. Another promising machine architecture is the Alice machine [DarRee81]. This machine has a network of processors linked via a kind of telephone exchange mechanism which allows physical connections between processors to be changed dynamically. One problem I foresee with implementations on either a Transputer or an Alice machine architecture is that they both seem to require large blocks of sequential sub-computations to efficiently utilize the computing power of each processor in the network. However, the translation of Plain CHOCS into Mobile Processes and more generally the results on simulation of functional languages reported in [Let91] indicate that we should be looking for a machine architecture where there are very small blocks of sequential sub-computations and where the basic computation is reconfiguring the network. It will be a major challenge to see how this can be realized in practice.

Bibliography

[Abr87] S. Abramsky: *Observation Equivalence as a Testing Equivalence*, Theoretical Computer Science 53, pp. 225-241, North-Holland, 1987.

[Abr87a] S. Abramsky: *Domain Theory in Logical Form*, Proceedings of LICS 87, 1987. Full version to appear in Annals of Pure and Applied Logic.

[Abr90] S. Abramsky: *The Lazy Lambda Calculus*, Chapter 4 in D. Turner (ed.), Research Topics in Functional Programming, pp. 65-116, Addison Wesley, 1990.

[Abr90a] S. Abramsky: *A Domain Equation For Bisimulation*, (to appear in Information and Computation 1990), Department of Computing, Imperial College, London University, 1987.

[Abr90b] S. Abramsky: *Causal Semantics in Process Algebra*, Draft, Department of Computing, Imperial College, London University, 1990.

[Ace89] L. Aceto: *On Relating Concurrency and Nondeterminism*, Report No. 6/89, Dept. Computer Science, University of Sussex, 1989.

[Acz84] P. Aczel: *A Simple Version of SCCS and Its Semantics*, Unpublished notes, Edinburgh 1984.

[Ame87] P. America: *POOL-T: A Parallel Object–Oriented Language*, Proceedings of Object Oriented Concurrent Programming, pp. 199-220, MIT Press, 1987.

[Ast89] E. Astesiano: *Open Letter to Mr. Bent Thomsen, author of the POPL'89 paper: "A Calculus of Higher Order Communicating Systems"*, Email, 1989.

[AstGioReg88] E. Astesiano, A. Giovini, G. Reggio: *Generalized Bisimulation in Relational Specifications*, Proceedings of STACS 88, Lecture Notes in Computer Science 294, pp. 207-226, Springer-Verlag, 1988.

[AstReg87] E. Astesiano & G. Reggio: *SMoLS-Driven Concurrent Calculi*, Proceedings of TAPSOFT 87, Lecture Notes in Computer Science 249, pp. 169-201, Springer-Verlag, 1987.

[AstReg87b] E. Astesiano & G. Reggio: *Direct Semantics for Concurrent Languages in the SMoLCS approach*, IBM Journal of Research and Development, vol. 31, no. 5, pp. 512-534, 1987.

[Atk89] C. Atkinson: *An Object–Oriented Language for Software Reuse and Distribution*, Ph. D. Thesis, Department of Computing, Imperial College, London University, 1989.

[AusBou84] D. Austry & G. Boudol: *Algèbre de Processus et Synchronisation*, Theoretical Computer Science 30(1), pp. 91–131, North-Holland, 1984.

[Bar84] H. P. Barendregt: *The Lambda Calculus — Its Syntax and Semantics*, North–Holland, 1984.

[BerBou90] G. Berry & G. Boudol: *The Chemical Abstract Machine*, Proceedings of POPL 90, pp. 81-94, The Association for Computing Machinery, 1990.

[BerKlo84] J. Bergstra & J. W. Klop: *Process Algebra for Synchronous Communication*, Information and Control, vol. 60, pp. 109–137, 1984.

[Bjø89] D. Bjørner: *A ProCos Project Description ESPRIT BRA 3104*, Bulletin of the EATCS number 39, pp. 60-73, 1989.

[BolBri87] T. Bolognesi & E. Brinksma: *Introduction to the ISO Specification Language LOTOS*, in Computer Networks and ISDN Systems 14, pp. 25-59, North-Holland, 1987.

[Bou89] G. Boudol: *Towards a Lambda–Calculus for Concurrent and Communicating Systems*, Proceedings of TAPSOFT 89, Lecture Notes in Computer Science 351, pp. 149-161, Springer-Verlag, 1989.
Preliminary version in Research Report no. 885, INRIA Sophia Antipolis, Autumn 1988.

[BouCas87] G. Boudol & I. Castellani: *On the Semantics of Concurrency: Partial Orders and Transition Systems*, Proceedings of TAPSOFT 87, Lecture Notes in Computer Science 249, pp. 122-137, Springer-Verlag, 1987.

[BouLar89] G. Boudol & K. G. Larsen: *Graphical Versus Logical Specifications*, Technical report R 89-33, University of Aalborg, 1989.

[Chr88] P. Christensen: *The Domain of CSP Processes*, incomplete draft, The Technical University of Denmark, 1988.

[CleParSte89] R. Cleveland, J. Parrow & B. Steffen: *The Concurrent Workbench: a semantics based tool for the verification of concurrent systems*, report ECS-LFCS-89-83, University of Edinburgh, 1989.

[CouCou79] P. Cousot & R. Cousot: *Systematic design of Program Analysis Frameworks*, In Conf. Record of the 6th ACM symposium on Principles of Programming Languages, 1979.

[Coz90] J. Cozens: *Adaptable Computer Systems*, incomplete draft, University of Surrey, 1990.

[DahMyhNyg68] O. J. Dahl, B. Myhrhaug & K. Nygaard: *SIMULA 67 Common Base Language*, Norwegian Computing Center, 1968.

[DarRee81] J. Darlington, M. Reeve: *Alice – A Multiprocessor Reduction Machine for the Parallel Evaluation of Applicative Languages*, in Proceedings of the 1981 ACM Symposium on Functional Languages and Computer Architecture, pp. 65-76, 1981.

[EngNie86] U. Engberg & M. Nielsen: *A Calculus of Communicating Systems with Label Passing*, Technical report DAIMI PB-208, Computer Science Department, Aarhus University, 1986.

[GiaMisPra89] A. Giacalone, P. Mishra & S. Prasad: *FACILE, A Symmetric Integration of Concurrent and Functional Programming*, In J. Diaz & F. Orejas (eds.), Proceedings of TAPSOFT 89, pp. 184-209, Lecture Notes in Computer Science 352, Springer-Verlag, 1989.

[GiaMisPra90] A. Giacalone, P. Mishra & S. Prasad: *Operational and Algebraic Semantics for Facile: A Symmetric Integration of Concurrent and Functional Programming*, Proceedings of ICALP 90, pp. 765-780, Lecture Notes in Computer Science 443, Springer-Verlag, 1990.

[GodLarZee89] J. C. Godskesen, K. G. Larsen & M. Zeeberg: *TAV (Tools for Automatic Verification) users manual*, report R 89-19, University of Aalborg, 1989.

[GolRob83] A. Goldberg & D. Robson: *Smalltalk 80: The Language and its Implementation*, Addison Wesley, 1983.

[Gog etal77] J. A. Goguen, J. W. Thatcher, E. G. Wagner & J. B. Wright: *Initial algebra semantics and continuous algebras*, Journal of the ACM, vol. 24, pp. 68-95, 1977.

[GraSif86] S. Graf & J. Sifakis: *A Logic for the Description of Non-deterministic Programs and Their Properties*, Information and Control, vol. 68 pp. 254-270, 1986.

[GroVaa89] J. F. Groote & F. Vaandrager: *Structured Operational Semantics and Bisimulation as a Congruence*, Draft, Centre for Mathematics and Computer Science, Amsterdam, 1989.

[HenMil85] M. Hennessy & R. Milner: *Algebraic Laws for Nondeterminism and Concurrency*, Journal of the Association for Computing Machinery, vol. 32, No. 1, pp. 137-161, 1985.

[Hen81] M. Hennessy: *A term model for synchronous processes*, Information and Control, vol. 51(1), pp. 58-75, 1981.

[Hen84] M. Hennessy: *Axiomatising Finite Delay Operators*, Acta Informatica 21, pp. 61-88, Springer-Verlag, 1984.

[Hen88] M. Hennessy: *An Algebraic Theory of Processes*, MIT Press, Cambridge Massachusetts, 1988.

[HenNic87] M. Hennessy & R. de Nicola: *CCS without τ's*, Lecture Notes in Computer Science 249, pp. 138-152, Springer-Verlag, 1987.

[HenPlo79] M. Hennessy & G. Plotkin: *A term model for CCS*, Proceedings of MFCS 79, Lecture Notes in Computer Science 74, Springer-Verlag, 1979.

[HenPlo80] M. Hennessy & G. Plotkin: *A term model for CCS*, Proceedings of MFCS 80, Lecture Notes in Computer Science 88, pp. 261-274, Springer-Verlag, 1980.

[Hoa81] C. A. R. Hoare: *A Model for Communicating Sequential Processes*, Technical monograph, Computer Laboratory, University of Oxford, 1981.

[Hoa85] C. A. R. Hoare: *Communicating Sequential Processes*, Prentice Hall, 1985.

[INM88] INMOS Limited: occam *2 Reference Manual*, Prentice Hall, 1988.

[KenSle83] J. R. Kennaway & M. R. Sleep: *Syntax and Informal Semantics of DyNe, a Parallel language*, Proceedings of workshop on The analysis of Concurrent Systems 1983, Lecture Notes in Computer Science 207, pp. 222-230, Springer-Verlag, 1985.

[KenSle88] J. R. Kennaway & M. R. Sleep: *A Denotational Semantics for First Class Processes*, Draft, School of Information Systems, University of East Anglia, Norwich, 1988.

[Lar 86] K. G. Larsen: *Context Dependent Bisimulation Between Processes*, Ph. D. Thesis, Edinburgh University 1986.

[Lar87] K. G. Larsen: *From Modal Logic to Process Algebra*, Draft, University of Aalborg, 1987.

[LarTho88] K. G. Larsen & B. Thomsen: *Compositional Proofs by Partial Specifications of Processes*, Proceedings of MFCS 88, Lecture Notes in Computer Science 324, pp. 414-423, Springer-Verlag, 1988.

[LarTho88b] K. G. Larsen & B. Thomsen: *A Modal Process Logic*, Proceedings of LICS 88, pp. 203-210, Computer Society Press, 1988.

[LecMadVer88] V. Lecompte, E. Madelaine & D. Vergamini: *Auto: a verification system for parallel and communicating processes*, INRIA Sophia Antipolis, 1988.

[Let91] L. Leth: *Functional Programs as Reconfigurable Networks of Communicating Processes*, Forth coming Ph. D. Thesis, Department of Computing, Imperial College, University of London, 1991.

[Maz77] A. Mazurkiewicz: *Concurrent Program Schemes and Their Interpretation*, Proceedings of Aarhus Workshop on Verification of Parallel Processes, Aarhus University, 1977.

[Mil80] R. Milner: *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[Mil81] R. Milner: *A Complete Inference System for a Class of Regular Behaviours*, University of Edinburgh, 1981.

[Mil81b] R. Milner: *Modal characterisation of observable machine behaviour*, Proceedings of CAAP 81, Lecture Notes in Computer Science 112, pp. 25-34, Springer-Verlag, 1981.

[Mil83] R. Milner: *Calculi for Synchrony and Asynchrony*, Theoretical Computer Science 25, pp. 267-310, North Holland, 1983.

[Mil86] R. Milner: *Process Constructors and Interpretations*, in H. J. Kugler (ed.), Information Processing 86, pp. 507-518, Elsevier Science Publishers B. V. (North-Holland), IFIP, 1986.

[Mil89] R. Milner: *Communication and Concurrency*, Prentice Hall, 1989.

[Mil90] R. Milner: *Functions as Processes*, Proceedings of ICALP 90, pp. 167-180, Lecture Notes in Computer Science 443, Springer-Verlag, 1990.

[MilParWal89] R. Milner, J. Parrow & D. Walker: *A Calculus of Mobile Processes, Part I*, report ECS-LFCS-89-85, University of Edinburgh, 1989.

[MilParWal89b] R. Milner, J. Parrow & D. Walker: *A Calculus of Mobile Processes, Part II*, report ECS-LFCS-89-86, University of Edinburgh, 1989.

[Nie84] F. Nielson: *Abstract Interpretation using Domain Theory*, Ph. D. Thesis, Edinburgh University, 1986.

[Nie89] F. Nielson: *The Typed λ-Calculus with First-Class Processes*, Proceedings of PARLE 89, Lecture Notes in Computer Science 366, Springer-Verlag, 1989. Preliminary version: Technical Report ID-TR: 1988-43 ISSN 0902-2821, Department of Computer Science, Technical University of Denmark, August 1988.

[Ong88] C-H. L. Ong: *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*, Ph. D. Thesis, Department of Computing, Imperial College, London University, 1988.

[Par81] D. Park: *Concurrency and Automata on Infinite Sequences*, Theoretical Computer Science VII, Lecture Notes in Computer Science 104, Springer-Verlag, 1981.

[Plo76] G. Plotkin: *A Powerdomain Construction*, SIAM Journal on Computing, 5 (1976), pp. 452-487, 1976.

[Plo81] G. Plotkin: *A Structural Approach to Operational Semantics*, Technical report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Plo81b] G. Plotkin: *Post–graduate notes in advanced domain theory (incorporating the "Pisa Notes")*, Department of Computer Science, University of Edinburgh, 1981.

[Plo82] G. Plotkin: *A Powerdomain for Countable Non–determinism*, Proceedings of Automata, Language and Programmming, Lecture Notes in Computer Science 140, pp. 360-372, Springer-Verlag, 1982.

[Pra88] K. V. S. Prasad: *Combinators and Bisimulation Proofs for Restartable Systems*, Ph. D Thesis, Edinburgh University, 1988.

[Rei85] M. Reisig: *Petri Nets*, EATCS Monographs on Theoretical Computer Science 4, Springer-Verlag, 1985.

[Sch86] D. A. Schmidt: *Denotational Semantics: A Methodology for Language Development*, Allyn and Bacon, Inc., 1986.

[Sim85] R. de Simone: *Higher–level Synchronising Devices in MEIJE–SCCS*, Theoretical Computer Science 37, pp. 245-267, North-Holland, 1985.

[Sti87] C. Stirling: *Modal logics for communicating systems*, Theoretical Computer Science 49, pp. 311-347, North-Holland, 1987.

[Tar55] A. Tarski: *A Lattice-Theoretical Fixpoint Theorem and Its Applications*, Pacific Journal of Math. 5, 1955.

[Tho87] B. Thomsen: *An Extended Bisimulation Induced by a Preorder on Actions*, M. Sc. Thesis, University of Aalborg, 1987.

[Tho89] B. Thomsen: *A Calculus of Higher Order Communicating Systems*, Proceedings of POPL 89, pp. 143-154, The Association for Computing Machinery, 1989.

[Tho89b] B. Thomsen: *Plain CHOCS*, Technical report 89/4, Department of Computing, Imperial College, London University, 1989.

[Wal88] D. Walker: *Bisimulation and Divergence*, Proceedings of LICS 88, pp. 186-192, Computer Society Press, 1988.

[Win80] G. Winskel, *Events in Computation*, Ph. D. Thesis, University of Edinburgh, Scotland, 1980.

[Win85] G. Winskel, *A complete proof system for SCCS with modal assertions*, Proceedings of Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 206, Springer-Verlag, pp. 392-410, 1985.

Index