PhD Thesis

Process Algebraic Theories for Mobile Systems

Michele Boreale

Dottorato di Informatica - VII ciclo
Dipartimento di Scienze dell'Informazione
Università di Roma "La Sapienza"

November 1994

Abstract

The thesis is concerned with the descriptions, by algebraic means, of *mobile systems*, i.e. concurrent systems with a dynamically changing communication topology. In particular, we consider the language *π-calculus*, where mobility is achieved by allowing channel names (also called simply *names*) to be transmitted among processes.

We aim at generalizing results for static process algebras (such as CCS) to the name-passing setting, in particular to the π-calculus. The results concern semantic theories and proof techniques to reason about systems descriptions. Most semantics proposed for static process algebra are based on the concepts of *bisimulation* or *testing*; another major division, somewhat orthogonal to the former, is between *interleaving* and *non-interleaving* semantics. The reasoning is mainly concerned with proving (or disproving) that two descriptions are semantically related.

We will firstly consider a testing-based semantics for the π-calculus. For this semantics, we work out a sound and complete proof system and a fully abstract denotational model. In the actual development of these results, it turns out to be crucial the discriminating power provided by an operator originally not considered in the calculus, that permits negative tests on identity of names. We assess the importance of this operator by proving that its omission leads to a different, unnatural testing theory.

We then consider the inefficiency problems that arise with the standard definition of bisimulation equivalence for the π-calculus. A substantial part of this inefficiency is due to name-passing, in that a case-analysis on the received names is needed on input actions. We consider a notion of *symbolic* bisimulation, by which non-essential case-analysis can be avoided, thus leading to an efficiency gain; then, we prove that this semantics can be used to establish the usual bisimulation equivalences. A sound and complete proof system is also proposed to reason about the symbolic equivalence.

Finally, we consider a notion of non-interleaving *causal* equivalence for the π-calculus that discriminates also on the basis of the causal dependencies among actions of processes. In order to properly define this semantics, we study the meaning of causality in a name-passing setting and single out two forms of dependencies: *subject* dependencies, due to the structure of processes, and *object* dependencies, due to the binding mechanism on names. We then study the relationship between our causal equivalence and a standard interleaving equivalence; we show that the former equivalence can be faithfully represented in the latter. This is mainly a result of expressiveness for the calculus. We also show how the encoding can be used to reason about causality by exploiting the simpler mathematical theory of interleaving.

Acknowledgments

I met my advisor Rocco De Nicola for the first time when I was an undergraduate student in Pisa. Since then he has been a constant source of encouragement, teaching me everything I know on the way of doing research. For all this and for his friendship I am sincerely grateful to him.

Davide Sangiorgi is the other person with whom I am most in debt on the scientific side. My understanding of the π-calculus has surely sharpened thanks to my interaction with him. Part of the work presented in this thesis arises from this collaboration.

I want also to thank all my friends and colleagues in Rome, at the Computer Science Department of University "La Sapienza", and in Pisa. I shared with them many enjoyable moments that made the labour of writing this thesis more tolerable.

Prof. Corrado Böhm and Prof. Maurizio Lenzerini were the attentive members of the internal thesis committee. Matthew Hennessy and David Walker were the external reviewers: I am grateful to them for having read so carefully a previous version of the thesis and for the suggestions they gave me.

People at Istituto di Elaborazione dell'Informazione - C.N.R. in Pisa have often kindly tolerated my somewhat "extraneous" presence and provided me with technical support.

A special thought for Marida, with whom I have shared so much during the last five years, for her love and patience.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A lot of research in Theoretical Computer Science has been done in the past years to better understand concurrent systems. Several models and languages have been proposed to formally analyze and describe such systems. Among these, the *Process Algebra* approach, introduced by Milner in 1980 with his work on the Calculus of Communicating Systems (CCS) [Mil80, Mil89], is one of the most successful. Other seminal results in the area have been achieved by Hoare [Hoa85], by Bergstra and Klop [BK89], by Hennessy [Hen88] and by Baeten and Weijland [BW90].

Within a process algebra, description and analysis of concurrent systems are carried out in an algebraic fashion. A small set of operators is provided, each of which corresponds to a primitive way of building a system from subcomponents. Sequenzialization and parallel composition are typical examples of such operators. A process algebra permits describing a system at different levels of abstraction. Generally, an abstract description may be viewed as a *specification* of the wanted system; a more detailed one, where also information about the logical/physical structure of the system is given, may be viewed as the description of an *implementation*. The main concern is how to relate systems descriptions at different levels: this is clearly a matter of semantics. A common approach is to equip the language with a suitable *behavioural relation*. Typically, this takes the form of an equivalence (saying when two descriptions represents the "same" system), or of a preorder (saying, for example, when one system can be considered a good "approximation" of another one). This permits facing, on a formal basis, tasks of practical importance, such as *verification* (checking whether a specification and a given implementation are correctly related) and *stepwise development* (building an implementation starting from a specification by substituting, in a stepwise fashion, each component with a more detailed, but related, one). However, there is no general agreement on what a "suitable" relation should be; therefore a plenty of different approaches to process algebras semantics have been and are being studied. An overview of these will be given in the next section.

A limitation of the "traditional" process algebras such as CCS [Mil89], ACP [BK89], TCSP [Hoa85], is that only pure synchronization among processes can be modeled: with these languages it is impossible to directly express the transmission of a value from one process to another (*value-passing*). Although it is quite straightforward to extend the syntax of a pure process algebra to permit value-passing (this had already been done, e.g., in [Mil80]), the development of a suitable semantic theory for it has turned out to be highly non-trivial. Indeed, only recently a complete semantic analysis for a value-passing language has been presented by Hennessy and Ingolfsdottir [HI91]. A further, and more decisive, step is to include in the language *name-passing*, i.e. to allow processes to exchange communication channels, also called *names*. This permits the description of systems with a dynamically changing communication topology (mobile systems). One of the first attempts to describe mobility in an algebraic setting was made by Engberg an Nielsen, who put forward the language ECCS [EN86]. But, by now, the prototypal name-passing calculus is considered the π-calculus, proposed in 1989 by Milner, Parrow and Walker [MPW92] as a substantially cleaned-up version of ECCS. A rather different approach to mobility in process algebra has been also proposed by Thomsen with the language CHOCS [Tho89]: there *process-passing* is considered as a primitive notion. Davide Sangiorgi has showed that a version of CHOCS with static binding for restriction (*Plain* CHOCS) can be faithfully encoded in the π-calculus.

The theory of name-passing languages has just begun being investigated. The main purpose of this thesis is to extend some well-established concepts and techniques of static process algebras to the name-passing setting, i.e. to the π-calculus. In particular, we want to devise well-founded *formal theories* and *proof-techniques* to reason about the calculus. These theories and techniques should permit, given any two system descriptions, to prove (or to disprove) that they are behaviourally related. Another aim of the thesis is to shed further light on some aspects of its *expressive power*; we shall investigate an encoding of a behavioural equivalence into simpler one, and show how this result can be used to reason about the former, more complex equivalence.

In the next section, some background on process algebras will be given. Section 1.2 introduces informally mobility in the context of process algebras. In Section 1.3, an overview of the thesis, highlighting its main contributions, will be presented.

1.1 Background in Process Algebras

The section contains a survey of well established concepts of static (CCS-like) process algebras which are relevant for the thesis: syntax, operational semantics and observational semantics.

1.1.1 Syntax

The syntax of a process algebra is given by specifying a set of operators and their arities. From now on we will consider CCS to exemplify. We presuppose an infinite set of elementary actions *Act*, ranged over by α, and an infinite set of *agent variables*, ranged over by X. The language of CCS, built from the operators of *agent variable, inaction, action prefix, summation, restriction, parallel composition, recursion*, can be described by the following grammar:

$$P ::= X \mid \mathbf{0} \mid \alpha.P \mid P + P \mid P\backslash\alpha \mid P \mid P \mid recX.P.$$

Beside the above operators, also an action relabeling operator is usually considered, but our selection is sufficient to illustrate the main ideas of the approach.

Inaction $\mathbf{0}$ represents the process that cannot perform any action; $\alpha.P$ performs the action α and then becomes P; the execution of an elementary action α may be viewed as a *synchronization* with the external environment through a channel named α. Restriction provides us with an abstraction mechanism analogous to local declarations in imperative languages: in $P\backslash\alpha$, $\backslash\alpha$ prevents P from interacting through the channel α; therefore α can be used for internal communications only. For expressiveness reasons, a distinct action τ is also considered. It is called the *silent* or *internal* action because it models an internal activity of the system; thus its execution is meant not to be influenced by the environment. As such, it cannot be restricted. $P + Q$ denotes the non-deterministic choice between P and Q and $P \mid Q$ represents the parallel execution of P and Q. Finally, X and $recX$ together are enough to describe recursive behaviour: $recX.P$, where X can occur inside P, has the same meaning as the agent defined by the equation $X = P$.

Let us consider now the issue of providing a formal semantics for CCS. For sequential languages a denotational semantics style is well-established, where each program is compositionally assigned a mathematical object, representing the input/output function it computes. This approach however is not suitable for concurrent languages (and process algebras are very abstract concurrent languages). Concurrent systems are no longer deterministic, thus it does not make sense to consider I/O functions; moreover, intermediate states of computations are as much relevant as the final ones. The essence of a concurrent system lies indeed in its ability or inability, at each instant, to interact with the external environment. An operational point of view is therefore taken, based on the claim that *two terms should have the same semantics whenever no external observer can distinguish among them.* This gives an *extensional* view of processes. The standard approach to achieve this goal is to give a two-level semantics. At the lower level, each term is assigned an abstract machine accounting for its operational behaviour (*operational* or *transitional semantics*). At the upper level, a behavioural relation abstracting from "irrelevant" details of the operational behaviour of processes is defined over the term algebra (*observational semantics*). We next examine in detail these two levels.

1.1.2 Operational semantics

The most common abstract machines used for describing terms behaviour are *Labelled Transition Systems (LTS)*. These are also known as non-deterministic state automata in Language Theory. A LTS can be described as a (finite or infinite) rooted directed

$$\texttt{Act} \;:\; \frac{-}{\alpha.P \xrightarrow{\alpha} P}$$

$$\texttt{Sum} \;:\; \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'}$$

$$\texttt{Par} \;:\; \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2} \qquad \texttt{Com} \;:\; \frac{P_1 \xrightarrow{\alpha} P_2',\; P_2 \xrightarrow{\overline{\alpha}} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

$$\texttt{Res} \;:\; \frac{P \xrightarrow{\alpha} P'}{P\backslash\beta \xrightarrow{\alpha} P'\backslash\beta}\, \alpha \neq \beta \qquad \texttt{Rec} \;:\; \frac{P[recX.P/X] \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

Symmetrical versions of rules `Sum` and `Par` are omitted.

Table 1.1: SOS rules for relabelling-free CCS.

graph, with labelled arcs. The nodes are called states and the arcs are called transitions. Equivalently, a LTS with states in a set $\mathcal{S}$ and actions in a set A may be defined as a ternary relation $\longrightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$, where $(P, \alpha, Q) \in \longrightarrow$ is written as $P \xrightarrow{\alpha} Q$. Each term of the algebra is assigned a LTS in such a way that states are terms, labels are elementary actions (taken from *Act*) and the existence of a transition $P \xrightarrow{\alpha} Q$ means, intuitively, that the process P can evolve to Q by performing an action α. The assignment of LTS to terms is done using a methodology called *Structural Operational Semantics (SOS)*, due to Plotkin [Plo81]. It consists in giving for each operator of the language a (possibly empty) set of inference rules, defining its transitions in terms of those of its subcomponents. The SOS rules defining CCS operational semantics (for the considered fragment) are given in Table 1.1.

Note that the `Par` rule for parallel composition says that two parallel processes can evolve independently; this is rendered by stating that their actions can be *interleaved* in any possible order. The rule `Com` says that two parallel processes capable of executing complementary actions (such as α and $\overline{\alpha}$) may synchronize; this synchronization results in an internal action. In the rule `Rec` , $[Y/X]$ is used to denote substitution of the free (not inside the scope of $recX.$) occurrences of X with the term Y. The other rules should be self-explanatory.

1.1.3 Observational semantics

Operational semantics is too "concrete", that is it makes too many distinctions among processes. For example, it is clear that we wish to consider terms such as $P + \mathbf{0}$ and P as equivalent, though they have different LTS's. Another example is the pair $P \mid \mathbf{0}$ and P. Other subtler examples involve the internal action τ. Since this is considered as "invisible" to any observer, we would like to have equivalences allowing us to identify processes such as $\alpha.\tau.P$ and $\alpha.P$.

Since LTS's contain in a sense too much information, the problem is to abstract away from unwanted details via some uniform observation mechanism. Depending on *what* is being observed, there is a whole scale of observation mechanisms that yield finer and finer semantics: they range from observing just the sequences of actions on the transitions, regardless to the internal structure of the LTS, to taking into account not only the internal structure of the LTS, but also the internal structure of the states, which is yielded by the syntax. At the first extreme, we find *trace equivalence* (somewhat derived from Language Theory), which considers two terms as equivalent whenever they (better, their LTS's) "accept" the same language (i.e. the same subset of Act^*). This solution is unsatisfactory in at least one respect: it ignores possible deadlocks. For example, consider the two processes $P = \alpha.(\beta + \gamma)$ and $Q = \alpha.\beta + \alpha.\gamma$ (where we have abbreviated $\alpha.\mathbf{0}$ in α, for any action α). They are trace equivalent (the accepted language is in both cases $\{\epsilon, \alpha, \alpha\beta, \alpha\gamma\}$), but P delays the choice between β and γ after the execution of α. Thus, if we put P in an environment requiring the synchronization with $\overline{\alpha}.\overline{\beta}$, P can always do it; Q deadlocks after executing α, if it chooses the "wrong" branch. Thus we need finer, deadlock-sensitive semantics, which take into account also the internal structure of LTS's.

We next examine the two main families of semantics for process algebras: *interleaving* semantics and *true-concurrent* semantics. Both take into account the internal structure of LTS's, but only in the latter the structure of states is considered.

Interleaving

The basic assumption of interleaving semantics is that the observer regards the system as a "black-box", from which it can only detect the occurrence of one action at each step. We survey below the classical interleaving deadlock-sensitive semantics based on *bisimulation* [Par81, Mil89] and *testing equivalence* [DH84, Hen88].

A symmetric binary relation $\mathcal{R}$ over CCS is a *strong bisimulation* if, for each pair $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\alpha} P'$, then there exists a Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{R}$. We define $P \sim Q$ ("P strongly bisimilar to Q") if and only if there exists a strong bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

It is easily seen that $\sim$ is an equivalence relation, and furthermore it is a *congruence* with respect to CCS operators. This fact has a fundamental importance: it means that, when analyzing a concurrent system, we can freely substitute, within any context, a subsystem with a strong bisimilar one, being guaranteed that the result will be an equivalent system. Furthermore, as easily seen, $\sim$ is deadlock-sensitive. Yet, it is too strong, in that internal actions are visible to the observer, just like any other action. The above definition can be modified to obtain a *weak* version of bisimulation relation, which abstracts from internal moves: define $\Longrightarrow$ to be the reflexive and transitive closure of $\xrightarrow{\tau}$; then, for any *visible* (different from τ) action α define $\stackrel{\alpha}{\Longrightarrow}$ to be the composition of relations $\Longrightarrow \xrightarrow{\alpha} \Longrightarrow$. The definition of weak bisimulation is obtained from the strong one by replacing $\xrightarrow{\alpha}$ with $\stackrel{a}{\Longrightarrow}$. We let $P \approx Q$ ("P weakly bisimilar to Q") if and only if there exists a weak bisimulation $\mathcal{R}$ containing the pair (P, Q). Again, it is easily seen that $\approx$ is an equivalence relation; yet, it is *not* a congruence, in that it is not preserved by the non-determinism operator $+$. Usually indeed, one works with the *observational congruence*, the maximal congruence included in $\approx$.

The general setting for *testing equivalence* requires formally defining a set of *observers*, a *way of observing* and a general criterion for *interpreting the results* of observations. If we call $\mathcal{S}$ the set of systems we want to experiment upon, to apply the general setting, we need to define a set of observers, say $\mathcal{O}$, and explain the evolutions of the pairs $(S, o) \in \mathcal{S} \times \mathcal{O}$, which will represent the *interactions* between S and o. Interactions may be failing or successful, depending on whether particular states (which report success) are reached. For specific system S and observer o, one might be interested in knowing whether a successful interaction does exists, i.e. whether S *may satisfy* o, or whether all interactions are successful, i.e. whether S *must satisfy* o. Two testing preorders over $\mathcal{S}$, naturally arise, associated with each of the above points of view: one define that S is may/must - "less or equal than" R if the set of observers which are may/must - satisfied by S is a subset of those satisfied by R. From each of the these preorders, a distinct testing equivalence can be extracted (as the kernel of the preorder). In [DH84] the above machinery is applied to CCS, and the view is taken that *observers are processes themselves*, which interact with other processes by communicating with them. Furthermore, it is shown that (mild modifications of) the testing preorders are indeed precongruences.

Let us now contrast benefits and drawbacks of the bisimulation approach with those of the testing one. Bisimulation enjoys a nice proof technique (that can be informally described as "finding bisimulation relations"); in fact there are algorithms that, given two LTS's, can verify whether they are bisimilar in polynomial time [KS90]. However, bisimulation (even the weak version) is in many respects too discriminating: in [Abr87], Abramsky argues that bisimulation corresponds to a testing scenario where the observer is allowed, among other things, to produce an unbound number of copies of the process and make them run in parallel. On the other hand, testing lacks an equally nice proof technique, but is more natural: it appears to be the weakest deadlock-sensitive extensional equivalence one would like to impose over processes. Furthermore, the notion of preorder naturally supports a method of implementation based on stepwise development (for example, we can interpret the must preorder as an "is implemented by" relation). Another advantage of testing w.r.t. bisimulation is that it can be naturally re-approached with the denotational semantics. In [DH84, Hen88], partial-order denotational models of *acceptance trees* are given for a CCS-like language. Each model is *fully abstract* w.r.t. a testing preorder, in the sense that two process are related by the testing preorder if and only if their denotations are related in the partial-order model of trees. Such models for process algebras are important in that they give an abstract view of processes as mathematical objects.

Both bisimulation and testing preorders have simple *algebraic* characterizations over CCS [DH84, Mil89, Hen88]. We see this as one of the most important achievement of the process algebra approach. An algebraic characterization of a behavioural relation (preorder or equivalence) is given through an (in-)equational proof system, consisting of a set of inference rules plus a set of (in-)equations over the algebra of terms. The system has to be *sound*, i.e. each (in-)equation derivable within it must hold w.r.t. the relation, and *complete*, i.e. whenever two terms are behaviourally related, they are related by the proof system as well. This permits manipulating terms in a completely syntactical way, and is the basis for a class of verification tools founded on rewriting techniques [DIN90]. When defining a new behavioural relation for a process algebra, finding an algebraic characterization for it is a major concern.

Thus we have *three views* of interleaving semantics: *behavioural*, *denotational* (at least for testing) and *algebraic*.

True-concurrency

The black-box assumption on which interleaving semantics are based permits to develop a simple and elegant mathematical theory, but it has a major drawback: it reduces parallelism to sequentiality plus non-determinism. This is condensed into the equation that is the slogan of the interleaving approach:

$$\alpha \mid \beta = \alpha.\beta + \beta.\alpha\,.$$

Several authors claim that this is not a "faithful picture of reality". Furthermore, unfolding a parallel system into its interleaving representation, leads to an explosion of the number of states to consider. A number of semantics have been thus proposed aiming at giving a description of true concurrency ($\alpha|\beta$) as a primitive notion, distinguished from interleaving ($\alpha.\beta + \beta.\alpha$). Most of these proposals take into account either the causal dependencies among actions [DDM88, DD89, Kie91, DP92, Kie93], or the spatial distribution of the system [CH89, BCHK93, BCHK94, Kie91, DP92, MY92, CD93, Kie93]. For example, in $\alpha \mid \beta$ the actions α and β are independent and have two distinct locations, while in $\alpha.\beta + \beta.\alpha$ there is always a dependency between the two actions (the firing of α enables β, or vice-versa) and just one location.

By and large, the actual definition of these semantics extends the observation scenario of the interleaving ones, to allow the observer to look at the structure of the states.

Equivalently, part of the information concerning the structure of the states can be moved on the transitions, giving rise to an *enriched* transition system. For example in [Kie91], building on an idea of Degano and Darondeau [DD89], a new transition relation is introduced where transitions take the form $P \xrightarrow[K;k]{\alpha} Q$: here, K represents the set of causes of the action and k is a pointer to the location inside P where the action (α) has been executed. On top of this operational semantics, one can define bisimulation (or testing as well, though no investigation has been made in this direction) with familiar techniques, by requiring not only the matching of the actions (α), but also of their causes and/or locations. Indeed (weak/ strong) causal- and spatial- preserving versions of bisimulation are currently studied. They give a better account of the structure and of the flows of activity in the concurrent systems than interleaving bisimulation equivalences. As a consequence, some causal semantics permit for example recovering compact representation of concurrent systems in terms of Petri Nets or partial-order-like structures [DDM88]; again, these semantics are useful, in the debug of a system, when detecting a fault in a computation: causality/locality information helps us in understanding which events of the computation may have been affected by the fault, and which ones may not.

1.2 Mobility

Mobility is an important phenomenon in concurrent systems. Consider an operating system that grants the access to resources dynamically. We may view a resource as a process itself, which is accessed through a particular channel. As time passes, processes can acquire the access right by receiving the channel. Another classical example is a procedure which may be called by an unbound number of processes at the same time. Thus several copies of it may be running in parallel. When a copy terminates, the control and the result are given back to the appropriate point of the calling process, thus a return address is needed. We may view the activation of the procedure as a synchronization on a specific channel; the return address is a particular channel itself, passed by the caller to the procedure when activating it, on which the procedure synchronizes and give the result back upon finishing.

Static process algebra such as CCS cannot describe directly such phenomena: they need to be extended. Let us discuss now the way this can be done. This will lead us essentially to name-passing calculi such as ECCS and π-calculus. The presentation below intends to be informal; a complete presentation of the π-calculus is deferred to the next chapter. Some operators remain the same as in CCS; this is the case for $\mathbf{0}$, the terminated process; $\tau.$, the silent action prefix; $+$, the nondeterministic choice of two processes; $|$, the operator for parallel composition and rec, recursion. But surely, one has to abandon the blurring between input and output proper of CCS, and to explicitly distinguish input and output prefixes. This fact will also force a radical revision of the restriction operator.

Let us write as $x(y).P$ the process that receives a name along the channel x and then behaves like P; here y is, so to speak, the formal parameter for the received name; the prefix $x(y)$. is considered a *binder* for the name y (in the same sense as, in λ-calculus, λy. is). The process that outputs a name z along the channel x and then becomes Q is written $\overline{x}z.Q$. After receiving a name, a process can use it for further communications. Thus, as an example, the the following transitions should be possible:

$$(x(y).\overline{y}w.\mathbf{0}) \mid \overline{x}z.\mathbf{0} \xrightarrow{\tau} (\overline{z}w.\mathbf{0}) \mid \mathbf{0} \xrightarrow{\overline{z}w} \mathbf{0} \mid \mathbf{0}\,.$$

Here, the interaction at x, between the input action $x(y)$ and the output action $\overline{x}z$, results in a communication (τ); in this communication, the name z is passed to the receiving process and substituted for the name (formal parameter) y. After that, the receiving process, that previously was not able to interact along channel z, can perform an output (with the external environment) of the name w along channel z.

The other subtle point is the behaviour of the restriction operator. To stress the difference from CCS, restriction is written $(y)P$ in the π-calculus; it has to be considered as a binder too, though of a different nature than input prefix. Clearly, (y) should prevent P from communicating through the channel y. But it is not obvious what the effect of (y) should be on y *as a datum* (in, say, an output prefix $\overline{x}y$). To simply forbid such a datum to be communicated is a too weak solution from the viewpoint of expressiveness. Instead, we want to be able to express the communication of a private, i.e. restricted, name. This implies *scope extrusion*: when passing a restricted name, the scope of the corresponding restriction is expanded to include the receiver process. Therefore, after the communication, the name, that previously was private to the sender, remains private to both the sender and the receiver. An example of possible transition thus should be (supposing $w \neq x$ and denoting the substitution of y with w by $\{w/y\}$):

$$x(y).P \mid (w)(\overline{x}w.Q) \xrightarrow{\tau} (w)(P\{w/y\} \mid Q)$$

provided that w is not free in $x(y).P$. The reason of the latter condition will be explained shortly. Indeed, the transition can also be thought of as happening via the following steps: first, the scope of (w) is enlarged to include $x(y).P$, thus obtaining a term $(w)(x(y).P \mid \overline{x}w.Q)$; this requires again that w is not free in $x(y).P$, to avoid capturing free names; if it is not the case, w has first to be renamed in $(w)(\overline{x}w.Q)$; then a standard interaction between $x(y)$ and $\overline{x}w$ occurs underneath the restriction (w). Thus, the "freshness" condition on w, or equivalently its renaming, are needed to prevent a free name of the receiver, *accidentally* equal to w, from being bound after the scope extrusion. We want namely a *static scoping* discipline for restriction. This choice agrees with the viewpoint that identity of bound names is not important, thus processes that are the same up to the choice of bound names should be identified.

It is not an exaggeration to say that the expressive power of the π-calculus (and similar languages) lies mainly in the way it handles private names. The treatment of private names we described above was first introduced by Engberg and Nielsen with their language ECCS [EN86]. Their work remained scarcely known for a long time in the process algebra community, probably due to the difficult semantic treatment induced by the distinction between "constant" and "variable" channel name. The π-calculus was introduced later, as a substantially refined version of ECCS. The major simplification w.r.t. ECCS is a uniform treatment of constants and variables, that are viewed as a unique syntactic entity. Indeed, there are only two basic entities in the π-calculus: *names* and *processes*. The exchange of names is the only computational activity of the system. Surprisingly, this simple basis is enough to encode, in a fairly natural way, computation over arbitrary data types [MPW92], besides mobility phenomena that often arise in the computer world, such as the ones mentioned at the beginning of the section.

However, the π-calculus is also important from a more theoretical point of view. In [Mil92], it has been shown that the λ-calculus can be naturally and faithfully encoded in the π-calculus. This fact establishes a bridge between functional and concurrent programming; in particular, it is the foundation for correctly implementing the functional languages with the concurrent ones.

1.3 Overview of the Thesis

1.3.1 Achievements of the thesis

The thesis gives original contributions in the fields of formal theories and reasoning techniques for new and old semantic theories and in the field of expressive power. Below, we discuss the significance of each of these contributions.

Testing Semantics

Milner, Parrow, Sangiorgi and Walker have developed basic semantic theories for the π-calculus [MPW92, PS93, San93]; all these theories are bisimulation-based and, moreover, they are *strong*, in the sense that the considered behavioural equivalences do not abstract from internal moves of processes. It is therefore interesting to investigate alternative, possibly not strong, theories for the calculus.

The aim of this part of the thesis is to explore the impact on a name-passing calculus of the testing approach; this is interesting also because the testing approach naturally leads to *weak* behavioural relations, which abstract from internal moves of processes.

The testing framework has been applied successfully to CCS in [DH84]. Applying it to the π-calculus requires additional care w.r.t. two points: the distinction between input and output actions (which is absent in CCS) and the presence of two distinct kinds of output actions (output of public and private names). The latter fact, in particular, implies that, when prompted for an output, a π-calculus process has in general different options.

To fully discriminate the different behaviours, we need observers that can test whether two received names equal or not. Recalling that observers are processes themselves, we have extended the language with a *mismatch* operator that gives us the needed observational power. Although not present in the original calculus, the relevance of this operator to the calculus is currently debated (for example, it has been used in [PS93] to axiomatize a bisimulation-based equivalence). An achievement of this part of the thesis is to establish the importance of mismatch from an observational point of view: we prove that its omission from the language leads to different testing equivalences, that moreover do not correspond to any operational intuition.

We also show the "stability" of the testing approach w.r.t. different variations of the underlying operational semantics proposed in the literature: they all lead to the same testing behavioural relations (the same result is not true for bisimulation).

The major achievements of our study of testing are an axiomatization and a fully abstract denotational model for testing preorders. As regards the former, we prove that a slight adaptation of the testing axiomatization over CCS (plus a few simple axioms for match and mismatch operators) is sound and complete over the π-calculus as well. This may provide a support for the "universality" of the testing axioms (as a matter of fact, the same is not true for bisimulation, where an extra non-trivial axiom is needed in the axiomatizations of a bisimulation-based semantics, the *early* equivalence [MPW92, PS93]).

The denotational model is based on reduction of processes to certain normal forms. Though this model is somehow syntax-dependent, its existence and the fact that it is based on normal forms are important, because they throw light on the mathematical nature of π-calculus processes.

This part of the thesis is based on joint work with Rocco De Nicola [BD92b, BD92a].

Symbolic Semantics

Quite a number of semantics for the π-calculus have so far been explored, but research to support *effective* verification has just began.

This part aims at laying the basis for efficient verification of known bisimulation-based equivalences. The fundamental notion in the theory of bisimulation for the π-calculus is that of *ground* bisimulation, introduced in [MPW92]. It enjoys a simple, CCS-like definition, which however runs into serious efficiency problems. In fact, a *case analysis* on input actions is required to check that receiving equal names leads to equivalent states. To see this, suppose we want to verify the equivalence of two processes $a(y).P$ and $a(y).Q$; following the definition, we would have to verify that P and Q are bisimilar when instantiating the "formal parameter" y of the input action with any name occurring free in P and in Q. On the other side, it is easy to see that, often, a lot of of these checks are useless. The reason of this inefficiency is that the usual definition of bisimulation completely ignores the logical structure of processes.

Building on previous work by Hennessy and Lin on static value-passing process algebras [HL92, HL93], we use *symbolic transition system* to give a new semantics for the π-calculus, more amenable to efficient checking. Moreover, we show that the usual bisimulation-based semantics [MPW92, PS93] can be characterized in terms of the symbolic semantics. As a result, more efficient checking of these equivalences is possible.

The symbolic transitions we introduce are of the form $P \xrightarrow{\phi,\alpha} P'$, where ϕ is a *boolean constraint* over names that has to hold for the transition to take place, and α is a standard π-calculus action. We define symbolic bisimulation on the top of the symbolic transition system, in a way that is parametric over the language of boolean constraints. This leads to distinct symbolic equivalences $\simeq^\phi$, depending on the boolean constraint ϕ. The idea is to perform the case analysis directly on the logical constraints, in such a way that useless alternatives can be discarded. This leads to a significant gain in efficiency.

The main result of this part is a "consistency and adequacy" theorem relating standard and symbolic bisimulations; it says essentially that each symbolic equivalence $\simeq^\phi$ is the closure of the ground equivalence under all name-substitutions satisfying ϕ.

Another outcome of this part of the thesis is a proof system to reason about symbolic bisimulation. The statements derivable within the system are of the form $\phi \rhd P = Q$ and the system is sound and complete in the sense that $\phi \rhd P = Q$ is derivable if and only if $P \simeq^\phi Q$.

The symbolic characterization of the standard equivalences has also an additional advantage; it sheds new light on the conceptual difference between different forms of bisimulations. Actually, in [MPW92], two forms of ground bisimulations were introduced. Intuitively, they correspond to two different instantiation strategies for the formal parameter of input actions: in the first strategy, the instantiation is performed as soon as the action fires; in the second one, the instantiation is performed later on, when a communication is actually inferred. Our symbolic formulation indicates that each of the mentioned strategies corresponds to a different degree of generality in performing case-analysis.

This part of the thesis is based on joint work with Rocco De Nicola [BD94a, BD94b]

Causal Semantics

The main goals of this part of the thesis is to investigate causality in a name-passing setting and the the relationship between interleaving and causal semantics for the π-calculus. We focus on the interleaving *observation equivalence*, written $\approx$ [Mil89, MPW92], and on the non-interleaving *causal equivalence*, written $\approx_c$ [DDM88, DD89, Kie91]; these are among the most studied behavioural equivalences in the respective groups. Our main theorem shows that causal equivalence can be reconducted to, or implemented as, observation equivalence. This is, primarily, an expressiveness result for the π-calculus (the encoding exploits in a crucial way the peculiar features of the π-calculus of passing names and generating "fresh" names); it also allows us to use the simpler theory of the interleaving semantics to reason about the causal one.

Another achievement of this part of the thesis is to shed light on the meaning of causality in calculi for mobile processes. We have singled out two forms of dependencies between actions, which we have called *subject dependencies* and *object dependencies*. Subject dependencies originate from the nesting of prefixes and are propagated through internal communications of processes. In CCS-like languages, i.e., languages in which actions represent pure synchronization, subject dependencies are the only form of causal dependencies. The other form of dependencies, namely the object dependencies, are determined by π-calculus binding mechanisms on actions; therefore they do not exist in CCS. Approximately, an action λ is object dependent on a previous action μ if μ acts as binder for some name of λ; as such, μ can be thought of as supplying some names to λ. We define a causal bisimulation which distinguish among processes on the basis of the object and subject dependencies they exhibit. We enrich the standard (i.e., interleaving) transition system of the π-calculus with information about the subject dependencies. This is enough, since the object dependencies can already be detected in the standard system.

The main result of this part is to show that the sophisticated observational machinery of causal bisimulation ($\approx_c$) can be faithfully represented from within the interleaving observation equivalence ($\approx$). More precisely, we define a compositional encoding $[\![\,.\,]\!]$ from the enriched calculus to the basic one, and we prove that it is fully abstract w.r.t. $\approx_c$ and $\approx$; i.e., for each pair of terms A and B in the enriched calculus we have:

$$A \approx_c B \text{ if and only if } [\![A]\!] \approx [\![B]\!]\,.$$

Through various examples, we show how this result and the algebraic theory of $\approx$ can be used to prove properties about $\approx_c$. When restricted to the sub-language with no object dependencies, that is CCS, our encoding yields a characterization of the classic causal bisimulation of [DD89, Kie91] in terms of the observation equivalence of the π-calculus.

This part of the thesis is based on joint work with Davide Sangiorgi [BS94, BS94b].

1.3.2 Structure of the thesis

The rest of the thesis is as follows:

- Chapter 2 gives a complete presentation of the π-calculus. First the basic language of [MPW92] and then a few variants of it that are relevant for the thesis will be introduced. The reason for considering different variants is that certain operators that are relevant in a given context, are not in another one.

- Chapter 3 presents a theory of *testing* that yields algebraic and denotational characterizations of testing-based behavioural relations over the language;

- Chapter 4 presents a *symbolic semantics* that gives us a more efficient formulation of known bisimulation-based equivalences;

- Chapter 5 presents a study of causality in a name passing setting and a reducibility result; the latter says that a specific *causal semantics* based on bisimulation can be expressed, in a precise sense, in an interleaving bisimulation-based one.

- Chapter 6 presents conclusive remarks on the achieved results and detailed comparison with related work.
- Appendices A and B contain detailed expositions of the most lengthy or difficult proofs, which for ease of reading are omitted in the main body.

Chapter 2

The π-calculus

The aim of this section is to give a comprehensive presentation of the π-calculus. In Section 2.1, the basic language of [MPW92] and *late* transitional semantics are introduced; then a few more operators, often found in the literature and relevant for the thesis, are presented; some basic properties of the late semantics are established and finally the *early* variant of the transitional semantics is introduced. Section 2.2 presents a substantial extension of the basic calculus, the *polyadic* version of [Mil91], where the exchange of tuples of names is allowed. Here also a sorting discipline over names has to be considered. Our presentation of sorts and sortings owes much to that of [San92]. Section 2.3 contains some definitions of bisimulation-based equivalences that will be used throughout the thesis. The chapter ends with an example, the description of a scheduler in polyadic π-calculus.

In the thesis, we shall deal with three slightly different languages, which may be considered as different fragments of the same general calculus. This separation has been done mostly for the sake of clarity. Certain operators that are critical for a given subject are not for another subject: in the latter case, they have been omitted if their treatment could be easily accommodated. This is the case, for example, of polyadic input and output and the sorting discipline, which are necessary in dealing with causal semantics, but have no particular relevance in the testing and symbolic semantics.

2.1 The Basic Language

The calculus considered below is essentially that of [MPW92], to which the reader is referred to for additional details. The only exception w.r.t. the presentation in [MPW92] is that we consider a slightly simpler form of recursive operator.

2.1.1 Syntax

We presuppose a countable set $\mathcal{N}$ of *names* ranged over by the metavariables $x, y, z, \ldots$ and (less frequently) by $a, b, \ldots$. The language $\mathcal{P}$ is built from the operators of *inaction, agent* *variable, silent prefix, input prefix, free output prefix, restriction, match, sum, parallel composition* and *recursion* according to the following grammar:

$$\begin{array}{rl} P ::= & \mathbf{0} \;\mid\; X \;\mid\; \tau.P \;\mid\; x(y).\,P \;\mid\; \overline{x}y.P \;\mid\; (y)P \\ & \mid\; [x=y]P \;\mid\; P+P \;\mid\; P\,|\,P \;\mid\; recX.P \end{array}$$

where X ranges over a countable set of agent variables. The elements of $\mathcal{P}$ are called *processes*. Metavariables $P, Q, R, \ldots$ range over $\mathcal{P}$. We will often use the *bound output prefix* $\overline{x}(y).P$, $x \neq y$, as a shorthand for $(y)(\overline{x}y.P)$.

We have two kinds of *name binders*: input prefix $x(y).P$ and restriction $(y)P$ bind the name y in P; consequently, the notions of *free names, fn(.), bound names, bn(.)*, and *α-equality,* $\equiv_\alpha$, over process terms and actions are the expected ones (in analogy with λ-calculus). We define $n(.) = fn(.) \cup bn(.)$. We shall use $fn(y_1, ..., y_n, \alpha_1, ..., \alpha_k, P_1, ..., P_h)$ as an abbreviation for

$$\{y_1, ..., y_n\} \cup fn(\alpha_1) \cup \ldots \cup fn(\alpha_k) \cup fn(P_1) \cup \ldots \cup fn(P_h).$$

Similar notations hold for the functions $bn(\,.\,)$ and $n(\,.\,)$.

We impose two constraints on the syntax above: (i) every occurrence of an agent variable X is within the scope of a $recX.$ operator, and (ii) for each term $recX.P$, no occurrence of X inside P is within the scope of a y-binder, for $y \in fn(P)$. The latter constraint ensures that no free name of P becomes bound when "unfolding" the term $recX.P$ into $P[recX.P/X]$ (where $[Y/X]$ denotes substitution of the agent variable X with the term Y). The latter requirement can always be met by suitable α-renaming of bound names.

Substitutions, ranged over by σ, ρ, are functions from $\mathcal{N}$ to $\mathcal{N}$; for any $x \in \mathcal{N}$, $\sigma(x)$ will be written as $x\sigma$; $P\sigma$ denotes the result of applying the substitution σ to P, i.e. the expression obtained from P by simultaneously replacing each $x \in fn(P)$ with $x\sigma$. As usual, we assume that clashes of names are dealt with by means of renaming of bound names with fresh names. Composition of substitutions is denoted by juxtaposition: given two substitutions σ_1 and σ_2, $\sigma_1\sigma_2$ denotes the substitution such that for all x $x(\sigma_1\sigma_2) = (x\sigma_1)\sigma_2$. A set $\{x_1/y_1, \ldots, x_n/y_n\} = \{\widetilde{x}/\widetilde{y}\}$, with the x_i's pairwise distinct, will denote the following substitution σ: $x\sigma = y_i$ if $x = x_i$ for some $i \in \{1, \ldots, n\}$, $x\sigma = x$ otherwise.

The language is an extension of CCS [Mil80] which permits expressing communication of channel names and thus describing systems with a dynamically changing communication structure. Indeed, some operators are essentially the same as in CCS; this is the case for $\mathbf{0}$, the terminated process; $[\,.\,] + [\,.\,]$, the nondeterministic choice of two processes; and $[\,.\,]|[\,.\,]$, the operator for parallel composition and $recX.[\,.\,]$, recursion. Action prefix, $\alpha.[\,.\,]$, is only similar to that of CCS; here a distinction is introduced between input and output actions. Restriction $(y)[\,.\,]$ (another commonly found notation is $(\nu\, y)$) is similar in spirit to the corresponding CCS operator, but its scope may change dynamically, when a process exports a restricted name. The meaning of the match operator $[x = y][\,.\,]$ is that $[x = y]P$ behaves like P if x is syntactically equal to y and like $\mathbf{0}$ otherwise.

Convention

Following [MPW91, PS93], in the thesis, *we work up to α-equivalence*. This means that α-equivalent processes and actions are identified. The symbol $=$ denotes syntactic equality up to α-equivalence.

Notations

We often abbreviate $\alpha.\mathbf{0}$ simply as α. To save parentheses, we assume the following precedences among operators in process expressions:

{ restriction, prefix, match } > parallel composition > summation > *rec*.

Moreover, substitutions will have the strongest precedence, thus e.g. $\alpha.P\sigma$ will stand for $\alpha.(P\sigma)$. We will often use the symbol $\stackrel{def}{=}$ for defining objects (e.g., $P \stackrel{def}{=} Q$ stands for "define P to be Q"). Finally, the symbol $\equiv$ denotes syntactic equality (up to α-equivalence) too; it will be used in place of $=$ whenever the latter symbol is reserved to proof-theoretic equality (equality induced by a proof system).

2.1.2 Late operational semantics

The basic semantics of the π-calculus is given in terms of a labelled transition system in the SOS style of [Plo81]. Different flavours of this semantics have been proposed. We present below the *late* operational semantics, which was introduced (in a slightly different, but equivalent, form) in [MPW92].

Besides the *invisible* transition labelled by τ, we have three kinds of transitions:

- the *input transition* $P \xrightarrow{x(y)} P'$; this means that P is able to receive a name from the environment along the channel x, and in doing so it becomes P'; the round brackets around y record that it only represents a "formal parameter" for the received value.

- the *free output transition* $P \xrightarrow{\overline{x}y} P'$; this means that P is able to export the public name y along x, and in doing so it becomes P';

- the *bound output transition* $P \xrightarrow{\overline{x}(y)} P'$; this means that P is able to export the private name y along x, and in doing so it becomes P'; the round brackets around y record that it is a restricted name, thus it has to be kept distinct from any other name.

Accordingly, we shall consider the following sets of actions:

- the set of *input actions* as $IA = \{x(y) |\, x, y \in \mathcal{N}\}$;

- the set of *output actions* as

$$OA = \{\overline{x}y |\, x, y \in \mathcal{N}\} \cup \{\overline{x}(y) |\, x, y \in \mathcal{N} \text{ and } x \neq y\}$$

- the set of *visible actions* as $\Lambda = OA \cup IA$;
- the set of all *actions* as $Act = \Lambda \cup \{\tau\}$.

Symbols $\alpha, \beta, \ldots$ will be used to range over Act.

If $\alpha = x(y)$ or $\alpha = \overline{x}y$ or $\alpha = \overline{x}(y)$, we let $subj(\alpha) = x$ and $obj(\alpha) = y$. Moreover, we extend the function fn, bn and n to Act by letting:

- $fn(\overline{x}y) = \{x, y\}$, $fn(x(y)) = fn(\overline{x}(y)) = x$, $fn(\tau) = \emptyset$
- $bn(\overline{x}y) = \emptyset$, $bn(x(y)) = bn(\overline{x}(y)) = y$, $bn(\tau) = \emptyset$.

and, as usual, $n(.) = bn(.) \cup fn(.)$.

The structural operational semantics of $\mathcal{P}$ is given in Table 2.1. Note that, since we are working up to α-equivalence, *α-equivalent processes are deemed to have the same transitions* (see also [MPW91, PS93]).

We comment on some of the inference rules. The instantiation of the formal parameter with the received name only happens when a communication is actually inferred with `Com` (This justifies the adjective "late" for the semantics). Up to that moment, we want to keep the formal parameter distinct from any other name. This explains the side condition of `Par` . The condition can always be met by a suitable α-renaming of the formal parameter y of the input action.

The export of a private name implies the scope extrusion of the corresponding restriction. This is implemented by the rules `Open` and `Close` together. The rule `Open` "opens" the scope of the restriction, which is closed when a communication is inferred by `Close` . Note that, when closing the scope with `Close` , the side condition of `Par` prevents any free name of the receiver to be bound by the restriction. This satisfies the static binding assumption for restriction. Finally, note that it is not restrictive to impose, in the premise of `Close` , that the bound names of the input and output actions are equal: this condition can be always met by means of suitable α-renamings. We refer the reader to [MPW92, MPW91] for further explanations on the meaning of these rules. We shall use the late operational semantics in Chapter 3 and 4.

Notations

We set that $P \longrightarrow Q$ stands for $P \xrightarrow{\tau} Q$ and that $P \Longrightarrow Q$ stands for $P(\xrightarrow{\tau})^*Q$ (where $\mathcal{R}^*$ denotes the reflexive and transitive closure of a binary relation $\mathcal{R}$); $P \stackrel{\alpha}{\Longrightarrow} Q$, stands for $P \Longrightarrow \xrightarrow{\alpha} \Longrightarrow Q$ (where $\mathcal{R}_1\mathcal{R}_2$ denotes the compositions of two binary relations $\mathcal{R}_1$ and $\mathcal{R}_2$).

2.1.3 Other operators

There are other operators, not included in the original calculus, that have been considered in the literature. The most important ones are *replication* $!P$ and *mismatch* $[x \neq y]P$. The

$$\texttt{Act} \ : \frac{-}{\alpha.P \xrightarrow{\alpha} P}$$

$$\texttt{Sum} \ : \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad \texttt{Par} \ : \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2} \ bn(\alpha) \cap fn(P_2) = \emptyset$$

$$\texttt{Com} \ : \frac{P_1 \xrightarrow{\overline{x}z} P_1',\ P_2 \xrightarrow{x(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'\{z/y\}} \qquad \texttt{Close} \ : \frac{P_1 \xrightarrow{\overline{x}(y)} P_1',\ P_2 \xrightarrow{x(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (y)(P_1' \mid P_2')}$$

$$\texttt{Res} \ : \frac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'} \ y \notin n(\alpha) \qquad \texttt{Open} \ : \frac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(y)} P'} \ x \neq y$$

$$\texttt{Match} \ : \frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'} \qquad \texttt{Rec} \ : \frac{P[recX.P/X] \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

Symmetrical versions of rules `Sum` , `Par` , `Com` and `Close` omitted.

Table 2.1: SOS rules for $\mathcal{P}$.

former, first proposed in [Mil91], has the meaning of "as many copies of P as we want" ($!P = P \mid P \mid P \mid \cdots$). Its formal transitional semantics is given by the inference rule:

$$\texttt{Repl} \;:\; \frac{P \mid {!P} \xrightarrow{\alpha} P'}{{!P} \xrightarrow{\alpha} P'}.$$

In [Mil91], it is shown that it can encode recursion under fairly general assumption, while having nicer algebraic properties. We shall consider it in Chapter 5.

Mismatch has the exact opposite meaning of match: $[x \neq y]P$ behaves like P if x is different from y and like $\mathbf{0}$ otherwise. It is worth noticing that match and mismatch, together with sum, permit easily expressing the if-then-else construct; indeed $[x = y]P + [x \neq y]Q$ corresponds to `if` $x = y$ `then` P `else` Q. The formal semantics of mismatch is:

$$\texttt{Mismatch} \;:\; \frac{P \xrightarrow{\alpha} P'}{[x \neq y]P \xrightarrow{\alpha} P'}\; x \neq y.$$

We shall consider this operator in Chapter 3. In Chapter 4, we shall consider a more general form, that subsumes both match and mismatch.

Another operator we shall make use of, in Chapter 3, is the *divergence* constant Ω. Generally, it is used in the testing approach to represent those agents which perform only infinite sequences of internal actions. It has no inference rule associated with it.

2.1.4 Basic properties of late transitional semantics

We establish below some elementary properties of the late transitional semantics that will be extensively used in later chapters. We assume here that processesmay contain any of the operators described in the preceeding subsections. All the three lemmata below are easily proven by induction on the derivation of the transition $P \xrightarrow{\alpha} P'$.

A first obvious fact is the following.

Lemma 2.1.1 *If $P \xrightarrow{\alpha} P'$ then: $fn(\alpha) \subseteq fn(P)$ and $fn(P') \subseteq fn(P) \cup bn(\alpha)$.*

Next two lemmata will give us some freedom in renaming names in transitions. Lemma 2.1.2 asserts that transitions are preserved by injective substitutions. Lemma 2.1.4 allows us to α-rename bound names with fresh names in transitions. Furthermore, in both cases, the renamed transitions have derivations "equivalent" to the original ones.

In what follows, we will use the following terminology (borrowed from [MPW92], part II). Sentences such as "if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\beta} Q'$ is derivable with the same depth as $P \xrightarrow{\alpha} P'$" will be abbreviated as "if $P \xrightarrow{\alpha} P'$ then *equally* $Q \xrightarrow{\beta} Q'$".

Lemma 2.1.2 *Let σ be an injective substitution over $fn(P)$ and suppose $bn(\alpha) \cap n(\sigma) = \emptyset$. If $P \xrightarrow{\alpha} P'$ then equally $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.*

Remark 2.1.3 When the mismatch operator is not considered, a stronger version of the above lemma holds. More precisely, in that case the hypothesis on the injectivity of σ can be discarded (see [MPW92]).

Lemma 2.1.4 *If $P \xrightarrow{\alpha} P'$, with $bn(\alpha) = \{y\}$ and $y_0 \notin fn(P)$, then equally $P \xrightarrow{\beta} P'\{y_0/y\}$, with β α-equivalent to α and $bn(\beta) = \{y_0\}$.*

Remark 2.1.5 The assumption of identifying α-equivalent agents and the above two lemmata allow us to assume w.l.o.g. that *bound* names, both in processes and in actions, are always *fresh*. In particular, we shall assume that given a bound name y and a substitution σ, y is fresh w.r.t. σ, i.e. $y \notin n(\sigma)$. A completely formalized argument would require, in each case, to rename y to a fresh y_0, by α-converting agents (this could in turn imply using Lemma 2.1.2 to rename y throughout part of a derivation in which y occurs free) or actions (using Lemma 2.1.4).

2.1.5 Early operational semantics

Early operational semantics, first proposed in [MPW91], differs from the late one as to the instantiation of the input formal parameter. In early operational semantics the instantiation is done as soon as the rule for input prefix is used (recall that in the late case the instantiation is instead performed only when a communication is actually inferred). As a consequence, in place of *bound* input transitions $P \xrightarrow{x(y)} P'$, now *free* input transitions $P \xrightarrow{xy} P'$ are considered; the meaning of the latter is that P receives the name y along x and in doing so becomes P'. The actual SOS definition of the early transitional semantics is obtained by replacing the input case of `Inp` , `Com` and `Close` of table Table 2.1 with the following three rules:

$$\texttt{E-Inp} \;:\; \frac{-}{x(y).P \xrightarrow{xw} P\{w/y\}}$$

$$\texttt{E-Com} \;:\; \frac{P_1 \xrightarrow{xw} P_2' \; P_2 \xrightarrow{\overline{x}w} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

and

$$\texttt{E-Close} \;:\; \frac{P_1 \xrightarrow{xw} P_2' \; P_2 \xrightarrow{\overline{x}(w)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (w)(P_1' \mid P_2')} \quad w \notin fn(P_1)$$

We shall often drop the `E-` prefix in the name of these rules when it is clear that we are referring to the early system. We let μ[1] range over the set of early actions and denote with

[1] not to be confused with α, that ranges over prefixes; note that prefixes and actions are the same in the late transitional semantics.

$\xrightarrow{\mu}_e$ the early transition relation (the subscript $_e$ will be dropped when no confusion can arise). We extend the functions bn and fn to early actions by letting: $fn(xw) = \{x, w\}$ and $bn(xw) = \emptyset$. The relationship between early and late transitional semantics can be stated as follows (see [MPW91]):

- $P \xrightarrow{xw}_e P'$ if and only if $P \xrightarrow{x(y)} P''$, with $P' = P''\{w/y\}$;
- $P \xrightarrow{\alpha}_e P'$ if and only if $P \xrightarrow{\alpha} P'$.

It is worth considering early transitional semantics because it naturally leads to a notion of *early* bisimulation, which we will consider in Section 2.3.

2.2 The Polyadic π-calculus

2.2.1 Syntax

In the language we have been considered so far, input and output channels carry exactly one name. For this reason the language is often called *monadic* π-calculus. The *polyadic* π-calculus [Mil91] is obtained by removing this restriction, thus allowing channels to carry *tuples* of names, of any length including zero. Accordingly, we replace the monadic input and output prefixes of Section 2.1 with two new operators, *polyadic input* and *polyadic output* prefixes:

$$a(\widetilde{b}).P \qquad \text{and} \qquad \overline{a}\langle\widetilde{b}\rangle.\,P$$

where $\widetilde{b}$ denotes a tuple of names, which, in the input case, are assumed to be pairwise distinct. The remaining operators are the same as the monadic ones. In the sequel, we use the following notations. Brackets surrounding tuples of length zero or one in prefixes are often omitted. For the restriction operator, we use the notation $\nu\, b$ (instead of (b)) and also abbreviate $\nu\, c_1 \cdots \nu\, c_n$ as $\nu\,\widetilde{c}$ or $(\nu\,\widetilde{c})$, where $\widetilde{c}$ is the tuple $(c_1 \cdots c_n)$. Conventions about binding of names and α-conversion over processes and actions are the same as before.

2.2.2 Sorting

Virtually all π-calculus processes described in the literature obey some discipline in the use of names. The introduction of sorts and sortings into the π-calculus [Mil91] intends to make this name discipline explicit. In the polyadic π-calculus, sorts are also essential to avoid disagreement in the arities of tuples carried by a given name, or applied to a given constant. Below, we review the definition of sorting and of well-sorted process.

Names are partitioned into a collection of *subject sorts*, each of which contains an infinite number of names. We write $a : s$ to mean that name a belongs to the subject sort s; this notation is extended to tuples componentwise. Then *object sorts*, are just tuples of subject sorts, such as $(s_1, \ldots, s_n)$ or (s). Finally, a *sorting* is a function Ob mapping each subject sort onto an object sort. We write $s \mapsto (\tilde{s}) \in Ob$, if Ob assigns the object sort $(\tilde{s})$ to s. By assigning the object sort (s_1, s_2) to the subject sort s, one forces the tuples carried by any name in s to be a pair whose first component is a name of s_1 and whose second component is a name of s_2. Thus, a prefix $a(\widetilde{b}).\,P$ or $\overline{a}\langle\widetilde{b}\rangle.\,P$ is *well-sorted* for a sorting Ob if $a : s$ and $s \mapsto (\widetilde{s}) \in Ob$ imply $\widetilde{b} : \widetilde{s}$. A process P is *well-sorted* for Ob if all input and output prefixes in P are well-sorted. The sorting system also affects substitutions, which are only defined between names of the same sort. In the remainder of the thesis, all polyadic processes are supposed to be well-sorted for some sorting Ob.

2.2.3 Operational semantics

We consider here only the early version of operational semantics. Similar definitions and considerations hold for the late version. Besides silent transitions, a polyadic π-calculus process has two possible forms of transition. *Input* and *output transition* are, respectively, of the form

$$P \xrightarrow{a\langle\widetilde{b}\rangle} P' \quad \text{and} \quad P \xrightarrow{(\nu\,\widetilde{b'}\,)\overline{a}\langle\widetilde{b}\rangle} P'\,.$$

In both cases, the action occurs at a. In the input action, $\widetilde{b}$ is the tuple of names which are received. In the output action, $\widetilde{b}$ is the tuple of names which are emitted, and $\widetilde{b'} \subseteq \widetilde{b}$ are private names which are carried out from their current scope. The angled brackets in the input action are to stress that names $\widetilde{b}$ represent the *values* communicated (since we are in an early setting), similarly to their role in an output action $\overline{a}\langle\widetilde{b}\rangle$, as opposed to *binders*, which are found in an input prefix $a(\widetilde{b}).\,P$.

All names in an input action are *free*. In an output action $(\nu\,\widetilde{b'}\,)\overline{a}\langle\widetilde{b}\rangle$, names $\widetilde{b'}$ are *bound*, the remaining ones free. Bound and free names of an action μ, respectively written $bn(\mu)$ and $fn(\mu)$, are defined accordingly. The *names* of μ, briefly $n(\mu)$, are $bn(\mu) \cup fn(\mu)$.

We come to the actual SOS of the polyadic π-calculus. Reminding that we are working up to α-equivalence, the two new action prefixes are given the following rules:

$$\texttt{Inp} \;:\; \frac{-}{a(\widetilde{b}).P \xrightarrow{a\langle\widetilde{c}\rangle} P\{\widetilde{c}/\widetilde{b}\}} \qquad \text{and} \qquad \texttt{Out} \;:\; \frac{-}{\overline{a}\langle\widetilde{b}\rangle.\,P \xrightarrow{\overline{a}\langle\widetilde{b}\rangle} P}\,.$$

The rules `Open` and `Com` of Table 2.1 are changed as follows:

$$\texttt{Open} \;:\; \frac{P \xrightarrow{(\nu\,\widetilde{b'})\overline{a}\widetilde{b}} P'}{\nu\,c\,P \xrightarrow{(\nu\,\widetilde{b'c})\overline{a}\widetilde{b}} P'} \; c \neq a, c \in \widetilde{b} - \widetilde{b'}$$

and

$$\texttt{Com} \;:\; \frac{P_1 \xrightarrow{(\nu\,\widetilde{b'})\overline{a}\widetilde{b}} P_1' \; P_2 \xrightarrow{a\langle\widetilde{b}\rangle} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} \nu\,\widetilde{b'}\,(P_1' \mid P_2')} \; \widetilde{b'} \cap fn(P_2) = \emptyset\,.$$

The `Close` rule is suppressed, because it is subsumed by `Com` (it is the case when $|\widetilde{b}| = |\widetilde{b'}| = 1$). The remaining rules remain formally the same as the monadic ones. Note that the above rules are really extensions of the monadic ones: the latter are indeed obtained when considering tuples of length one.

The importance of the polyadic π-calculus lies also in the fact that any *higher-order* extension of the π-calculus can be encoded in it. In these extensions, names can carry tuples of *processes* as well as their *abstractions* (functions taking processes to processes). As an example, Thomsen's process-passing calculus "Plain CHOCS" [Tho89] may be viewed as a second-order extension of the π-calculus. In his PhD. thesis [San92], Davide Sangiorgi has shown that all higher-order extensions can be faithfully encoded in the polyadic π-calculus.

We shall consider the polyadic π-calculus in Chapter 5, where the fact that many channels can be passed at once among processes will be crucial when proving that a non-interleaving equivalence can be encoded into an interleaving one.

2.3 Bisimulation-based Semantics

We fix here some definitions of bisimulation-based semantics, which will be used several times throughout the thesis. The theory of bisimulation for the π-calculus is becoming now well-understood, thanks to the works by Milner, Parrow, Sangiorgi and Walker [MPW92, MPW91, PS93]. We refer the reader to cited papers for additional definitions and a more comprehensive discussion on the meaning of the different equivalences defined below.

We will first consider 'strong' equivalences. Early and late bisimulation equivalences will be defined on the top of the corresponding transitional semantics. These equivalences give rise to two distinct congruences. Finally, the 'weak' versions of these relations will be considered.

Definition 2.3.1 (Late bisimulation) *A symmetric relation $\mathcal{R}$ over processes is a* ground late bisimulation *iff whenever $(P,Q) \in \mathcal{R}$ and $P \xrightarrow{\alpha} P'$, with $bn(\alpha) \cap fn(P,Q) = \emptyset$, then*

- *if α is not an input action, then $\exists\, Q'$: $Q \xrightarrow{\alpha} Q'$ and $(P',Q') \in \mathcal{R}$.*
- *if $\alpha = x(y)$, then $\exists Q'$: $Q \xrightarrow{\alpha} Q'$ and $\forall\, z \in fn(P,Q,y)$ $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$.*

Two processes P and Q are late strongly bisimilar *written $P \sim_l Q$, if $(P,Q) \in \mathcal{R}$, for some ground late strong bisimulation $\mathcal{R}$.*

We have only considered monadic actions here; the extension of the above definition to the polyadic case is straightforward. The early equivalence enjoys a more uniform definition, that can be stated as follows:

Definition 2.3.2 (Early bisimulation) *A symmetric relation* $\mathcal{R}$ *is an* ground early strong bisimulation *if* $(P,Q) \in \mathcal{R}$ *and* $P \xrightarrow{\mu}_e P'$ *with* $bn(\mu) \cap fn(P,Q) = \emptyset$ *imply that there exists* Q' *s.t.* $Q \xrightarrow{\mu}_e Q'$ *and* $(P',Q') \in \mathcal{R}$. *Two processes* P *and* Q *are* early strongly bisimilar, *written* $P \sim_e Q$, *if* $(P,Q) \in \mathcal{R}$, *for some ground early strong bisimulation* $\mathcal{R}$.

It is remarkable that early bisimulation can be defined on top of the *late* transitional semantics as well; it is enough to change the input-clause (the second one) of Definition 2.3.1 by a weaker clause, thus obtaining the following:

Definition 2.3.3 (Early bisimulation - alternative formulation) *A symmetric relation* $\mathcal{R}$ *over processes is a* ground early bisimulation *iff* $(P,Q) \in \mathcal{R}$ *and* $P \xrightarrow{\alpha} P'$ *with* $bn(\alpha) \cap fn(P,Q) = \emptyset$, *imply*

- *if* α *is not an input action, then* $\exists\, Q'$: $Q \xrightarrow{\alpha} Q'$ *and* $(P',Q') \in \mathcal{R}$.
- *if* $\alpha = x(y)$, *then* $\forall\, z \in fn(P,Q,y)\ \exists Q'$: $Q \xrightarrow{\alpha} Q'$ *and* $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$.

Two processes P *and* Q *are early strongly bisimilar, written* $P \sim_e Q$, *if* $(P,Q) \in \mathcal{R}$, *for some ground early strong bisimulation* $\mathcal{R}$.

It can be easily proven (see [MPW91]) the two definitions are equivalent, in the sense that they give rise to the same equivalence. We shall often refer to Definition 2.3.3 (particularly in Chapters 3 and 4), because it better evidences similarities and the differences of early and late bisimulations. In [MPW92], it is shown that late bisimulation is strictly finer than the early one.

We shall often omit the subscripts ($_l$ and $_e$) when it is clear from the context which kind of bisimulation we are referring to. In what follows, $\sim$ stands for either $\sim_l$ or $\sim_e$.

Relation $\sim$ is preserved by all operators but input prefix, for which, however, a special inference rule is valid:

$$\text{if, for each } x,\ P\{x/y\} \sim Q\{x/y\} \text{ then } a(y).P \sim a(y).Q.$$

(A similar inference rules holds for the polyadic case: just replace y by $\widetilde{y}$ and x by $\widetilde{x}$.)

We sometimes also use the largest *congruence* contained in $\sim$, denoted by $\sim^c$, and defined as follows:

$$P \sim^c Q \text{ if and only if, for each name substitution } \sigma,\ P\sigma \sim^c Q\sigma.$$

Table 2.2 lists a few simple algebraic laws which are valid for $\sim^c$ (and hence for the relations coarser than it, such as $\sim$). For $\sim$, an expansion law similar to that used in CCS [Mil89] is valid; details can be found in [MPW92] and [PS93].

Abelian monoid laws for $+$:		$P_1 + P_2$	$\sim^c$	$P_2 + P_1$
		$P_1 + (P_2 + P_3)$	$\sim^c$	$(P_1 + P_2) + P_3$
		$P + \mathbf{0}$	$\sim^c$	P
Abelian monoid laws for $\mid$:		$P_1 \mid P_2$	$\sim^c$	$P_2 \mid P_1$
		$P_1 \mid (P_2 \mid P_3)$	$\sim^c$	$(P_1 \mid P_2) \mid P_3$
		$P \mid \mathbf{0}$	$\sim^c$	P
Restriction:		$\nu\, a\, \mathbf{0}$	$\sim^c$	$\mathbf{0}$
		$\nu\, a\, \nu\, b\, P$	$\sim^c$	$\nu\, b\, \nu\, a\, P$
	if $a \notin fn(P_2)$, then	$(\nu\, a\, P_1) \mid P_2$	$\sim^c$	$\nu\, a\, (P_1 \mid P_2)$
Replication:		$!\, P$	$\sim^c$	$P \mid !\, P$

Table 2.2: A few laws for strong bisimulation congruence ($\sim^c$)

Remark 2.3.4 The third law for restriction in Table 2.2 allows us to extrude the scope of a restriction across a parallel composition, provided that the restricted name is fresh. Since we work up-to alpha conversion, we shall apply the law without recalling this side condition. □

Thus, we have *four* basic strong bisimulation-based equivalences: early and late strong bisimilarity and the congruences they induce.

The extension of the equivalences previously defined to the weak case is straightforward. As an example, we give the definition of *weak early ground bisimilarity*, which is the only one we will actually use. As usual, the 'weak arrow' $\Longrightarrow$ is the reflexive and transitive closure of $\longrightarrow$. We use $P \stackrel{\widehat{\mu}}{\Longrightarrow} Q$ to mean: $P \stackrel{\mu}{\Longrightarrow} Q$, if $\mu \neq \tau$, and $P \Longrightarrow Q$, if $\mu = \tau$.

Definition 2.3.5 (Observation equivalence) *A symmetric relation over processes is a* weak bisimulation *(or a* $\approx$-bisimulation*) if $(P, Q) \in \mathcal{R}$ and $P \stackrel{\mu}{\Longrightarrow} P'$, with $bn(\mu) \cap fn(P, Q) = \emptyset$, imply that there exists Q' s.t. $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$. Two processes P and Q are* (early) observationally equivalent*, (or simply* weakly bisimilar*), written $P \approx Q$, if $(P, Q) \in \mathcal{R}$, for some weak bisimulation $\mathcal{R}$.*

Observation equivalence is preserved by all operators except input prefix and sum; for input prefix, a congruence rule similar to that stated for $\sim$ is valid.

We shall consider weak bisimulation in Chapter 5.

2.4 An Example

To end the chapter, we present an example, taken from the operating systems world. It shows the benefits of name-passing w.r.t. efficiency and modularity. A scheduler S schedules the requests of a set of processes $P_1, \ldots, P_n$ that want to access some input devices. Each request concerns a specific device. S receives a request through a channel req, asks for a datum to the appropriate input device and finally gives the datum back to the process that had requested it. Furthermore, several requests can be served in parallel. We want the system to be *modular*. Namely, it should be possible to write the scheduler knowing in advance little of the processes P_i's (for example, not their number) and of the input devices. In other words, it should be possible adding to the system more processes and input devices at any moment, without having to re-write the program of any of the involved processes.

Below, we present a solution that rely on name-passing to naturally achieve the wanted parallelism and modularity. Each input device D is accessible through a specific channel d. The generic P that wants to access D, sends to S the channel d together with a private name x (that will act as return address) and then waits for an answer along x; S asks (through d) D for the datum and then sends it back, via the private x, to P. The fact that x is private to S and P guarantees that the answer is not "intercepted" by another process. This makes it possible different requests to be served in parallel. Here is the definition of S and of the generic requesting process P_i, $i = 1, 2, \ldots$:

$$\begin{aligned} S &\stackrel{def}{=} \; !\, req(d, x).d(b).\overline{x}b \\ P_i &\stackrel{def}{=} \; \nu\, x\, \overline{req}\langle d_i, x\rangle.\, x(b).R_i, \quad x \notin fn(R_i) \cup \{d_i, req\}. \end{aligned}$$

Recall that $!Q$ stands intuitively for $Q \,|\, Q \,|\, Q \cdots$; the use of this operator in the definition of S allows unboundedly many requests to be served at the same time.

In Table 2.3, a possible evolution of a system composed by S and two requesting processes, $Sys \stackrel{def}{=} (\nu\, req)(S \mid P_1 \mid P_2)$ is showed. We use the notation $R_i(b)$ to stress that the behaviour of R_i depends on b and abbreviate req as r; furthermore, we use simple algebraic manipulations of terms, such as commutativity of parallel composition and the laws $\nu\, x\, Q = Q$ and $\nu\, x\, (Q'|Q) = \nu\, x\, (Q')|Q$, if $x \notin fn(Q)$. These are self-evident laws that are valid in any reasonable behavioural equivalence. Each step of the evolution is followed by a short comment.

Note that a solution to this problem using static process algebras (for example the value-passing CCS of [Mil80]) would be very difficult. The requirements of modularity would be hardly met. In particular:

- one would need to know in advance the channel names of the input devices and the (distinct) return addresses of the requesting processes, and

Sys	$\xrightarrow{\tau}$	$\nu\, r\, (\nu\, x\, (d_1(b).\overline{x}b \mid x(b).R_1(b))\vert P_2\vert S)$ Interaction of P_1 and S at r triggers a copy of S
	$\xrightarrow{d_1\langle b'\rangle}$	$\nu\, r\, (\nu\, x\, (\overline{x}b' \mid x(b).R_1(b))\vert P_2\vert S)$ A datum b' is received from d_1
	$\xrightarrow{\tau}$	$\nu\, r\, (\nu\, x\, (\overline{x}b' \mid x(b).R_1(b))\vert \nu\, x\, (d_2(b).\overline{x}b \mid x(b).R_2(b))\vert S)$ Also P_2 interacts with S; a new copy of S is triggered
	$\xrightarrow{d_2\langle b''\rangle}$	$\nu\, r\, (\nu\, x\, (\overline{x}b' \mid x(b).R_1(b))\vert \nu\, x\, (\overline{x}b'' \mid x(b).R_2(b))\vert S)$ A datum b'' is received from d_2
	$\xrightarrow{\tau}$	$\nu\, r\, (R_1(b')\vert \nu\, x\, (\overline{x}b'' \mid x(b).R_2(b))\vert S)$ The datum b' is communicated to the requesting process through the private x; after that, restriction $\nu\, x$ is discarded from R_1
	$\xrightarrow{\tau}$	$\nu\, r\, (R_1(b')\vert R_2(b'')\vert S)$ The datum b'' is communicated to the requesting process through the private x; after that, restriction $\nu\, x$ is discarded from R_2.

Table 2.3: A possible evolution of the system $Sys \stackrel{def}{=} (\nu\, req)(S \mid P_1 \mid P_2)$

- if adding a new process, one would need would need to change the program of the scheduler accordingly and to analyze the whole system to ensure that the return address of the new process is distinct from every old return address.

Here, the first problem is avoided thanks to mobility (name-passing) and the second one thanks to the dynamic scoping of the restriction operator, that models the creation and transmission of new names.

Chapter 3

Testing Semantics

The aim of this chapter is to investigate applicability of the testing approach to a name-passing process algebra. Its main contributions are an axiomatization of a *weak* (i.e. abstracting from internal moves) testing-based behavioural relation over the monadic π-calculus and a fully abstract denotational model. The new semantics is defined by following the general approach of [DH84, DeN87, Hen88], that requires formally defining a *set of observers, a way of observing* and a *general criterion for interpreting results of observations*. If we call $\mathcal{S}$ the set of systems we want to experiment upon, to apply the testing setting, we need to define a set of observers, say $\mathcal{O}$, and explain the evolution of pairs such as $(P, o) \in \mathcal{S} \times \mathcal{O}$ that will represent the interactions between P and o. Interactions may be failing or successful, depending on whether particular states (which *report success*) are reached. For specific process P and observer o, one might be interested in knowing whether a successful interaction does exists, i.e. whether P **may satisfy** o, or whether all interactions are successful, i.e. whether P **must satisfy** o. Two testing preorders over $\mathcal{S}$, naturally arise, associated with each of the above points of view, the *may* preorder and the *must* preorder:

- $P \sqsubseteq\!\!\!\!\sim_{may} Q$ if and only if for each $o \in \mathcal{O}$:
  P **may satisfy** o implies Q **may satisfy** o;

- $P \sqsubseteq\!\!\!\!\sim_{must} Q$ if and only if for each $o \in \mathcal{O}$:
  P **must satisfy** o implies Q **must satisfy** o.

These preorders can be combined to get a third one, called the *testing* preorder:

- $P \sqsubseteq\!\!\!\!\sim Q$ if and only if $P \sqsubseteq\!\!\!\!\sim_{may} Q$ and $P \sqsubseteq\!\!\!\!\sim_{must} Q$.

In [DH84], the above machinery is applied to CCS, and the view is taken that *observers should be processes themselves*, which interact with other processes by *communicating* with them. This gives rise to three testing preorders with simple axiomatic and denotational characterizations.

Applying this framework to the π-calculus is harder; difficulties are mainly due to the distinction between input and output actions and to the specific instantiation scheme of input actions. Additional complications are due to the presence of two distinct kinds of output actions. In particular, a π-calculus process can export public names (*free outputs actions*), as well as private ones (*bound outputs actions*) and thus it has, in general, different options when prompted for an output.

To fully discriminate between different behaviours, observers are needed which can test whether two received names are the same or not. This has called for an extension of the experimenters language; the original π-calculus has hence been enriched with a *mismatch* operator, that together with the original *match* operator gives us the needed observational power. The relevance of mismatch to the π-calculus is currently debated. Recently, it has been used in [PS93] to provide axiomatizations for various bisimulation-based equivalences. Here, we establish the importance of mismatch from an observational point of view. We will see that the new operator is necessary to obtain "reasonable" testing equivalences: its omission from the calculus would affect in a counter-intuitive way the testing power of our observers.

The rest of the chapter is organized as follows. In Section 3.1, the variant of the π-calculus we use is introduced, together with a presentation of the testing approach and its application to the π-calculus. Evidence is given of the importance of the mismatch operator and the impact of testing on different variants of the transitional semantics is studied. In Section 3.2, a sound and complete (w.r.t. a testing preorder) proof system for the finite calculus is introduced. It consists of two groups of laws: one containing (essentially) the laws for the strong observational equivalence of [MPW92], the other dealing with internal actions and nondeterminism. In Section 3.3, a fully abstract denotational model for the finite π-calculus is presented. The model is based on an algebraic cpo, whose compact elements are represented by (equivalence classes of) process normal forms. The generalization to the full calculus is described in Section 3.4. There, the π-calculus with recursion is presented and a complete proof system and a fully abstract denotational model are worked out as extensions of those for the finite language. All detailed proofs are reported in Section 3.5.

3.1 A Testing Framework for the π-calculus

3.1.1 Preliminaries

Let $\mathcal{P}_1$ be the language generated by the following syntax:

$$P ::= \mathbf{0} \ \mid\ \Omega \ \mid\ \alpha.P \ \mid\ (y)P \ \mid\ [x=y]P \ \mid\ [x \neq y]P \ \mid\ P+P \ \mid\ P\,|\,P$$

where α ranges over *Act* as usual. Note that for the moment we confine ourselves to finite agents, i.e. consider a language without recursion, but, w.r.t. the language of [MPW92], add the two operators of mismatch $[x \neq y]$ and divergence Ω. The reason for doing so will become clear in the sequel. The transitional semantics for this language is the late one presented in Sections 2.1.

It is worth noticing that match and mismatch, together with sum, permit easily expressing the if-then-else construct; indeed $[x = y]P + [x \neq y]Q$ corresponds to `if` $x = y$ `then` P `else` Q.

Notations

Besides that introduced in Chapter 2, we fix some more notation for this chapter:

- $P[+R]$ denotes an agent expression where the summand R is optional;
- $[P]$ denotes an agent expression that may have a certain number ($n \geq 0$) of restrictions at its top level, i.e. $[P]$ stands for $(y_1)...(y_n)P$;
- $[y \notin Y][\,.\,]$ will be a shorthand for the context $[y \neq y_1] \cdots [y \neq y_k][\,.\,]$, where Y is a finite set of names, $Y = \{y_1, ..., y_k\}$, $k \geq 0$; if $Y = \emptyset$, $[y \notin Y]$ will denote the empty context $[\,.\,]$;
- $P|Q \xrightarrow{c} [P'|Q']$ denotes a communication between P and Q, i.e. a transition $P|Q \xrightarrow{\tau} [P'|Q']$ whose derivation (from the inference rules) contains an application of the `Com` or `Close` rule involving P and Q; we will write $P|Q \stackrel{c}{\Longrightarrow} [P'|Q']$ for $P|Q \Longrightarrow \xrightarrow{c} [P'|Q']$.

3.1.2 Basic definitions

We set up now a few definitions that permit applying the testing machinery to the π-calculus. In particular, we will first introduce observers, then experiments and finally testing preorders. From now on, we will then use the word *agent* to mean either process or observer or experiment.

Definition 3.1.1 (Observers)

- *The set $\mathcal{O}$ of* observers *is defined like $\mathcal{P}_1$, but an additional distinct action $\omega \notin Act$ is used to report success.*
- *We let $Act' = Act \cup \{\omega\}$, $\Lambda' = \Lambda \cup \{\omega\}$, $bn(\omega) = fn(\omega) = \emptyset$, $subj(\omega) = obj(\omega) = -$. The actions $\overline{x}y$ (*free output*), τ and ω are* free actions, *the remaining ones are* bound actions.
- *The operational semantics for $\mathcal{P}_1$ extends to $\mathcal{O}$ by adding the obvious rule: $\omega.o \xrightarrow{\omega} o$.*
- *Unless otherwise stated, $\mathcal{O}$ is ranged over by the metavariables $o, o', \ldots$.*

We recall that square brackets [] around a term indicate that it might be prefixed by some restrictions.

Definition 3.1.2 (Experiments) *The set $\mathcal{E}$ of* experiments *is defined as:*

$$\{[P|o] \mid P \in \mathcal{P}_1,\ o \in \mathcal{O}\}.$$

Metavariables $e, f, \ldots$ range over $\mathcal{E}$.

Definition 3.1.3 (Interactions) *Given an experiment $[P|o]$, an* interaction *(or* computation*) from $[P|o]$ is a maximal sequence of τ transitions, i.e. a sequence of τ-transition:*

$$[P|o] \xrightarrow{\tau} [P_1|o_1] \xrightarrow{\tau} \ldots \xrightarrow{\tau} [P_k|o_k]$$

with $k \geq 0$ and $[P_k|o_k] \stackrel{\tau}{\not\longrightarrow}$.

It is worth noting that, since we are considering the finite sub-language, all computations are finite.

The next definition formalizes the concept of partially divergent agent. Here, the term Ω represents the 'totally divergent' or 'totally undefined' agent, i.e. it models the agent that computes an infinite sequence of internal actions, without ever interacting with the external environment. In spite of the fact that we have not yet considered recursion and infinitary behaviours, we have already extended our language with Ω, because this will allow us to smoothly generalize our theory to the full language in Section 3.5. Furthermore, Ω will be needed when giving the denotational semantics to the finite fragment we are considering: it will represent the *bottom* of our model.

Definition 3.1.4 (Definedness and convergence predicates)

- *$\downarrow$ is the least (postfix) predicate over $\mathcal{O}$ which satisfies:*

- $\mathbf{0}\downarrow$, $(\alpha.P)\downarrow$, $([x \neq x]P)\downarrow$, $x \neq y$ *implies* $([x = y]P)\downarrow$, $([x \neq x]P)\downarrow$;
- $P\downarrow$ *and* $Q\downarrow$ *imply:* $((y)P)\downarrow$, $([x = y]P)\downarrow$, $([x \neq y]P)\downarrow$, $(P|Q)\downarrow$ *and* $(P+Q)\downarrow$.

We write $P\uparrow$ *if it is not the case that* $P\downarrow$.

- $\Downarrow$ *is the least (postfix) predicate over* $\mathcal{O}$ *which satisfies:*
  $P\Downarrow$ *iff* $P\downarrow$ *and for every* P', $P \Longrightarrow P'$ *implies* $P'\Downarrow$.
  We write $P\Uparrow$ *if it is not the case that* $P\Downarrow$.

It is easy to prove that $P\uparrow$ if and only if P contains a *unguarded* occurrence of Ω; i.e. a Ω not inside inside a context of the kind $\alpha.[\,.\,]$, or $[x = y][\,.\,]$ with $x \neq y$, or $[x \neq x][\,.\,]$.

Definition 3.1.5 (May and Must relations)

- P **may** o *if and only if there exists an interaction* $P|o \xrightarrow{\tau} \ldots \xrightarrow{\tau} [P_k|o_k]$ *such that for some* i, $0 \leq i \leq k$, $o_i \xrightarrow{\omega}$.
- P **must** o *if and only if for each interaction* $P|o \xrightarrow{\tau} \ldots \xrightarrow{\tau} [P_k|o_k]$ *there exists* j, $0 \leq j \leq k$, *such that* $o_j \xrightarrow{\omega}$ *and* $\forall i \leq k$, $((P_i|o_i)\uparrow$ *implies* $i \geq j)$.

Definition 3.1.6 (Testing preorders)

- $P \sqsubseteq_{may} Q$ *if and only if for every* $o \in \mathcal{O}$ *we have:*
  P **may** o *implies* Q **may** o.
- $P \sqsubseteq_{must} Q$ *if and only if for every* $o \in \mathcal{O}$ *we have:*
  P **must** o *implies* Q **must** o.
- $P \sqsubseteq Q$ *if and only if* $P \sqsubseteq_{may} Q$ *and* $P \sqsubseteq_{must} Q$.

The negations of P **may** o and P **must** o will be written as P $\not\!\mathbf{may}$ o and P $\not\!\mathbf{must}$ o, respectively. As it might be expected, while $\sqsubseteq_{may}$ is preserved by the $+$ operator, $\sqsubseteq_{must}$ (hence $\sqsubseteq$) is not. A standard way of tackling this kind of problem is "closing" the $\sqsubseteq_{must}$ relation with respect to summation (see [DH84, Mil89]). An alternative characterization of the new testing preorders can also be given in terms of the original preorders and the initial invisible actions of agents. This leads to the following:

Definition 3.1.7 (Revisited testing preorders)

- $P \sqsubseteq^{+}_{must} Q$ *if and only if* $(P \sqsubseteq_{must} Q$ *and* $(P \Uparrow$ *or* $(Q \xrightarrow{\tau}$ *implies* $P \xrightarrow{\tau})))$;
- $P \sqsubseteq^{+} Q$ *if and only if* $(P \sqsubseteq_{may} Q$ *and* $P \sqsubseteq^{+}_{must} Q)$.

It can be easily shown that the above preorders coincide with those generated by the closure under $+$ of the original preorders; i.e.:

- $P \sqsubseteq^{+}_{must} Q$ if and only if for each R, $P + R \sqsubseteq_{must} Q + R$;
- $P \sqsubseteq^{+} Q$ if and only if for each R, $P + R \sqsubseteq^{+} Q + R$.

Thus, the above definition provides us with preorders which are substitutive for the sum operator; however, differently from CCS, we have that none of the relations $\sqsubseteq_{may}$, $\sqsubseteq^{+}_{must}$, $\sqsubseteq^{+}$ is a pre-congruence; indeed, they are preserved by all operators, but input prefix. The failure for the latter operator is proved by the following counter-example.

Let x, y, u, w, z be distinct names in $P \stackrel{def}{=} [x = y]\overline{x}w.\mathbf{0}$ and $Q \stackrel{def}{=} [x = z]\overline{x}w.\mathbf{0}$. We have $P \sqsubseteq_{may} Q$, because both P and Q are trivially equivalent to $\mathbf{0}$. On the other hand we have that $u(y).P$ is not equivalent $u(y).Q$. Indeed, if we take $o \stackrel{def}{=} \overline{u}x.x(w).\omega.\mathbf{0}$ we have:

$$(u(y).P)|o \xrightarrow{\tau} [x = x]\overline{x}w.\mathbf{0}|x(w).\omega.\mathbf{0} \xrightarrow{\tau} \mathbf{0}|\omega.\mathbf{0} \xrightarrow{\omega} \mathbf{0}|\mathbf{0}$$

thus $u(y).P$ **may** o. Conversely, the only possible computation from $(u(y).Q)|o$ is:

$$(u(y).Q)|o \xrightarrow{\tau} [x = z]\overline{x}w.\mathbf{0}|x(w).\omega.\mathbf{0} \not\xrightarrow{\tau}$$

and thus $u(y).Q$ $\cancel{\textbf{may}}$ o.

Similar counter-examples can be exhibited for $\sqsubseteq^{+}_{must}$ and $\sqsubseteq^{+}$. As we saw in Chapter 2, Section 2.3, this kind of problem for the input prefix arises also for bisimulation. As in [MPW92], also for our testing preorders a sort of congruence law for the input prefix operator can be given. Equivalence of two given processes P and Q is preserved by prefix $u(y).[\,.\,]$ provided that it is preserved by all substitutions $\{w/y\}$, for any name $w \in fn(P) \cup fn(Q) \cup \{y\}$. Indeed, this will be a rule of our proof system; since we have a finite number of names to test (as $fn(P) \cup fn(Q)$ is finite), this is still effective (although not very 'efficient').

3.1.3 Testing without mismatch

In this subsection we attempt an assessment of the testing power of observers when the mismatch operator is discarded from the language. We will denote by $\sqsubseteq^{-}_{may}$, $\sqsubseteq^{-}_{must}$ and $\sqsubseteq^{-}$ the testing preorders obtained by permitting only observers without mismatch and by $\simeq^-_{may}$, $\simeq^-_{must}$ and $\simeq^-$ the corresponding equivalences. We will see that each of these relations is different from those of the previous subsection (obtained by considering also observers that may contain the mismatch operator). Pairs of processes will be exhibited that can be fully discriminated by using observers with mismatch, but cannot be taken apart without mismatch. The counterexamples will make it evident that the testing relations generated without mismatch are counter-intuitive.

We begin with exhibiting two pairs of processes that show that the two may and the two must preorders are indeed different; this result is then extended to the equivalences. The key proposition is:

Proposition 3.1.8 *Let x and y be any two names. Then:*

a. $\overline{x}(w).\mathbf{0} \sqsubseteq\!\!\!\!\sim^-_{may} \overline{x}y.\mathbf{0}$.

b. $\overline{x}(w).\Omega \sqsubseteq\!\!\!\!\sim^-_{must} \overline{x}(w).\mathbf{0} + \overline{x}y.\Omega$.

Proof: See Section 3.5, Proposition 3.5.5. □

The above proposition holds essentially because, in absence of mismatch, all transitions that can be performed by an agent upon receipt of a bound (hence 'new') name, can also be performed upon receipt of any name. This is no more possible if mismatch is introduced; this operator permits additional discriminations between free and bound names. Indeed, within the richer language, the observer $o \stackrel{def}{=} x(z).[z \neq y]\omega.\mathbf{0}$, with $z \neq y$, permits concluding:

- $\overline{x}(w).\mathbf{0} \not\sqsubseteq\!\!\!\!\sim_{may} \overline{x}y.\mathbf{0}$;
- $\overline{x}(w).\Omega \not\sqsubseteq\!\!\!\!\sim_{must} \overline{x}(w).\mathbf{0} + \overline{x}y.\Omega$.

Now, we report below three pairs of processes that are (may, must, testing) equivalent when the restricted language is considered and non-equivalent when observers with the mismatch operator are used. When proving equivalence, we will freely use, both for $\sqsubseteq\!\!\!\!\sim^-_{may}$ and for $\sqsubseteq\!\!\!\!\sim^-_{must}$, substitutivity under + - contexts[1], associativity and commutativity. We will also use the following laws, that can be easily proven sound:

$L1$	$P \simeq^- P + P$
$L2$	$P \sqsubseteq\!\!\!\!\sim^-_{may} P + Q$
$L3$	$\alpha.P + \beta.Q \sqsubseteq\!\!\!\!\sim^-_{must} \alpha.P$ if $\alpha, \beta \in OA$ and $subj(\alpha) = subj(\beta)$.

[1]All processes under consideration cannot perform invisible initial actions; and it is easy to see that when this condition holds, testing preorders are preserved by +-contexts.

- May: $\overline{x}y.\mathbf{0} \simeq^{-}_{may} \overline{x}y.\mathbf{0} + \overline{x}(w).\mathbf{0}$. Indeed:

$$\begin{array}{llll}
1.\ \overline{x}y.\mathbf{0} & \sqsubset\!\!\!\sim^{-}_{may} & \overline{x}y.\mathbf{0} + \overline{x}(w).\mathbf{0} & \text{(L2);} \\
2.\ \overline{x}y.\mathbf{0} + \overline{x}(w).\mathbf{0} & \sqsubset\!\!\!\sim^{-}_{may} & \overline{x}y.\mathbf{0} + \overline{x}y.\mathbf{0} & \text{(Prop. 3.1.8.a)} \\
 & \simeq^{-}_{may} & \overline{x}y.\mathbf{0} & \text{(L1)}
\end{array}$$

On the other hand it is easily seen that the observer o defined above suffices to take the two processes apart within the richer language.

- Must: $P \simeq^{-}_{must} P + \overline{x}y.\Omega$, where $P \stackrel{def}{=} \overline{x}(w).\Omega + \overline{x}y.\mathbf{0}$. Indeed:

$$\begin{array}{llll}
1.\ P + \overline{x}y.\Omega & \sqsubset\!\!\!\sim^{-}_{must} & P & \text{(L3)} \\
2.\ P & \simeq^{-}_{must} & P + \overline{x}(w).\Omega & \text{(L1)} \\
 & \sqsubset\!\!\!\sim^{-}_{must} & P + \overline{x}(w).\mathbf{0} + \overline{x}y.\Omega & \text{(Prop.3.1.8.b)} \\
 & \sqsubset\!\!\!\sim^{-}_{must} & P + \overline{x}y.\Omega & \text{(L3)}
\end{array}$$

It is easily seen that the observer $o' \stackrel{def}{=} x(z).([z = y]\tau.\omega.\mathbf{0} + [z \neq y]\omega.\mathbf{0})$ suffices to take the two processes apart within the richer language.

- Testing: $P \simeq^{-} P + \overline{x}y.\Omega$.

This follows from the fact that the two processes are both $\simeq^{-}_{may}$ and $\simeq^{-}_{must}$ equivalent.

These examples show that the omission of mismatch leads to identifications that do not correspond to any operational intuition. The $\simeq^{-}_{may}$ equivalence relates a process that may perform a bound output to one that may not. The $\simeq^{-}_{must}$ equivalence relates a process that after performing a free output $\overline{x}y$ always terminates to one that after the same action may diverge.

3.1.4 Early vs. late operational semantics

In this subsection the testing approach is compared with the bisimulation approach of [MPW92], especially for the differences induced by different early and late transitional semantics discussed in Chapter 2.

We would like to argue that the proliferation of equivalences, in the early and late forms, is specific of bisimulation; it arises from the interplay between quantification over names and processes in the input clauses of the definition (see Chapter 2, Section 2.3). Thus, if only the original late transitional semantics of [MPW92] is considered, within the testing approach it does not make any sense to talk about late or early equivalences.

The above claim holds even if one considers the early transitional semantics (denoted here by $\stackrel{\alpha}{\longrightarrow}_e$), that naturally leads to early bisimulation. It is very easy to see that the testing preorders over processes obtained from the early transition system ($\stackrel{\alpha}{\longrightarrow}_e$) and those obtained from the late one ($\stackrel{\alpha}{\longrightarrow}$) do coincide[2]. The key to the proof is the fact that the τ-actions of the late and the early transition systems are the same, as is stated in Chapter 2, Section 2.1. As a corollary of this fact, we have:

Corollary 3.1.9 *The testing preorders over* $\mathcal{P}_1$ *defined on the top of* $\stackrel{\alpha}{\longrightarrow}_e$ *coincide with the corresponding ones defined on the top of* $\stackrel{\alpha}{\longrightarrow}$ *.*

PROOF: The previous fact easily generalizes to sequences of τ-transitions (formally, this can be done by induction on the length of the sequences). Furthermore, clearly $P \stackrel{\omega}{\longrightarrow}_e$ iff $P \stackrel{\omega}{\longrightarrow}$. From these facts and from the definition of testing preorders, we easily obtain the thesis. $\square$

3.2 A Proof System for Finite π-calculus

In this section, we provide an inequational theory for one of the testing preorders, namely $\sqsubseteq\!\!\!\!\sim^{+}$. The extensions to $\sqsubseteq\!\!\!\!\sim^{+}_{must}$ and $\sqsubseteq\!\!\!\!\sim_{may}$ are only a matter of rephrasing the results. The axioms and the inference rules of the proof system, which we call $\mathcal{F}$, are shown in Table 3.2 and 3.2, respectively. The standard rules for preorder reflexivity and transitivity have been omitted. Each equation "$P = Q$" is an abrreviation for the pair of equations "$P \sqsubseteq Q$ and $Q \sqsubseteq P$".

Most of the axioms and rules are taken from [MPW92] and [DH84]. In particular, the Restriction, Match and Expansion laws and the inference rules are from [MPW92]; the Omega law and the τ-laws N1, N2 and N4 are from [DH84]; the τ- law N3 is a modification of the corresponding law in [DH84] to take into account the fact that a process may have different options when prompted for an output. The Mismatch laws are new. In the Expansion law, an empty summation by convention denotes **0**. By $\mathcal{F} \vdash P \sqsubseteq Q$ we mean that the relation $P \sqsubseteq Q$ is provable by applying the axioms and the inference rules of system $\mathcal{F}$. Sometimes, we abbreviate this simply as $P \sqsubseteq Q$.

The next two propositions state soundness of the inference rules and of the axioms; the actual proofs (see Section 3.5) can be done along the lines of the corresponding ones for CCS. However, specific care is needed when interactions between processes and observers are analyzed, due to the exchange of names between them. Thus the proofs of the precongruence rules for parallel composition and restriction of C0 are significantly harder than those for CCS; rule C1 was not present in the proof system for CCS.

Proposition 3.2.1 (Soundness of the inference rules) *The inference rules of Table 3.2 are sound w.r.t.* $\sqsubseteq\!\!\!\!\sim^{+}$ *.*

Proposition 3.2.2 (Soundness of the axioms) *The axioms of Table 3.2 are sound w.r.t.* $\sqsubseteq\!\!\!\!\sim^{+}$ *.*

[2] A similar result is proven in [Ing93] for a value-passing process algebra.

Sum		*Restriction*	
$S0$	$P + \mathbf{0} = P$	$R0$	$(x)P = P,\ x \notin fn(P)$
$S1$	$P + P = P$	$R1$	$(x)(y)P = (y)(x)P$
$S2$	$P + Q = Q + P$	$R2$	$(x)(P + Q) = (x)P + (x)Q$
$S3$	$P + (Q + R) = (P + Q) + R$	$R3$	$(x)\alpha.P = \alpha.(x)P,\ x \notin n(\alpha)$
		$R4$	$(x)\alpha.P = \mathbf{0},\ subj(\alpha) = x$
Match		*Mismatch*	
$M0$	$[x = y]P = \mathbf{0},\ x \neq y$	$U0$	$[x \neq x]P = \mathbf{0}$
$M1$	$[x = x]P = P$	$U1$	$[x \neq y]P = P,\ x \neq y$

Expansion

Suppose P is $\sum_{i\in I} \alpha_i.P_i[+\Omega]$ and Q is $\sum_{j\in J} \beta_j.Q_i[+\Omega]$.
Suppose that no α_i (resp. β_j) binds a name free in Q (resp. P).

$$P \mid Q = \sum_{i\in I} \alpha_i.(P_i \mid Q) + \sum_{j\in J} \beta_j.(P \mid Q_j) + \sum_{\alpha_i \ \mathbf{comp}\ \beta_j} \tau.R_{ij}$$
$$[+\Omega \mid\ \Omega \text{ is a summand either of } P \text{ or of } Q]$$

where α_i **comp** β_j and R_{ij} are defined as follows:
1. $\alpha_i = \overline{x_i}z$ and $\beta_j = y_j(y)$; then $R_{ij} = P_i \mid Q_j\{z/y\}$
2. $\alpha_i = \overline{x_i}(y)$ and $\beta_j = y_j(y)$; then $R_{ij} = (y)(P_i \mid Q_j)$
3. The "converse" of 1.
4. The "converse" of 2.

Omega

Ω $\quad \Omega \sqsubseteq P$

$\tau - laws$

$N1$ $\quad \alpha.P + \alpha.Q = \alpha.(\tau.P + \tau.Q)$
$N2$ $\quad P + \tau.Q \sqsubseteq \tau.(P + Q)$
$N3$ $\quad \alpha.P + \tau.(\beta.Q + R) = \tau.(\alpha.P + \beta.Q + R)$
with $subj(\alpha) = subj(\beta)$ and ($\alpha, \beta \in IN$ or $\alpha, \beta \in OUT$)
$N4$ $\quad \tau.P \sqsubseteq P$

Table 3.1: Axioms of the proof system $\mathcal{F}$.

$C0$	$\dfrac{P \sqsubseteq Q}{P' \sqsubseteq Q'}$	$P' \sqsubseteq Q'$ either of: $P+R \sqsubseteq Q+R, \alpha P \sqsubseteq \alpha.Q\,(\alpha \notin IN), (x)P \sqsubseteq (x)Q,$ $[x=y]P \sqsubseteq [x=y]Q, [x \neq y]P \sqsubseteq [x \neq y]Q, P\|R \sqsubseteq Q\|R.$
$C1$	$\dfrac{\forall z \in fn(P,Q,y).\ P\{z/y\} \sqsubseteq Q\{z/y\}}{x(y).P \sqsubseteq x(y).Q}$	

Table 3.2: Inference rules of the proof system $\mathcal{F}$.

The above propositions give us the following:

Theorem 3.2.3 (Soundness of $\mathcal{F}$) $\mathcal{F} \vdash P \sqsubseteq Q$ *implies* $P \sqsubsim^{+} Q$.

The completeness proof is heavily based on the existence of certain standard forms for processes called *head normal forms (hnf)*. Similar forms were already used in [DH84] (though there the completeness proof was based on normal forms). Head normal forms aim at presenting processes as an internal non-deterministic choice among a set, possibly a singleton, of *initial states* (each represented, in Definition 3.2.5, by a $P_A, A \in L$). Each initial states is essentially described by a set (A) of input or output *channels* through which the process can communicate with the environment, and by the corresponding *initial actions* (α's) and their *α-derivatives* (P_α 's). If there are several initial states (this is the case of τ-hnf's), the non-deterministic choice among them is made by performing an internal (τ) action.

In the actual definition of hnf, a functional notation is used to make it explicit that each input or output channel a is associated with a unique process $g(a)$. In case a is an input channel, say x, process $g(a)$ has a unique initial input action with subject x, i.e. it is of the form $x(y).P$. In case a is an output channel, say $\overline{x}$, $g(a)$ is a *non-empty* summation of the form $\sum_{z \in F} \overline{x}z.P_z[+\overline{x}(w).P']$, where square brackets enclose an optional summand. Head normal forms will enable us to provide a syntactical characterization of the preorders that will be crucial when proving completeness (see Section 3.5.3). This syntactical relation relies on a notion of *set saturation*; differently from [DH84], this notion applies to set of input or output channels, not to set of actions.

Definition 3.2.4 (Saturation) *Let* $IN = \mathcal{N}$, $OUT = \{\overline{x} \mid x \in \mathcal{N}\}$, $Ev = IN \cup OUT$; *Ev is the set of all* events *and is ranged over by the metavariables* a, b.

- *Given a set of sets of events* $L = \{A_1, ..., A_n\}$, *define* $Ev(L) = A_1 \cup \ldots \cup A_n$;
- *A non-empty finite set of finite sets of events L is* saturated *if:*

for each $A \subseteq Ev$*:* $(\, \exists B \in L : B \subseteq A \subseteq Ev(L)\,)$ *implies* $A \in L$*.*

Definition 3.2.5 (Head normal forms)

- *A partial function* $g : Ev \longrightarrow \mathcal{P}_1$ *is a* normal function *if for each* $a \in domain(g)$ *we have:*

  a. $a \in IN$ *implies* $g(a)$ *is of the form* $a(y).P$*, and*

  b. $a \in OUT$ *implies* $g(a)$ *is a non-empty summation of the form*

  $$\sum_{z \in F} az.P_z[+a(w).P']$$

  with $F \subseteq_{\text{fin}} \mathcal{N}$*.*

- *Let* $A \subseteq_{\text{fin}} EV$ *and let* L *be a saturated set of subsets of* EV*. An agent* P *is in* head normal form (hnf) *if one of the following conditions holds:*

  i. *(α-hnf)* $P = \sum_{a \in A} g(a)$*, where* $g : A \longrightarrow \mathcal{P}_1$ *is a normal function;*

  ii. *(τ-hnf)* $P = \sum_{A \in L} \tau.P_A$*, where* $P_A = \sum_{a \in A} g(a)$ *and* $g : Ev(L) \longrightarrow \mathcal{P}_1$ *is a normal function.*

  In both cases, function g *is referred to as the* associated function *of* P*.*

In the sequel, given a hnf P with associated function g, we will let $Ev(P)$ denote the domain of g and $Init(P)$ denote the set of initial visible actions of P, i.e. the set $\{\alpha \in \Lambda |\, P \stackrel{\alpha}{\Longrightarrow}\}$, (up to α- equivalence). It is evident that if P is a hnf then $P \Downarrow$. From the above definition, it should be also evident that for each $\alpha \in Init(P)$, there exists a unique P_α such that $P \Longrightarrow \stackrel{\alpha}{\longrightarrow} P_\alpha$; this P_α is called *α-derivative* of P. The following property of τ-head normal forms follows directly from the functional notation used in the definition:

Property 3.2.6 *Let* $P \stackrel{def}{=} \sum_{A \in L} \tau.P_A$ *be a* τ*-head normal form and let* $A \in L$*. If* $Init(P_A)$ *contains an output action with subject* x*, then* $Init(P_A)$ *contains also* all *output actions of* $Init(P)$ *with subject* x*.*

A few examples will clarify these points. Consider $P \stackrel{def}{=} x(y).\mathbf{0} + x(z).\alpha.\mathbf{0}$, with $y \notin fn(\alpha)$; P is not a hnf because we cannot associate to the input channel x a unique term $g(x)$; in particular, P has not a unique $x(y)$-derivative, as $P \stackrel{x(y)}{\longrightarrow} \alpha\{y/z\}.\mathbf{0}$ and $P \stackrel{x(y)}{\longrightarrow} \mathbf{0}$. Next, consider $Q \stackrel{def}{=} \tau.\overline{x}y.\mathbf{0} + \tau.\overline{x}z.\mathbf{0}$, with $y \neq z$; Q is not a τ-hnf, since Property 3.2.6 above does not hold for the output channel $\overline{x}$; indeed, it is not possible to define an associated normal function for Q. As an example of τ-hnf, consider

$$\tau.w(x).\mathbf{0} + \tau.(\overline{x}y.\mathbf{0} + \overline{x}z.\mathbf{0}) + \tau.(w(x) + \overline{x}y.\mathbf{0} + \overline{x}z.\mathbf{0})$$

with x, w, y, z distinct; the associated function g is given by: $g(w) = w(x).\mathbf{0}$ and $g(\overline{x}) = \overline{x}y.\mathbf{0} + \overline{x}z.\mathbf{0}$; notice that the set $L = \{\,\{w\}, \{\overline{x}\}, \{w, \overline{x}\}\}$ is saturated.

The counterpart of head normal form for divergent agents (i.e. those P's such that $P \Uparrow$ holds) is the notion of *Ω-Head Normal Form (Ω-hnf)*.

Definition 3.2.7 (Ω-Head normal forms) *An agent P is in* Ω-Head Normal Form (Ω-hnf) *if $P = (\sum_{\alpha \in A} \alpha.(P_\alpha + \Omega)) + \Omega$, for some $A \subseteq_{\mathrm{fin}} \Lambda$, A not containing distinct α-equivalent actions.*

Notice the double occurrence of Ω in the above definition; it, somehow, indicates that once the process diverges at the top-level, it does so at each inner level. If P is an Ω-hnf we let $Init(P)$ and $Ev(P)$ have the same meaning as for hnf.

We establish now a few facts about head normal forms. Their proofs (see Section 3.5) follow the same lines as the corresponding ones for CCS in [DH84]; additional complications are again introduced by the need of applying rule (C1) and, in the proof of Proposition 3.2.8, by the need of reducing P to a form satisfying Property 3.1.

Proposition 3.2.8 *If $P \Downarrow$ then there exists a head normal form $h(P)$ such that $\mathcal{F} \vdash P = h(P)$.*

Proposition 3.2.9 *If $P \Uparrow$, then there exists a Ω-head normal form $\Omega(P)$ such that $\mathcal{F} \vdash P = \Omega(P)$.*

For the proof of completeness, we need a special induction parameter, namely the largest number of communications that a process can perform with another agent. This parameter can be defined as the length, in terms of number of visible actions, of the maximal among the observers which are satisfied by the process. We remind the reader that here we have confined ourselves to finite processes, thus this number is finite.

Definition 3.2.10 (Depth of communication) *For each P we define the depth of communication of P, $dpc(P)$, as:*

$$max\{k \mid P \ \mathbf{may}\ \alpha_1 \cdots \alpha_k.\omega.\mathbf{0},\ \textit{with the}\ \alpha_i,\ 0 \leq i \leq k,\ \ \textit{ranging over}\ \Lambda\,\}.$$

Theorem 3.2.11 (Completeness of $\mathcal{F}$) *$P \sqsubseteq\!\!\!\!\sim^+ Q$ implies $\mathcal{F} \vdash P \sqsubseteq Q$.*

Proof: (Sketch; See Section 6 for a detailed proof) The proof goes by induction on $dpc(P)$. It relies on the fact that, in virtue of the above propositions, we can suppose that both P and Q are either in hnf or in Ω-hnf and on the two following properties of α-derivatives of P and Q:

- if $\alpha \in OA$ then $P_\alpha \sqsubseteq\!\!\!\!\sim^+ Q_\alpha$;
- if $\alpha = x(y) \in IA$ then $P_\alpha\sigma \sqsubseteq\!\!\!\!\sim^+ Q_\alpha\sigma$ for any name z and any substitution $\sigma = \{z/y\}$.

By applying the induction hypothesis, the above relations still hold when substituting $\sqsubseteq\!\!\!\!\sim^+$ by $\sqsubseteq$. Then, rules C0 and C1 and the τ-laws suffice to derive $P \sqsubseteq Q$. $\square$

3.3 Denotational Semantics

In this section, we sketch a denotational semantics for the π-calculus and discuss its relationships with the operational semantics of Section 3.1. We assume familiarity with the standard algebraic semantics techniques as described, for example, in [Hen88] or [Gue81]. To avoid complications which would not provide any additional insight, we confine ourselves in searching for a model which is fully abstract w.r.t. the must preorder $\underset{\sim}{\sqsubset}^{+}_{must}$ (rather than $\underset{\sim}{\sqsubset}^{+}$). A complete proof system can be obtained for this preorder from the one presented in Section 3 by adding the following three laws:

- $\Omega + P = \Omega$
- $\tau.P + \tau.Q \sqsubseteq P$
- $\alpha.P + \beta.Q \sqsubseteq \alpha.P$ if $subj(\alpha) = subj(\beta)$ and $(\alpha, \beta \in IA$ or $\alpha, \beta \in OA)$.

We will indicate with $\sqsubseteq_{must}$ the corresponding proof-theoretic preorder. The proof of completeness for this preorder is very similar to that of $\sqsubseteq$ presented in Section 3.5 and will be omitted. In fact, due to the law $\Omega + P = \Omega$, that considers all divergent processes equivalent to Ω, the actual proof is simpler than the one discussed in full detail.

We shall give the denotational model under the form of a *natural interpretation*, a slight variant of the usual algebraic semantics, introduced in [HP80] to deal with languages with explicit value passing. A natural interpretation (for the finite π-calculus) is a 4-tuple $(\mathcal{D}, F_{op}, inp, out)$, where:

- $\mathcal{D}$ is a cpo;
- F_{op} is a family of continuous functions of suitable arities, one for each operator different from input and output prefixes;
- $inp : (\mathcal{N} \times (\mathcal{N} \longrightarrow \mathcal{D})) \longrightarrow \mathcal{D}$ is a function continuous in its second argument, where $(\mathcal{N} \longrightarrow \mathcal{D})$ inherits its ordering pointwise from $\mathcal{D}$;
- $out : (\mathcal{N} \times \mathcal{N} \times \mathcal{D}) \longrightarrow \mathcal{D}$ is a function continuous in its third argument.

Given a natural interpretation, we can define a semantic function: $[\![\,.\,]\!] : \mathcal{P}_1 \longrightarrow \mathcal{D}$ as follows:

- $[\![op(P_1, ..., P_k)]\!] = f_{op}([\![P_1]\!], ..., [\![P_k]\!])$
  for each k-ary operator different from input and output, with $k = 0, 1$ or 2
- $[\![x(y).P]\!] = inp(x, \lambda z \in \mathcal{N}.[\![P\{z/y\}]\!])$
- $[\![\overline{x}y.P]\!] = out(x, y, [\![P]\!])$.

Since the considered calculus does not permit describing infinite behaviours, one might ask whether cpo's and continuous functions are really needed to model it. To obtain a fully abstract model, the partial ordering of the domain we use must properly reflect the testing preorder $\sqsubseteq^{+}_{\widetilde{must}}$. Now, although we are only dealing with finite agents, we still have elements which are approximable by infinite chains. Consider the process $P = x(y).\mathbf{0}$; P is (syntactically) finite but can be approximated by the infinite chain of processes

$$K = \{x(y).P_i |\, P_i = \sum_{z \in F_i} [y = z]\mathbf{0} + [y \notin F_i]\Omega,\ i \geq 0\}$$

where $\{F_i |\, i \geq 0\}$ is a strictly increasing chain of finite set of names such that for each j, $y \notin F_j$. Indeed, we have that

$$x(y).P_0 \sqsubseteq^{+}_{\widetilde{must}} x(y).P_1 \sqsubseteq^{+}_{\widetilde{must}} \ldots \sqsubseteq^{+}_{\widetilde{must}} x(y).\mathbf{0}$$

and these inequalities are clearly strict. We have here a kind of *continuous breadth* approximation; thus, we can expect every fully abstract model be based on a cpo rather than a poset. In fact, the construction illustrated below naturally leads us to a cpo. It is worth noting that exactly the same cpo is suitable for modelling the full calculus with recursion (see Section 3.4).

The non-trivial task to face when constructing the model is defining function $inp(.,.)$. We will take advantage of process *strong normal forms* (formally described below), that recursively reproduce the top structure of head normal forms. We will first define a poset, PO, of equivalence classes of "finite breadth" strong normal; this poset will represent the compact element of our cpo. Over PO, monotonic functions can be easily defined in a syntax-driven way; moreover, the elements of PO can be used to 'approximate' all processes. Thus, to get a natural interpretation, it will be sufficient first to define monotonic functions f_{op}, *out* and $inp(x, -)$ over PO corresponding to the operators of the calculus and then to take the ideal completion of PO and the unique continous extensions of functions f_{op}, *out* and *inp* over it. It is not hard to show that the natural interpretation so obtained is fully abstract. Below, we sketch the actual construction of the model; it takes four steps.

1. *Strong Normal Forms for all processes are introduced.*

Strong normal functions and *Strong normal forms (snf)* are defined by mutual recursion as follows:

- A partial function $g : Ev \longrightarrow \mathcal{P}_1$, is a strong normal function if for each $a \in domain(g)$ we have:

  a. if $a = x \in IN$, $g(x)$ is of the form

$$x(y).(\sum_{z \in F} [y = z]N_z + [y \notin F]N)$$

where:
1. $F \subseteq_{\mathrm{fin}} \mathcal{N}$, $y \notin F$;
2. for each $z \in F, y \notin fn(N_z)$;
3. N and all N_z, $z \in F$, are snf.

b. if $a = \overline{x} \in OUT$, $g(\overline{x})$ is a non-empty summation $\sum_{z \in F} \overline{x}z.N_z[+\overline{x}(w).N]$ where N and each N_z are snf.

- Let $A \subseteq_{\mathrm{fin}} N$ and L be a saturated set of subsets of Ev. A process P is in strong normal form if one of the following conditions holds:

  i. P is Ω;

  ii. (α-snf) P is of the form $\sum_{\alpha \in A} g(a)$, where $g : A \longrightarrow \mathcal{P}_1$ is a strong normal function;

  iii. (τ-snf) P is of the form
  $$\sum_{A \in L} \tau. \sum_{\alpha \in A} g(a)$$
  where $g : Ev(L) \longrightarrow \mathcal{P}_1$ is a strong normal function.

We call SNF the set of all strong normal forms. It can be shown, by extending the existence proof for head normal forms of Section 3.5, that each process P is provably equivalent to a snf. Differently from CCS, each process has in general syntactically different strong normal forms that cannot be reduced to each other by simply using associativity and commutativity of sum. As an example, consider, the pair of terms below, where y, w and z are distinct names and N is a snf:

$$x(y).([y = z]\mathbf{0} + [y \neq z]N) \text{ and } x(y).([y = z]\mathbf{0} + [y = w]N\{w/y\} + [y \notin \{z, w\}]N).$$

In general, to reduce two normal forms to each other, besides the above mentioned laws for sum, we would certainly need other laws to capture 'conditional renaming' like the one described above. However, our construction does not rely on uniqueness of normal forms; we work with their equivalence classes and simply assume existence of a function that associates an arbitrary, and provably must-equivalent, snf to each process.

The compact elements of our cpo will be equivalence classes of *finite strong normal forms (fsnf)*. These can be described informally as the 'finite-breadth' snf's, i.e., as those snf's that, for almost every name z, upon receipt of z become Ω. The formal definition is as follows.
A finite strong normal form is a snf N with associated function g, s.t.

a. $N = \Omega$, or

b. $x \in domain(g)$ implies $g(x)$ is of the form: $x(y).(\sum_{z \in F}[y = z]N_z + [y \notin F]\Omega)$, where each N_z is a fsnf.

2. *The set of finite strong normal form is turned into a poset.*
To guarantee antisimmetry, PO is defined as the poset given by the set of equivalence classes of fsnf's w.r.t. the kernel equivalence $=_M$ of the preorder $\sqsubseteq_{must}$ (i.e. $=_M = \sqsubseteq_{must} \cap \sqsubseteq_{must}{}^{-1}$); the partial ordering $\preceq$ over PO is defined as follows: $[N] \preceq [M]$ iff $N \sqsubseteq_{must} M$; the bottom element is $[\Omega]$. A set of functions over PO, each corresponding to a different operator of the calculus, is defined:

- For functions corresponding to operators different from input and output prefixes, this is done via snf; for example, for a restriction (y), we define $f_{(y)} : PO \longrightarrow PO$ as:
$$f_{(y)}([N]) = [snf((y)N)].$$
- To define function inp, consider the poset of functions
$$FIN(\mathcal{N} \longrightarrow PO) = \{f : \mathcal{N} \longrightarrow PO|\ \text{only for finitely many } x,\ f(x) \neq [\Omega]\}$$
with the partial ordering pointwise inherited from PO. Function $inp : \mathcal{N} \times FIN(\mathcal{N} \longrightarrow PO) \longrightarrow PO$ is then defined as follows:
$$inp(x, f) = [x(y).(\sum_{z \in F}[y = z]N_z + [y \notin F]\Omega)]$$
where $f(z) = [N_z]$, $F = \{x|\ f(x) \neq [\Omega]\}$ and y is fresh.
- Function $out : \mathcal{N} \times \mathcal{N} \times PO \longrightarrow PO$ is: $out(x, y, [N]) = [\overline{x}y.N]$.

It is easily checked that the above functions are well-defined and monotonic in PO (or $FIN(\mathcal{N} \longrightarrow PO)$ in the case of $inp(x, -)$).

3. *PO and $FIN(\mathcal{N} \longrightarrow PO)$ are extended to algebraic cpo's.*
Ideal completion [Hen88] is used to obtain the algebraic cpo's PO^∞ and $(FIN(\mathcal{N} \longrightarrow PO))^\infty$ and the unique continuous extensions of the above defined monotonic functions; for the sake of simplicity, we will use for the countinous extensions the same names as the monotonic functions they extend. The crucial point here is that, as easily seen, the algebraic cpo $(FIN(\mathcal{N} \longrightarrow PO))^\infty$ is isomorphic to $(\mathcal{N} \longrightarrow PO^\infty)$, thus they can be identified. The wanted domain is then $\mathcal{D} \stackrel{def}{=} PO^\infty$; this together with the extended functions give a natural interpretation of $\mathcal{P}_1$.

4. *Full abstraction of our interpretation with respect to $\sqsubseteq^{+}_{\sim must}$ is proven.*
The proof relies on the existence, for each P, of a set of syntactical *Finite-Breadth Approximants FBA(P)* (see Section 3.5), such that (we denote with sup the least upper bound in $\mathcal{D}$):

a. if for each $N \in FBA(P)$ $N \sqsubseteq_{must} Q$ then $P \sqsubseteq_{must} Q$;

b. $[\![FBA(P)]\!] = \{[\![N]\!] \mid N \in FBA(P)\}$ is directed in $\mathcal{D}$ and
$[\![P]\!] = sup\,[\![FBA(P)]\!]$.

It is easy to show that for each $N \in FSNF$, $[\![N]\!] = [N]$. By exploiting this fact, a. and b. above and the algebraicity of $\mathcal{D}$, it is not hard to show that for each P and Q: $[\![P]\!] \preceq [\![Q]\!]$ iff $P \sqsubseteq_{must} Q$, and, hence, iff $P \underset{\sim}{\sqsubset}^{+}_{must} Q$ (see detailed proofs in Section 6).

3.4 Dealing with Infinite Agents

In this section we sketch the extensions of the equational and denotational characterizations described in previous sections to the full calculus with recursive definitions. We will study the must preorder only; the may case is trivial and the treatment of the testing preorder can be obtained by "composing" may and must. Most of what we are going to do is in fact standard algebraic semantics in the style of [Hen88] and [DH84]. Below, we give all new definitions needed for extending the testing framework:

- The syntax is extended by adding the following two clauses to the grammar of Section 3.1:

$$P ::= X \quad | \quad recX.P$$

  where X ranges over a countable set AV of agents variables. The restrictions on this syntax and the operational semantics of rec are those discussed in Chapter 2, Section 2.1.

- Extending the processes language requires, in the testing framework, extending observers accordingly.

- The definition of predicate $\downarrow$ is extended with the clause:

$$P[recX.P/X]\downarrow \text{ implies } recX.P.\downarrow$$

  This amounts to saying that now $P\downarrow$ if and only if in P there are no unguarded occurrences of Ω and agent variables.

- The definition of the convergence predicate ($\Downarrow$) is restated as follows:

$$P \Downarrow \text{ iff } (P \Longrightarrow P' \text{ implies } P'\downarrow) \text{ and } (\text{not } P \xrightarrow{\tau^{\omega}}).$$

- An interactions is now either finite, as described in Definition 3.1.3, or an infinite sequence of τ-transitions.

- The definition of the must relation is extended to take into account the new infinite interactions. Thus P **must** o if and only if for all interactions (finite or infinite):

$$P|o = e_0 \xrightarrow{\tau} e_1 \xrightarrow{\tau} e_2 \xrightarrow{\tau} \cdots$$

  $\exists j \geq 0$, such that: $o_j \xrightarrow{\omega}$ and $\forall i \geq 0$, $((e_i \uparrow \text{ implies } i \geq j)$.

Notice that now we have now another type of unsuccessful computations, namely the infinite sequence of τ-transitions that never reaches a successful state:

$$P|o = e_0 \xrightarrow{\tau} e_1 \xrightarrow{\tau} e_2 \xrightarrow{\tau} \cdots \text{ and } e_i \overset{\omega}{\not\longrightarrow} \text{ for } i \geq 0\,.$$

In this new setting, we will first show how to extend the proof system presented at the beginning of Section 3.3 for the finite fragment to get a sound and complete (though infinitary) proof system for the full language. Then we will show that the denotational model of Section 3.3 is fully abstract for the full language as well.

All soundness proofs for the finite calculus easily extend to the infinite case (see Section 3.5). In order to get a complete proof system for our processes, we have to consider a proof system for *open* agents (i.e. terms that may contain agents variables not bound by a rec operator). First, the $\sqsubseteq^{+}_{must}$ relation is extended to open terms by letting: $P \sqsubseteq^{+}_{must} Q$ iff $P\rho \sqsubseteq^{+}_{must} Q\rho$ for each substitution of agent variables with closed processes ρ. Then, we add to the proof system for finite agents introduced at the beginning of Section 3.3, the rules to deal with recursion listed below, and, in the resulting proof system, allow $P, Q, \ldots$ to range over open agents and D over rec-free open agents. The resulting system will be referred to as $\mathcal{A}$.

$$REC1 \quad P[recX.P/X] = recX.P$$

$$REC2 \quad \frac{P \sqsubseteq Q}{recX.P \sqsubseteq recX.Q}$$

$$\omega - induction \quad \frac{\forall D \in FIN(P)\ D \sqsubseteq Q}{P \sqsubseteq Q}$$

$$Inst \quad \frac{P \sqsubseteq Q}{P\rho \sqsubseteq Q\rho} \quad \text{for every instantiation } \rho \text{ of agent variables with closed processes}$$

where $FIN(P)$ are the finite syntactic approximants of P; we skip its actual definition, since it is a standard construction of algebraic semantics and can be found in, e.g., [Hen88]. We wish only to remind the reader that set $FIN(P)$ is exclusively concerned with the approximation *in depth* of agent P: an element of $FIN(P)$ is in fact obtained from P by performing a finite number of unfolding of its recursive definitions (subterms of the form $recX.P'$) and then replacing them with Ω. On the contrary, finite strong normal forms (FBA) provide a *breadth* approximation for finite agents. These two kinds of approximation can be dealt with separately. We have already considered the breadth one when considering finite agents; now, we indicate how to compose that treatment with the depth one.

Note that the rule $Inst$ is sound by definition and that, thanks to this fact, soundness of $\mathcal{A}$ for open terms is an easy consequence of soundness for the closed ones. As to completeness, we are only interested in closed terms. Therefore, from now on, we shall be concerned with closed processes only (terms of our language). Soundness of $\mathcal{A}$ can be proved, again, along the same lines of [DH84]. Law $REC2$ is derivable from the other axioms in $\mathcal{A}$ (see [DH84]) and to prove soundness of $REC1$ it is sufficient to define $Der_\alpha(P) = \{P' \mid P \xrightarrow{\alpha} P'\}$, and to exploit the following fact:

if for each α $Der_\alpha(P) = Der_\alpha(Q)$ and ($P \uparrow$ iff $Q \uparrow$) then $P \simeq Q$.

The only non-trivial task is proving soundness of $\omega - induction$. Before proving it, it is useful to show *partial completeness* of $\mathcal{A} - \{\omega - induction, REC2\}$, i.e: for each finite D and for each Q:

$D \precsim^{+}_{must} Q$ implies $\mathcal{A} - \{\omega - induction, REC2\} \vdash D \sqsubseteq_{must} Q$.

To prove this, minor modifications of the completeness proof for finite closed agents of Section 6 are needed. We list them below.

Provably (within $\mathcal{A} - \{\omega - induction, REC2\}$) equivalent head normal forms for convergent agents still exist (see Proposition 3.5.14); however, this time the proof of the parallel composition case is done by induction on $mc(P)$, the maximal number of internal communications P can perform; this number is finite by a Koenigs lemma argument that relies on the fact that $P \Downarrow$. On the other side, since we are concerned with the must preorder only, divergent agents all collapse to Ω (due to $P + \Omega =_M \Omega$). These considerations allow us to extend the proof of Theorem 3.5.22 to prove partial completeness.

To prove soundness of $\omega - induction$ rule, it is also useful to have an alternative characterization of the must preorder which is based on a smaller set of observers.

Lemma 3.4.1 *The must testing preorder $<_M$ obtained by considering the set of observers without restriction coincides with* $\precsim_{must}$.

PROOF: See Section 3.5. □

Now, soundness of $\omega - induction$ rule easily follows from the definition of $FIN(P)$ and from the following crucial proposition, which relies on the soundness of $\mathcal{A} - \{\omega - induction, REC2\}$.

Proposition 3.4.2 *For any process P and any restriction-free observer o, if P* **must** *o then there exists $D \in FIN(P)$ such that D* **must** *o.*

PROOF: See Section 3.5. □

We now come to show completeness of $\mathcal{A}$. Suppose that for closed P and Q, $P \precsim^{+}_{must} Q$; by partial completeness, for each $D \in FIN(P)$, $\mathcal{A} \vdash D \sqsubseteq_{must} Q$; now, apply $\omega - induction$ rule. Thus we can state:

Theorem 3.4.3 (Completeness of $\mathcal{A}$) $P \precsim^{+}_{must} Q$ *implies* $\mathcal{A} \vdash P \sqsubseteq_{must} Q$.

We are left with showing how to build a fully abstract denotational model for the must preorder. Indeed the definition of the model remains unchanged, but that of the semantic function is extended to deal with agent variables and with the recursion operator. Let $ENV = (AV \longrightarrow \mathcal{D})$ be the set of *environments* ranged over by the metavariable ρ and let $\mathcal{RP}$ be the set of all terms generated by our grammar (either closed or not). The interpretation function

$$[\![\,.\,]\!] : \mathcal{RP} \longrightarrow (ENV \longrightarrow \mathcal{D})$$

is defined as as follows:

- $[\![X]\!]\rho = \rho(X)$
- $[\![op(P_1, ..., P_k)]\!]\rho = f_{op}([\![P_1]\!]\rho, ..., [\![P_k]\!]\rho)$, for each k-ary operator different from input and output, $k = 0, 1$ or 2
- $[\![x(y).P]\!]\rho = inp(x, \lambda z \in \mathcal{N}.[\![P\{z/y\}]\!]\rho)$
- $[\![\overline{x}y.P]\!]\rho = out(x, y, [\![P]\!]\rho)$
- $[\![recX.P]\!]\rho = fix(\lambda d \in D.[\![P]\!]\rho[d/X])$.

When P is a closed process, $[\![P]\!]\rho$ does not depend on the actual ρ; in this case we simply write $[\![P]\!]\rho$ as $[\![P]\!]$. It can be proved [HP80] that the above definition makes sense and that many of the usual results of traditional algebraic semantics are still valid. The following fact will be essential:

$$\{[\![D]\!]\rho \,|\, D \in FIN(P)\} \text{ is directed in } \mathcal{D} \text{ and } [\![P]\!]\rho = sup\{[\![D]\!]\rho \,|\, D \in FIN(P)\}$$

It permits concluding that the denotations of infinite elements are uniquely determined by those of the finite ones. We state:

Theorem 3.4.4 (Full abstraction for the full calculus) *$[\![P]\!] \preceq [\![Q]\!]$ in $\mathcal{D}$ if and only if $P \sqsubseteq^{+}_{must} Q$.*

PROOF: See Section 3.5. □

3.5 Detailed Proofs

Additional notations

In addition to that fixed in Chapter 2, we need some more notation for our proofs:

- $Obj(P) = \{x \in fn(P) \,|\, x = obj(\alpha)$ for some action α in $P\}$.
- Given a hnf H with g as associated function, and $\overline{x} \in Ev(H)$, we write the generic $g(\overline{x})$ simply as a sum $\sum_{z \in N_x} \overline{x}z.H_{\overline{x}z}$, with the convention that $g(\overline{x})$ has a summand of the kind $\overline{x}(w).H_{\overline{x}(w)}$ (i.e. a *bound output summand*) if and only if a special item (w) belongs to N_x.

3.5.1 Basic properties

In this subsection we fix some terminology that will be extensively used in later proofs and establish some basic facts about the testing preorders.

We introduce below the technical notion of equivalent experiments, that will be extensively used in later proofs.

Definition 3.5.1 (Equivalent sequences of τ-actions) *Given any two sequences of τ-transitions s_1, s_2 of the same length ($k \geq 0$):*

$$s_1 = (e_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} e_k) \text{ and } s_2 = (f_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} f_k)$$

we say that s_1 and s_2 are equivalent *if and only if for all i, $0 \leq i \leq k$, $\nu \in \{\omega, \tau\}$ ($e_i \xrightarrow{\nu}$ if and only if $f_i \xrightarrow{\nu}$) and ($e_i \uparrow$ if and only if $f_i \uparrow$).*

Definition 3.5.2 (Equivalent experiments)

- *Given two experiments e_1, e_2, we set $e_1 \leq e_2$ if and only if for each invisible sequence s_1 from e_1 there exists an equivalent invisible sequence s_2 from e_2; $\leq$ is clearly a preorder.*
- *We say that e_1 is* equivalent *to e_2, if and only if $e_1 \leq e_2$ and $e_2 \leq e_1$.*

Since computations are particular sequences of τ-actions, we can use the notion of equivalent experiments to establish testing preorder relations between processes. Indeed, if $e_1 \stackrel{def}{=} [P|o]$ and $e_2 \stackrel{def}{=} [Q|o]$ are two experiments, we have that:

- $e_1 \leq e_2$ implies (P **may** o implies Q **may** o), and
- $e_1 \leq e_2$ implies (P $\cancel{\textbf{m}}$**ust** o implies Q $\cancel{\textbf{m}}$**ust** o).

Thus, a possible technique for proving that $P \sqsubseteq_{may} Q$, would be proving that $P|o \leq Q|o$ for each observer o. Analogously $P \sqsubseteq_{must} Q$ can be proved by showing that $Q|o \leq P|o$ for each observer o. It follows that if $P|o$ is equivalent to $Q|o$ for each o, we have $P \simeq Q$ (i.e. P and Q are testing equivalent). A useful result about sequences of invisible actions is reported below.

Lemma 3.5.3 *If $e_2 = e_1\{w'/w\}$, with $w = w'$ or $w' \notin fn(e_1)$, then $e_1 \leq e_2$.*

PROOF: An easy induction on the length of the sequences of invisible actions performable by the two experiments. □

Next lemma states some basic properties of our preorders, that will be used in later proofs.

Lemma 3.5.4 *Given $P, Q \in \mathcal{P}_1$ and $\alpha \in \Lambda$ with $bn(\alpha) \cap fn(P,Q) = \emptyset$:*

1. *($P \sqsubseteq_{may} Q$ and $P \stackrel{\alpha}{\Longrightarrow}$) imply $Q \stackrel{\alpha}{\Longrightarrow}$;*
2. *($P \sqsubseteq^{+}_{must} Q$ and $P \Downarrow$ and $Q \stackrel{\alpha}{\Longrightarrow}$) imply ($Q \Downarrow$ and $P \stackrel{\alpha}{\Longrightarrow}$);*
3. *($P \sqsubseteq^{+}_{must} Q$ and $P \Downarrow$ and $Q \stackrel{\tau}{\longrightarrow}$) imply $P \stackrel{\tau}{\longrightarrow}$.*

PROOF:

1. We first show that P may satisfy a certain observer o, whose form depends on the kind of the performed action α, and which is defined as follows:

- if $\alpha = \overline{x}y$ then $o \stackrel{def}{=} x(z).[z = y]\omega.\mathbf{0}$, $z \neq y$;
- if $\alpha = x(y)$ then $o \stackrel{def}{=} \overline{x}w.\omega.\mathbf{0}$, with w any name;
- if $\alpha = \overline{x}(y)$ then $o \stackrel{def}{=} x(z).[z \notin Obj(P,Q)]\omega.\mathbf{0}$ (recall that $Obj(P,Q) = \{y \in fn(P,Q) \mid y$ is in object position in P or in $Q\}$) with $z \notin Obj(P,Q)$.

A simple case analysis on α shows that $P \stackrel{\alpha}{\Longrightarrow}$ implies P **may** o, which in turn implies Q **may** o. But the latter, as easily seen (by another simple case analysis on α), implies that $Q \stackrel{\alpha}{\Longrightarrow}$. Notice that in case $\alpha = \overline{x}(y)$, we rely on the ability of P to send a fresh name to o.

2. First we show that $P \Downarrow$ implies $Q \Downarrow$: $P \Downarrow$ implies P **must** $\tau.\omega.\mathbf{0}$ implies Q **must** $\tau.\omega.\mathbf{0}$ implies $Q \Downarrow$. Next, suppose by contradiction that not $P \stackrel{\alpha}{\Longrightarrow}$. Then we can exhibit an observer o, depending on the kind of the performed α, such that P **must** o and Q $\not$**must** o, that contradicts $P \sqsubseteq_{must} Q$. The observer o is defined as follows:

- $\alpha = \overline{x}y$ implies $o \stackrel{def}{=} x(z).[z \neq y].\omega.\mathbf{0} + \tau.\omega.\mathbf{0}$, $z \neq y$;
- $\alpha = x(y)$ implies $o \stackrel{def}{=} \overline{x}w.\mathbf{0} + \tau.\omega.\mathbf{0}$;
- $\alpha = \overline{x}(y)$ implies $o \stackrel{def}{=} x(z).([z = y_1]\omega.\mathbf{0} + ... + [z = y_k]\omega.\mathbf{0}) + \tau.\omega.\mathbf{0}$, with $Obj(P,Q) = \{y_1, ..., y_k\}$, $k \geq 0$ (if $Obj(P,Q) = \emptyset$ then the above summation reduces to $\mathbf{0}$) and with $z \notin Obj(P,Q)$.

Then, since $P \Downarrow$ and $P \stackrel{\alpha}{\not\Longrightarrow}$, by a simple case analysis on α, we can show that P **must** o, but Q $\not$**must** o, which is impossible. Thus it must be $P \stackrel{\alpha}{\Longrightarrow}$. Notice that, this time in the proof of case $\alpha = \overline{x}(y)$, we rely on the ability of Q to send a fresh (hence not belonging to $Obj(P,Q)$) name to o and on the inability of P to do so.

3. Suppose $Q \stackrel{\tau}{\longrightarrow} Q'$ and $P \stackrel{\tau}{\not\longrightarrow}$. Take $x \notin fn(P,Q)$ and $o \stackrel{def}{=} x(y).\omega.\mathbf{0}$. Then $P + \overline{x}y.\mathbf{0}$ **must** o: in fact $P \Downarrow$ and since $P \stackrel{\alpha}{\not\longrightarrow}$ with α **comp** $x(y)$ (in virtue of Lemma 2.1.1) and $P \stackrel{\tau}{\not\longrightarrow}$, the only possible computation is $(P + \overline{x}y.\mathbf{0})|o \stackrel{c}{\longrightarrow} \mathbf{0}|\omega.\mathbf{0}$, that is

successful. But $Q + \overline{x}y.\mathbf{0}$ **must** o, due to the unsuccessful computation starting as $(Q + \overline{x}y.\mathbf{0})|o \xrightarrow{\tau} Q'|o$, (where we have exploited the fact that, by Lemma 2.1.1, for no α such that α **comp** $\overline{x}y$, $Q' \stackrel{\alpha}{\Longrightarrow}$, and thus $Q'|o \stackrel{c}{\not\Longrightarrow}$). But this contradicts the fact that $P \sqsubseteq^{+}_{must} Q$. □

We now prove a proposition about a variant of the testing theory not using the mismatch operator. The significance of this result is discussed in Subsection 3.1.3.

Proposition 3.5.5 (Proposition 2.1) *Let be x and y be any two names. Then:*

a. $\overline{x}(w).\mathbf{0} \sqsubseteq^{-}_{may} \overline{x}.y.\mathbf{0}$;

b. $\overline{x}(w).\Omega \sqsubseteq^{-}_{must} \overline{x}(w).\mathbf{0} + \overline{x}y.\Omega$.

PROOF: To prove this proposition we need a stronger version of Lemma 2.1.2. More precisely, we need to discard the hypothesis on the injectivity of σ. This is possible, when mismatch is not considered (see Remark 2.1.3). Formally, we have:

a. Given a mismatch-free observer o, suppose P **may** o. We can assume, w.l.o.g., that $w \notin fn(o)$; then it must be, for some o', that $P|o \Longrightarrow P|o' \xrightarrow{c} (w)(\mathbf{0}|o_1)$, where $o_1 \stackrel{\omega}{\Longrightarrow}$ and $o \xrightarrow{x(w)} o_1$. On the other side, we have $Q|o \Longrightarrow Q|o' \xrightarrow{c} \mathbf{0}|(o_1\{y/w\})$. Then, by (repeatedly) applying Lemma 2.1.4 with the hypothesis σ *injective* discarded[3] it is easy to see that also $o_1\{y/w\} \stackrel{\omega}{\Longrightarrow}$ and thus also Q **may** o, and the thesis follows.

b. Let $P \stackrel{def}{=} \overline{x}(w).\Omega$ and $Q \stackrel{def}{=} \overline{x}(w).\mathbf{0}+\overline{x}y.\Omega$. Suppose, for any mismatch-free observer o, that Q **must** o. We can assume w.l.o.g. that $w \neq y$ and that $w \notin fn(o)$. We shall consider the possible cases of unsuccessful computation for $Q \mid o$ and in each case exhibit a unsuccesful computation for $P|o$ as well.

The non-trivial cases are those in which a communication between Q and the observer occurs before the observer itself reaches a divergent state, i.e. those computations having an initial segment of the kind:

$$Q|o \Longrightarrow Q \mid o_k \xrightarrow{c} [Q' \mid o_{k+1}]$$

where $o \xrightarrow{\tau} o_1 \xrightarrow{\tau} ... \xrightarrow{\tau} o_k$ with $o_i \stackrel{\omega}{\not\longrightarrow}$ for $0 \leq i \leq k+1$ and Q' $\cancel{\mathbf{must}}$ o_{k+1}. Let us analyze the possible cases for the communication $Q \mid o_k \xrightarrow{c} [Q' \mid o_{k+1}]$. There are two possibilities:

1. Q synchronizes via the bound output $\overline{x}(w)$. In this case, also $P|o$ has the failing computation $P|o \Longrightarrow P|o_k \xrightarrow{c} (w)(\Omega|o_{k+1})$.

[3] This is essential, because we do not actually know whether $y \notin fn(o_1)$, i.e. whether σ is injective on o_1.

2. Q synchronizes via the free output $\overline{x}y$. It is therefore $o_k \xrightarrow{x(z)} o'$ for some fresh z and o' such that $o_{k+1} = o'\{y/z\}$. Therefore, $P|o$ has the failing computation with the initial segment:
$$P|o \Longrightarrow P|o_k \xrightarrow{c} (w)(\Omega|o'\{w/z\}) .$$
To prove that this yields a failing computation, it suffices to show that $o'\{w/z\} \not\xrightarrow{\omega}$. Suppose the contrary, i.e. $o'\{w/z\} \xrightarrow{\omega}$. Since $w \notin fn(o_{k+1})$ (this is proven by repeatdly applying Lemma 2.1.1), it is $o_{k+1} = o'\{y/z\} = (o'\{w/z\})\{y/w\}$. In virtue of Lemma 2.1.4 with the hypothesis σ injective discarded, it would hence be also $o_{k+1} \xrightarrow{\omega}$; but this contradicts Q' $\cancel{\mathbf{must}}$ o_{k+1}.

$\square$

3.5.2 Soundness proofs

We need some preliminary lemmata. The following two will be used to show that the restriction operator preserves the testing preorder.

Lemma 3.5.6 *For all P, o and $y \notin fn(o)$: $(y)P|o$ and $P|o$ are equivalent as experiments.*

PROOF: Recall from Definition 3.5.2 that we have to show that for each P, o and $y \notin fn(o)$, (i) $P|o \leq (y)P|o$, and (ii) $(y)P|o \leq P|o$. Define $e \leq_k f$ iff for each sequence (of τ-actions) of length k from e there exists an equivalent sequence from f; clearly, $e \leq f$ iff for each $k \geq 0$ $e \leq_k f$. We now come to the actual proof of (i) and (ii).

(i) We show by induction on k that for each sequence $P|o \Longrightarrow [P_k|o_k]$ of length k there exists an equivalent sequence from $(y)P|o$, i.e $P|o \leq_k (y)P|o$. Suppose $k > 0$ and $P|o \xrightarrow{\tau} [P_1|o_1] \Longrightarrow [P_k|o_k]$. It is easily seen that $(P|o)\downarrow$ iff $((y)P|o)\downarrow$ and $P|o \xrightarrow{\omega}$ iff $(y)P|o \xrightarrow{\omega}$. Thus it suffices to show, by exploiting the induction hypothesis, that $(y)P|o \xrightarrow{\tau} e$, where e is an experiment such that $[P_1|o_1] \leq_{k-1} e$. This will be done by a case analysis on how transition $P|o \xrightarrow{\tau} [P_1|o_1]$, which we call TR, is inferred.

1. The cases of the form:
$$\texttt{Par} : \frac{P \xrightarrow{\tau} P'}{TR}, \quad \texttt{Par} : \frac{o \xrightarrow{\tau} o'}{TR}, \quad \texttt{R} : \frac{P \xrightarrow{\alpha} P' \quad o \xrightarrow{\beta} o'}{TR}$$
with α **comp** β and $\texttt{R} \in \{\texttt{Com, Close}\}$ and $y \notin n(\alpha, \beta)$, are obvious. In fact it is easily seen that a transition $(y)P|o \xrightarrow{\tau} [(y)P_1|o_1]$ always exists such that $y \notin fn(o_1)$, thus the thesis follows from the inductive hypothesis [4].

[4] Given two equivalent experiments, prefixing by restrictions one or both of them does not affect their equivalence as experiments

2.

$$\texttt{Com} \;:\; \frac{P \xrightarrow{\overline{x}y} P' \quad o \xrightarrow{x(w)} o'}{P|o \xrightarrow{\tau} P'|o'\{y/w\} \stackrel{def}{=} P_1|o_1}\,.$$

We can rename w into a fresh name $w' \notin n(y, P', o')$, thus:

$$\texttt{Lemma 2.1.4} : \; \frac{o \xrightarrow{x(w)} o'}{o \xrightarrow{x(w')} o'' = o'\{w'/w\}}\,.$$

Now, consider the following derivation:

$$\texttt{Close}: \frac{\texttt{Open} \;:\; \dfrac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(w')} P'\{w'/y\}} \qquad o \xrightarrow{x(w')} o''}{(y)P|o \xrightarrow{\tau} (w')(P'\{w'/y\}|o'') \stackrel{def}{=} e}$$

Since $y \notin fn(o)$, from Lemma 2.1.1 it follows $w = y$ or $y \notin fn(o')$, hence we have:

$$(P_1|o_1)\{w'/y\} = P'\{w'/y\}|o'\{y/w\}\{w'/y\} = P'\{w'/y\}|o'' = e'$$

and from Lemma 3.5.3 we get $P_1|o_1 \leq e'$. But $e = (w')e'$ is trivially equivalent as experiment to e' and thus $P_1|o_1 \leq e$, and the thesis follows.

3.

$$\texttt{Com} \;:\; \frac{P \xrightarrow{x(y)} P' \quad o \xrightarrow{\overline{x}w} o_1}{TR}\,.$$

Rename y into a fresh y' via Lemma 2.1.4, as done in the above case 1 with w.

4.

$$\texttt{Close} \;:\; \frac{P \xrightarrow{\overline{x}(y)} P' \quad o \xrightarrow{x(y)} o'}{TR}\,.$$

Rename y into a fresh y' via Lemma 2.1.4, as done in case 1 with w.

5. All remaining cases are such that:

$$\texttt{R} \;:\; \frac{P \xrightarrow{\alpha} P' \quad o \xrightarrow{\beta} o'}{P|o \xrightarrow{\tau} R_{\alpha\beta} \stackrel{def}{=} [P_1|o_1] = TR}\,\alpha \;\mathbf{comp}\; \beta$$

where `R = Com` or `R=Close` and $y \in n(\beta)$. It must be then $y = bn(\beta)$: it cannot be $subj(\beta) = y$ due to the hypothesis $y \notin fn(o)$. The subcase $\beta = x(y)$ and $\alpha = \overline{x}w$,

$w \neq y$, is trivial. Then the only remaining subcase is when $\beta = \overline{x}(y)$ and, hence, $\alpha = x(y)$. Therefore it is `R = Close` and $R_{\alpha\beta} = (y)(P'|o')$. Taken a fresh name z we have the following derivations:

$$\texttt{Res} \; : \; \frac{\text{Lemma 2.1.4} \;\; : \;\; \dfrac{P \xrightarrow{\alpha} P'}{P \xrightarrow{x(z)} P'' = P'\{z/y\}}}{(y)P \xrightarrow{x(z)} (y)P''}$$

and

$$\text{Lemma 2.1.4} \;\; : \;\; \frac{o \xrightarrow{\beta} o'}{o \xrightarrow{\overline{x}(z)} o'' = o'\{z/y\}}$$

By applying `Close` to the resulting transitions, we get

$$(y)P|o \xrightarrow{\tau} (z)((y)P''|o'') = e\,.$$

Now, we have $(z)(P''|o'') = (y)(P'|o') = [P_1|o_1]$ and $P''|o''$ is equivalent to $(z)(P''|o'')$ as experiment. Therefore, $P''|o''$ is equivalent to $[P_1|o_1]$. By applying the inductive hypothesis, we have $P''|o'' \leq_{k-1} (y)P''|o'' = e'$, since $y \notin fn(o'')$. Thus, it follows that also $[P_1|o_1] \leq_{k-1} e'$; but $e = (z)e'$ is trivially equivalent to e', thus $[P_1|o_1] \leq_{k-1} e$, and the thesis follows.

(ii) This half is done in the same style of part (i). The only remarkable difference is, in the inductive step, when the `Close` transition of $(y)P|o$ given by:

$$\texttt{Close} : \; \frac{\texttt{Open} \;\; : \;\; \dfrac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(w)} P'\{w/y\}} \qquad o \xrightarrow{x(w)} o'}{(y)P|o \xrightarrow{\tau} (w)(P'\{w/y\}|o') \stackrel{def}{=} [P_1|o_1]}$$

with $w \notin fn((y)P')$, is emulated by $P|o$ by the `Com` transition given by:

$$\texttt{Com} \;\; : \;\; \frac{P \xrightarrow{\overline{x}y} P' \;\; o \xrightarrow{x(w)} o'}{P|o \xrightarrow{\tau} P'|o'\{y/w\} = e}\,.$$

Now, since $w \notin fn((y)P')$, it is $e = P'|o'\{y/w\} = (P_1|o_1)\{y/w\}$ and thus, from Lemma 3.5.3, $P_1|o_1 \leq e$. But, being $P_1|o_1$ trivially equivalent to $[P_1|o_1]$, it follows that $[P_1|o_1] \leq e$, and thus the claim follows. $\square$

Of course, a statement symmetrical to that of the previous lemma holds for $P|(y)o$ and $P|o$, if $y \notin fn(P)$. Next, we have the following lemma:

Lemma 3.5.7 *Let w, y be two distinct names; $P \mathrel{\underset{\sim}{\sqsubset}} Q$ and $w \notin fn(P, Q), y \notin fn(o)$ imply:*

i. $P\{w/y\}$ **may** o *implies* $Q\{w/y\}$ **may** o*;*

ii. $P\{w/y\}$ **must** o *implies* $Q\{w/y\}$ **must** o.

Proof:

i. We have that $(P\{w/y\}|o)\{y/w\} = (P|o\{y/w\})$, with $y \notin fn(P\{w/y\}|o)$. Then by Lemma 3.5.3 we have $P\{w/y\}|o \leq (P|o\{y/w\})$ and thus P **may** $o\{y/w\}$. This implies Q **may** $o\{y/w\}$. But $(Q|o\{y/w\})\{w/y\} = (Q\{w/y\}|o)$, with $w \notin fn(Q|o\{y/w\})$. Therefore from Lemma 3.5.3 we have $Q|o\{w/y\} \leq (Q\{w/y\}|o)$ and therefore the thesis follows.

ii. Suppose $Q\{w/y\}$ $\not\mathbf{m}\mathbf{ust}$ o. Then by a reasoning dual to i. (substituting ' **may** ' with ' $\not\mathbf{m}\mathbf{ust}$ ' and interchanging 'P' and 'Q' the above chain of implications still holds) we can show that it is $P\{w/y\}$ $\not\mathbf{m}\mathbf{ust}$ o. □

A relevant consequence of the above lemma is the following proposition, which asserts that our preorder is preserved by substitutions under certain assumptions:

Proposition 3.5.8 *$P \mathrel{\underset{\sim}{\sqsubset}} Q$ and $z \notin fn(P, Q)$ imply $P\{z/y\} \mathrel{\underset{\sim}{\sqsubset}} Q\{z/y\}$.*

Proof: Suppose $P\{z/y\}$ **may** o. If $y \notin fn(o)$, we apply the previous lemma, otherwise take y' fresh and suppose $z \neq y$. Consider then the following implications:

$P\{z/y\}$ **may** o	implies $P\{z/y\}$ **may** $o\{y'/y\}$	(Lemma 3.5.3 and $P\{z/y\}\|o\{y'/y\} = (P\{z/y\}\|o)\{y'/y\})$
	implies $Q\{z/y\}$ **may** $o\{y'/y\}$	(Lemma 3.5.7)
	implies $Q\{z/y\}$ **may** $o\{y'/y\}\{y/y'\}$	(Lemma 3.5.3)
	implies $Q\{z/y\}$ **may** o	(Lemma 3.5.3).

By a dual reasoning (interchanging 'P' and 'Q' and substituting ' **may** ' with ' $\not\mathbf{m}\mathbf{ust}$ ' in the above chain of implications) we can show that $Q\{z/y\}$ $\not\mathbf{m}\mathbf{ust}$ o implies $P\{z/y\}$ $\not\mathbf{m}\mathbf{ust}$ o. □

The following lemma will be used to prove that our preorder is preserved by the parallel composition operator:

Lemma 3.5.9 *Let $P, R \in \mathcal{P}_1$, $o \in O$. Then:*

i. P **may** $R|o$ *iff* $P|R$ **may** o*;*

ii. P **must** $R|o$ *iff* $P|R$ **must** o.

PROOF: It suffices to show that $P|(R|o)$ and $(P|R)|o$ are equivalent as experiments. The structure of the proof is very similar to that of Lemma 3.5.6. We proof only that $(P|R)|o \leq P|(R|o)$ half, the other direction is perfectly symmetrical.

Suppose $(P|R)|o \xrightarrow{\tau} [[P_1|R_1]|o_1] \Longrightarrow [[P_k|R_k]|o_k]$, $k > 0$. It is obvious that $(P|R)|o \uparrow$ iff $P|(R|o) \uparrow$ and that $(P|R)|o \xrightarrow{\omega}$ iff $P|(R|o) \xrightarrow{\omega}$. Thus it suffices to show that $P|(R|o) \xrightarrow{\tau} e$, where e is an experiment such that $[[P_1|R_1]|o_1] \leq_{k-1} e$ (recall the definition of $\leq_k$ from the proof of Lemma 3.5.6). This will be done with a case analysis on the proof of transition $(P|R)|o \xrightarrow{\tau} [[P_1|r_1]|o_1]$; later we will refer to this transition by calling it TR.

1. The cases of the form

$$\texttt{Par}\ :\ \cfrac{\texttt{Par}\ :\ \cfrac{P \xrightarrow{\tau}}{P|R \xrightarrow{\tau}}}{TR},\quad \texttt{Par}\ :\ \cfrac{\texttt{Par}\ :\ \cfrac{R \xrightarrow{\tau}}{P|R \xrightarrow{\tau}}}{TR}\ \text{and}\ \texttt{Par}\ :\ \cfrac{o \xrightarrow{\tau}}{TR}$$

   are obvious, applying the inductive hypothesis.

2.

$$\texttt{Par}\ :\ \cfrac{\texttt{Close}\ :\ \cfrac{P \xrightarrow{\alpha} P' \quad R \xrightarrow{\beta} R'}{P|R \xrightarrow{\tau} (w)(P'|R')}}{(P|R)|o \xrightarrow{\tau} (w)(P'|R')|o \stackrel{def}{=} [[P_1|r_1]|o_1]}$$

   where α **comp** β and $bn(\alpha) = bn(\beta) = w$. We can suppose $w \notin fn(o)$ without loss of generality, since, via Lemma 2.1.4, we can always rename w into a fresh w' in the above derivation, obtaining an experiment equivalent to $[[P_1|R_1]|o_1]$ as a result. Thus we can consider the derivation:

$$\texttt{Close}:\ \cfrac{P \xrightarrow{\alpha} P' \quad \texttt{Par}\ :\ \cfrac{R \xrightarrow{\beta} R'}{R|o \xrightarrow{\beta} R'|o}}{P|(R|o) \xrightarrow{\tau} (w)(P'|(R'|o)) \stackrel{def}{=} e.}$$

   By applying the inductive hypothesis, we get $(P'|R')|o \leq_{k-1} P'|(R'|o) \stackrel{def}{=} e'$; but e' is trivially equivalent to e, and $(P'|R')|o$ is equivalent to $(w)(P'|R')|o = [[P_1|o_1]|R_1]$ in virtue of Lemma 3.5.6, since $w \notin fn(o)$, therefore we have $[[P_1|R_1]|o_1] \leq_{k-1} e$.

   All other cases are similar to, or easier than, case 2. The corresponding proofs are routine, and are omitted.

□

We come to the actual soundness proofs for inference rules and axioms.

Proposition 3.5.10 (Proposition 3.2.1) *The inference rules of Table 3.2 are sound w.r.t.* $\underset{\sim}{\sqsubset}^{+}$.

PROOF:

($C0$). We omit the proofs for the free actions prefixes (τ and free output) and summation cases, which are much the same as for CCS [DH84].

- The match case is trivial if $x \neq y$, otherwise it is proven by showing that for each o, $[x = x]P|o$ and $P|o$ are equivalent as experiments, which is easy. The mismatch case is analogous. Let us see in more detail the next two cases, which are harder.

- Restriction. Suppose $P \underset{\sim}{\sqsubset}^{+} Q$. Given any observer o, take $y' \notin fn(P, Q, o)$ and $P' \stackrel{def}{=} P\{y'/y\}$, $Q' \stackrel{def}{=} Q\{y'/y\}$. Then we have the following chain of implications:

$$\begin{array}{lll} (y)P \ \textbf{may}\ o & \text{implies } (y')P' \ \textbf{may}\ o & (\alpha\text{-equivalence}) \\ & \text{implies } P' \ \textbf{may}\ o & (\text{Lemma 3.5.6}) \\ & \text{implies } Q' \ \textbf{may}\ o & (\text{Proposition 3.5.8}) \\ & \text{implies } (y')Q' \ \textbf{may}\ o & (\text{Lemma 3.5.6}) \\ & \text{implies } (y)Q \ \textbf{may}\ o & (\alpha\text{-equivalence})\,. \end{array}$$

  Thus $(y)P \underset{\sim}{\sqsubset}_{may} (y)Q$. Now, by a dual reasoning (substitute ' **may** ' with ' **must** ' and exchange 'P' and 'Q' in the above implications) we show that, for any o, $(y)Q$ **must** o implies $(y)P$ **must** o, that is $(y)P \underset{\sim}{\sqsubset}_{must} (y)Q$. Moreover, if $(y)P \Downarrow$ then also $P \Downarrow$; thus we have that: $(y)Q \stackrel{\tau}{\longrightarrow}$ implies $Q \stackrel{\tau}{\longrightarrow}$ implies $P \stackrel{\tau}{\longrightarrow}$ (from the definition of $\underset{\sim}{\sqsubset}^{+}_{must}$) implies $(y)P \stackrel{\tau}{\longrightarrow}$. We have showed that $(y)P \underset{\sim}{\sqsubset}^{+}_{must} (y)Q$.

- Parallel Composition. Suppose $P \underset{\sim}{\sqsubset}^{+} Q$. Then, for any observer o, we have the following chain of implications:

$$\begin{array}{lll} P|R \ \textbf{may}\ o & \text{implies } P \ \textbf{may}\ R|o & (\text{Lemma 3.5.9}) \\ & \text{implies } Q \ \textbf{may}\ R|o & (P \underset{\sim}{\sqsubset}_{may} Q) \\ & \text{implies } Q|R \ \textbf{may}\ o & (\text{Lemma 3.5.9}) \end{array}$$

  Thus $P|R \underset{\sim}{\sqsubset}_{may} Q|R$. By a similar reasoning, we show that $P|R \underset{\sim}{\sqsubset}_{must} Q|R$. Moreover, if $(P|R) \Downarrow$ then it must be also $P \Downarrow$ and $R \Downarrow$; therefore, if $Q|R \stackrel{\tau}{\longrightarrow}$, a simple case analysis (on why $Q|R \stackrel{\tau}{\longrightarrow}$) shows that also $P|R \stackrel{\tau}{\longrightarrow}$. Thus $P|R \underset{\sim}{\sqsubset}^{+}_{must} Q|R$.

- Finally, the soundness of the congruence rule for bound output prefix is an easy consequence of the free output prefix and of restriction cases (recall that $\overline{x}(y).P$ stands for $(y)(\overline{x}y.P)$).

$(C1)$. Suppose $y(z).P$ **may** o. Besides the trivial case in which $o \stackrel{\omega}{\Longrightarrow}$, it must be $y(z).P|o \Longrightarrow y(z).P|o_k \stackrel{c}{\longrightarrow} [P_{k+1}|o_{k+1}]$, where P_{k+1} **may** o_{k+1} and $P_{k+1} = P\sigma$, for a suitable $\sigma = \{x/z\}$. Now, if $x \in fn(z, P, Q)$, from the hypothesis we have also $Q\sigma$ **may** o_{k+1}, from which it easily follows that $y(z).Q$ **may** o. Otherwise $x \notin fn(z, P, Q)$, and then from Proposition 3.5.8 we have again that $Q\sigma$ **may** o_{k+1} and hence $y(z).Q$ **may** o.

Suppose $y(z).Q$ **must** o. Besides the trivial case in which $o \Longrightarrow o_k$ with $o_k \uparrow$ and $o_i \not\stackrel{\omega}{\longrightarrow}$ for $0 \le i \le k$, it must hold one of the following:
1. $y(z).Q|o \Longrightarrow y(z).Q|o_k \stackrel{c}{\longrightarrow} [Q\sigma|o_{k+1}]$, where $o_i \not\stackrel{\omega}{\longrightarrow}$ for $0 \le i \le k$ and $Q\sigma$ **must** o_{k+1}, for a suitable $\sigma = \{x/z\}$. Now, if $x \notin fn(z, P, Q)$, from Proposition 3.5.8 we have that it is also $P\sigma$ **must** o_{k+1}, and then it easily follows that $y(z).P$ **must** o; otherwise, this result follows directly from the hypotheses.
2. $y(z).Q|o \Longrightarrow y(z).Q|o_k \not\stackrel{\tau}{\longrightarrow}$, where $o_i \not\stackrel{\omega}{\longrightarrow}$ for $0 \le i < k$. Then it is easily seen that an equivalent computation from $y(z).P|o$ exists, thus $y(z).P$ **must** o. Furthermore, $y(z).Q \not\stackrel{\tau}{\longrightarrow}$, so that it is $y(z).P \underset{\sim}{\sqsubset}^{+}_{must} y(z).Q$. □

Proposition 3.5.11 (Proposition 3.2.2) *The axioms of Table 3.2.1 are sound w.r.t.* $\underset{\sim}{\sqsubset}^{+}$.

PROOF:

- Define for each P and α, $Der_\alpha(P) = \{P' |\, P \stackrel{\alpha}{\longrightarrow} P'\}$. The soundness of the laws in Sum, Match, Mismatch, of laws R2, R4 and of Expansion follows from the following (easy to prove) fact: if for each α, $Der_\alpha(P) = Der_\alpha(Q)$ and ($P \uparrow$ iff $Q \uparrow$) then $P \underset{\sim}{\sqsubset}^{+} Q$ and $Q \underset{\sim}{\sqsubset}^{+} P$.

- $(R0)$. The soundness follows from the following fact: for each o and each P, $x \notin fn(P)$ implies $(x)P|o$ and $P|o$ are equivalent as experiments. If $x \notin fn(o)$, this fact follows from Lemma 3.5.6; otherwise, first rename x into a fresh x'.

- $(R1)$. Consider the following fact: for each o, $(x)(y)P|o$ and $(y)(x)P|o$ are equivalent as experiments. To prove this fact, consider x', y' fresh and $P' \stackrel{def}{=} P\{x'/x\}\{y'/y\}$. Then one has, for each o: $(x)(y)P|o = (x')(y')P'|o$, which in turn is equivalent, by using twice Lemma 3.5.6, to $P'|o$, which is equivalent, (Lemma 3.5.6 twice again, but in the opposite direction), to $(y')(x')P'|o = (y)(x)P|o$.

- $(R3)$. Consider first the case in which α is not a bound output; in order to prove soundness of the law, we first show the following facts:
a) for each $o, x \notin n(\alpha)$: $\alpha.(x)P|o \stackrel{c}{\longrightarrow} [P'|o']$ implies $(x)\alpha.P|o \stackrel{c}{\longrightarrow} [P''|o''] = [P'|o']$;
b) for each $o, x \notin n(\alpha)$: $(x)\alpha.P|o \stackrel{c}{\longrightarrow} [P'|o']$ implies $\alpha.(x)P|o \stackrel{c}{\longrightarrow} [P'|o']$.
a) is proven by a case analysis on why $\alpha.(x)P|o \stackrel{c}{\longrightarrow} [P'|o']$. The only non trivial case is when $\alpha = z(y)$, $z, y \neq x$, and

$$\texttt{Close} \;:\; \frac{\alpha.(x)P \xrightarrow{z(x)} (x')P\{x'/x\}\{x/y\} \quad o \xrightarrow{\overline{z}(x)} o'}{\alpha.(x)P|o \xrightarrow{c} (x)((x')P\{x'/x\}\{x/y\}|o')}, z \neq x$$

(notice that here we have made explicit the α-renaming of x into a fresh x' caused by the substitution $\{x/y\}$). Then, from Lemma 2.1.4, we have that $o \xrightarrow{\overline{z}(x'')} o'' = o'\{x''/x\}$, x'' fresh, so that we have the derivation:

$$\texttt{Close} \;:\; \frac{(x)\alpha.P \xrightarrow{z(x'')} (x)P\{x''/y\} \quad o \xrightarrow{\overline{z}(x'')} o''}{(x)\alpha.P|o \xrightarrow{c} (x'')((x)P\{x''/y\}|o'') = (x)((x')P\{x'/x\}\{x/y\}|o')}$$

b) is trivial to prove. Statements a) and b) together are sufficient to prove that $x \notin n(\alpha)$ implies that $\alpha.(x)P|o$ and $(x)\alpha.P|o$ are equivalent as experiments when α is not a bound output. Finally, if α is a bound output apply first law R1 then law R3.

- $(Omega)$. Trivial.
- $(\tau - laws)$. The proofs for these laws are very similar to those of the corresponding ones for CCS in [DH84]. As an example, we prove N3, confining ourselves to the 'must' part (the 'may' is trivial). We deal with the three cases ($\alpha = \beta = \tau$, $\alpha, \beta \in IA$ and $\alpha, \beta \in OA$) uniformely. In the sequel, 'P **moves-on** o' means '$P|o \stackrel{c}{\Longrightarrow}$ or $P \xrightarrow{\tau}$'. Let $T \stackrel{def}{=} \alpha.P+\tau.(\beta.Q+R)$ and $U \stackrel{def}{=} \tau.(\alpha.P+\beta.Q+R)$. Suppose P **m$\!\!\!/$ust** o. Then it is either $\alpha.P$ **m$\!\!\!/$ust** o or $\tau.(\beta.Q+R)$ **m$\!\!\!/$ust** o. Let us exclude the trivial case in which $o \Longrightarrow o_k$, with $o_k \uparrow$ and $o_i \not\xrightarrow{\omega}$ for $0 \leq i \leq k$. Suppose $\alpha.P$ **m$\!\!\!/$ust** o; then it is easily seen that if $\alpha.P$ **moves-on** o also U **m$\!\!\!/$ust** o, otherwise (i.e. not $\alpha.P$ **moves-on** o) it must be $\beta.Q+R$ **m$\!\!\!/$ust** o and hence also U **m$\!\!\!/$ust** o. In the second case, i.e. $\tau.(\beta.Q+R)$ **m$\!\!\!/$ust** o, it is also $\beta.Q+R$ **m$\!\!\!/$ust** o. Now, if $\beta.Q+R$ **moves-on** o, then it is easily seen that also U **m$\!\!\!/$ust** o; otherwise, if not $\beta.Q+R$ **moves-on** o, then there exists a computation $(\beta.Q+R)|o \Longrightarrow (\beta.Q+R)|o_k \not\longrightarrow$. But then, since $subj(\alpha) = subj(\beta)$, a unsuccessful computation $U|o \Longrightarrow (\alpha.P+\beta.Q+R)|o_k \not\xrightarrow{\tau}$ exists. Thus we have proven that $U \sqsubseteq_{must} T$. The converse (U **m$\!\!\!/$ust** o) is similar.

$\square$

3.5.3 Completeness proofs

In this subsection we shall use the symbol $\equiv$ to denote syntactic equality, to keep it distinct from proof-theoretic equality ($=$). We list in Table 3.5.3 a few derived laws, borrowed from [DH84], that will be useful in later proofs. I is any finite non-empty set.

$D1$	$\tau.P + \Omega = P + \Omega$
$D2$	$\alpha.(\sum_{i\in I} \tau.P_i) = \sum_{i\in I} \alpha.P_i$
$D3$	$\alpha.P + \Omega = \alpha.(P + \Omega) + \Omega$
$D4$	$P + \tau.Q = \tau.(P + Q) + \tau.Q$
$D5$	$\alpha.P + \alpha.Q = \alpha.P + \alpha.Q + \alpha.(P + Q)$
$D6$	$\alpha.P + \alpha.(P + Q + R) = \alpha.P + \alpha.(P + Q) + \alpha.(P + Q + R)$
$D7$	$\alpha.\tau.P = \alpha.P$
$D8$	$P + \tau.(P + Q) = \tau.(P + Q)$
$D9$	$\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$
$D10$	$\Omega + \alpha.(\sum_{i\in I} P_i + \Omega) = \Omega + \sum_{i\in I} \alpha.P_i$

Table 3.3: Derived Laws

Their derivation from the axioms and the inference rules of $\mathcal{F}$ is the same as in [DH84], except that each application of the action-prefix congruence rule has to be replaced by an application of either C0 - output or C1; this presents no problem. We have now to proof some preliminary lemmata about head normal forms. The first task is to show that each process has a provably equivalent hnf or Ω-hnf.

Lemma 3.5.12 *If P and Q are hnf's then there exists a hnf H s.t. $\mathcal{F} \vdash \tau.P + \tau.Q = H$.*

PROOF: We have three different cases, according to the form of P and Q.

1. $P \equiv \sum_i \tau.P_i$ and $Q \equiv \sum_j \tau.Q_j$ (we omit the ranges of the indeces to simplify notation). The schema of the proof is like that of Lemma 4.2.6 in [DH84], but an extra step is needed to reduce the resulting term into a form such that Property 3.2.6 of hnf's holds; in this step the law N3 is used. We only do in detail this part of the proof. By applying the laws D2, D8, N3 and N1 exactly like in [DH84] (just taking care of reading 'a' as 'α'), we can re-write $\tau.P + \tau.Q$ into an equivalent agent S such that:

$$\tau.P + \tau.Q = S \equiv \sum_{A\in L} \tau.S_A$$

where L is a suitable, possibly not saturated, collection of events sets, and for each $A \in L$, $S_A \equiv \sum_{\alpha\in\Lambda(A)} \alpha S_\alpha$, with $\Lambda(A) \subseteq \Lambda$ a suitable set of actions, with the property that $A = \{a \in Ev|$ for some $y, a(y) \in \Lambda(A)$ or $ay \in \Lambda(A)\}$. Now, it is possible that Property 3.2.6 of hnf's does not hold for such a S, i.e. there exist S_A and S_B such that S_A has a summand $\alpha.S_\alpha$ and S_B has a summand $\beta.S_\beta$, with $\alpha, \beta \in OA$ and $subj(\alpha) = subj(\beta)$, but such that $\alpha.S_\alpha$ is not a summand of S_B. Then, for suitable terms T, S_1, S_2, we can write:

$$\begin{array}{lll} S & = \tau.S_A + \tau.S_B + T & \\ & = \tau.(\alpha.S_\alpha + S_1) + \tau.(\beta.S_\beta + S_2) + T & \\ & = \alpha.S_\alpha + \tau.(\alpha.S_\alpha + S_1) + \beta.S_\beta + \tau.(\beta.S_\beta + S_2) + T & (D8) \\ & = \tau.(\alpha.S_\alpha + \beta.S_\beta + S_1) + \tau.(\alpha.S_\alpha + \beta.S_\beta + S_2) + T & (N3). \end{array}$$

By applying systematically this re-writing for each S_A and S_B, we can re-write S into a term S' for which the Property3.2.6 does hold, i.e., for a suitable normal fuction $g : Ev(L) \longrightarrow \mathcal{P}_1$,

$$S = \sum_{A \in L} \tau. \sum_{a \in A} g(a) \equiv S'.$$

Now, it remains to see whether L is saturated. If it is not, we can apply to S' the saturation construction of [DH84], based on the laws D5 and D6. The construction apply exactly in our case (only take care of reading '$a.p_a$' as '$g(a)$').

Case 2. (P in α-hnf and Q in τ-hnf) and case 3. (P and Q in α-hnf) easily reduce to case 1. via law D2 (see [DH84]). □

Lemma 3.5.13 *Let $\{P_i | \, 0 \leq i \leq k\}$, $k \geq 0$, be a set of hnf's; then there exists a hnf H s.t. $\mathcal{F} \vdash \sum_{0 \leq i \leq k} P_i = H$.*

PROOF: Consider any couple of summands P_i, P_j of $\sum_{0 \leq i \leq k} P_i$; we can reduce $P_i + P_j$ to a hnf in one of the following ways:

- if P_i, P_j are both in τ-hnf, we can apply first law D2 (with $\alpha = \tau$) and then the above lemma to reduce $P_i + P_j$ to a hnf;
- if P_i is in α-hnf and P_j is in τ-hnf, apply laws D4 and D7 to get: $P_i + P_j = \tau.(P_i + P_{jA}) + \tau.P_j$, where P_{jA} is any summand of P_j; by applying law N1, if needed, $P_i + P_{jA}$ is reduced to an α-hnf; applying the previous lemma we get the wanted result;
- if both P_i and P_j are in α-hnf, apply law N1, if needed, to reduce $P_i + P_j$ to a α-hnf.

By repeatedly applying the above reductions, we can re-write $\sum_{0 \leq i \leq k} P_i$ into a hnf . □

Proposition 3.5.14 (Proposition 3.2.8) *For every P s.t. $P \Downarrow$, there exists a process $h(P)$ in hnf s.t. $\mathcal{F} \vdash P = h(P)$.*

PROOF: (outline) The proof is by structural induction on P, and it is similar to the one for CCS in [DH84] (Proposition 4.2.16), only in the cases of Restriction and Parallel Composition there are some differences. In the former case ($P \equiv (y)P'$), when distributing the restriction over a summation $g(a)$, $a \in OUT$ (applying the law R2), an extra bound output summand may be generated; then we have to apply the law N1 in this way:

$\overline{x}(y).P_\alpha + \overline{x}(y).P_\beta = \overline{x}(y).(\tau.P_\alpha + \tau.P_\beta)$, to eliminate the double occurrence of the bound output summands. The Parallel Composition case ($P \equiv P_1|P_2$) is in turn an induction on $dpc(P_1|P_2)$. Both the cases of Parallel Composition ($P \equiv P_1 \mid P_2$) and the Summation ($P \equiv P_1 + P_2$) rely on the two previous lemmata. $\square$

Proposition 3.5.15 (Proposition 3.2.9) *For every P s.t. $P \Uparrow$, there exists an Ω-hnf $\Omega(P)$ s.t. $\mathcal{F} \vdash P = \Omega(P)$.*

PROOF: An easy structural induction on P, which relies on the laws D1, D3 and D10. $\square$

The following lemmata will be used in the proof of the completeness theorem.

Lemma 3.5.16 *Let P, Q be in hnf and $P \sqsubseteq\!\!\!\!\!\sim Q$. Then:*

i. *for each $\alpha \in Init(P) \cap Init(Q) \cap OA$, $P_\alpha \sqsubseteq\!\!\!\!\!\sim Q_\alpha$, and*

ii. *for each $\alpha = x(y) \in Init(P) \cap Init(Q) \cap IA$ and $z \in fn(P_\alpha, Q_\alpha, y)$,*
$P_\alpha\{z/y\} \sqsubseteq\!\!\!\!\!\sim Q_\alpha\{z/y\}$.

PROOF:
i. We first show the may part. Assume $\alpha = \overline{x}y$ and P_α **may** o. Then, taken a fresh variable $z \notin n(y, P, Q, o)$, let $o' \equiv x(z).[z = y]o$. Since $P|o' \stackrel{c}{\Longrightarrow} P_\alpha|[y = y]o \stackrel{\omega}{\Longrightarrow}$, it is P **may** o'. It follows Q **may** o'. But every successful computation for $Q|o'$ must begin like $Q|o \stackrel{c}{\Longrightarrow} Q_\alpha|[y = y]o$, from which it follows that also Q_α **may** o.

Suppose $\alpha = \overline{x}(w)$. We can suppose, by α-renaming, that $w \notin fn(P, Q, o)$. Now, let $o' \equiv x(w)[w \notin Obj(P, Q)]o$ and repeat the previous reasoning: we have that P **may** o', which implies Q **may** o'. But every successful computation from $Q|o'$ must be of the form $Q|o' \stackrel{c}{\Longrightarrow} (w)(Q_\alpha|[w \notin Obj(P, Q)]o) \stackrel{\omega}{\Longrightarrow}$ because each different computation, having an initial section of the form $Q|o' \stackrel{c}{\Longrightarrow} Q_\alpha|[y \notin Obj(P, Q)]o$, with $\beta \neq \alpha$ and $y \in Obj(Q)$, is unsuccessful. This implies Q_α **may** o. This suffices for the may part.

Now, let us consider the must part. Assume that $\alpha = \overline{x}y$ and P_α **must** o. Take $z \notin n(y, P, Q, o)$ and let $o' \equiv x(z).([z = y]o + [z \neq y]\omega.\mathbf{0}) + \tau.\omega.\mathbf{0}$. It is easily seen that P **must** o' and therefore Q **must** o'. Since $Q|o' \stackrel{c}{\Longrightarrow} Q_\alpha|([y = y]o + [y \neq y]\omega.\mathbf{0})$, it follows that Q_α **must** o. If $\alpha = \overline{x}(w)$ and P_α **must** o, the above reasoning can be repeated with the observer:

$$o' \equiv x(z).([z \notin Obj(P, Q)]o \ + \sum_{y \in Obj(P,Q)} [z = y]\omega.\mathbf{0}) + \tau.\omega.\mathbf{0},\ z \text{ fresh.}$$

Thus we have proven part i.

ii. Suppose $\alpha = x(y)$ and let $z \in fn(P_\alpha, Q_\alpha, y)$ and $\sigma = \{z/y\}$. Suppose $P_\alpha\sigma$ **may** o. We have to show that $Q_\alpha\sigma$ **may** o as well. Now, consider $o' \equiv \overline{x}z.o$. We have P **may** o', since $P|o' \stackrel{c}{\Longrightarrow} P_\alpha\sigma|o \stackrel{\omega}{\Longrightarrow}$. Therefore Q **may** o' and this implies in turn that $Q|o' \stackrel{c}{\Longrightarrow} Q_\alpha\sigma|o \stackrel{\omega}{\Longrightarrow}$, from which it follows $Q_\alpha\sigma$ **may** o.

Suppose $P_\alpha\sigma$ **must** o; we have to show that $Q_\alpha\sigma$ **must** o as well. Let $o' \equiv \overline{x}z.o + \tau.\omega.\mathbf{0}$. We have that P **must** o' which implies Q **must** o'. But, since $Q|o' \stackrel{c}{\Longrightarrow} Q_\alpha\sigma|o$, it follows $Q_\alpha\sigma$ **must** o. □

The following definition gives a syntactical relation for 'comparing' hnf's at their top levels. It is a *trait d'union* between the semantical relation $\sqsubseteq^+$ and the proof-theoretic one $\sqsubseteq$.

Definition 3.5.17 *Let P and Q be in hnf. Then $P \prec Q$ iff $Init(P) = Init(Q)$ and:*

i. $P \equiv \sum_{a \in A} g_1(a)$ *and* $Q \equiv \sum_{a \in A} g_2(a)$, *or*

ii. $P \equiv \sum_{A \in L} \tau.P_A$ *and* $Q \equiv \sum_{A \in K} \tau.Q_A$, *with* $K \subseteq L$, *or*

iii. $P \equiv \sum_{A \in L} \tau.P_A$ *and* $Q \equiv \sum_{a \in A} g_2(a)$, *with* $A \in L$.

The relationship between $\sqsubseteq^+$ and $\prec$ is given by the following:

Lemma 3.5.18 *Let P and Q be in hnf and $P \sqsubseteq^+ Q$. Then $P \prec Q$.*

Proof: Since $P \sqsubseteq_{may} Q$, $P \Downarrow$ and $P \sqsubseteq_{must} Q$, from Lemma 3.5.4 it easily follows that $Init(P) = Init(Q)$ (up to α-equivalence). Let us prove the second condition of Definition 3.5.17. There are three cases, according to the form of P and Q.

i. P and Q are in α-hnf. Then, since $Init(P) = Init(Q)$, condition i. of the definition holds.

ii. Suppose that P and Q are like in case ii. of the definition. We show that $K \subseteq L$. Now since $Init(P) = Init(Q)$, it is $Ev(K) = Ev(L)$. Since $L = \{A_1, ..., A_n\}$ is saturated, it suffices to show that: for each $B \in K$ there exists $A_i \in L$ s.t. $A_i \subseteq B$ (this suffices because $A_i \subseteq B \subseteq Ev(K) = Ev(L)$ implies $B \in L$). By contradiction, suppose there exists $B \in K$ s.t. for each $A_i \in L$, $A_i \not\subseteq B$, i.e. for each i, $0 \leq i \leq n$, there exists an $a_i \in A_i - B$. Let $A = \{a_1, ..., a_n\}$ and define for each $a \in A$: $o(a) = \overline{x}y.\omega.\mathbf{0}$ if $a = x \in IN$, $o(a) = x(y).\omega.\mathbf{0}$ otherwise, where y is any name. Also, define $o \equiv \sum_{a \in A} o(a)$. Now, it is clear that P **must** o, but Q $\not$**must** o, due to the unsuccessful computation $Q|o \stackrel{\tau}{\longrightarrow} \sum_{a \in B} g(a)|o$. Thus we arrived at a contradiction. Therefore $K \subseteq L$, and the thesis follows.

iii. Suppose P and Q are like in case iii. of the definition. Suppose by contradiction that $A \notin L$. Then we can repeat the above construction with $B = A$. Finally, if P is in α-hnf, then $P \stackrel{\tau}{\not\longrightarrow}$. But being $P \sqsubseteq^+_{must} Q$, it is also $Q \stackrel{\tau}{\not\longrightarrow}$, thus Q as well needs to be in α-hnf and we fall in case i. □

Lemma 3.5.19 *Let P and Q be in hnf and $P \prec Q$. Then $P \sqsubseteq Q$ if for each $\alpha \in Init(P)$:*

i. $\alpha \in OA$ *implies* $P_\alpha \sqsubseteq Q_\alpha$, *and*

ii. $\alpha = x(y) \in IA$ *implies for each* $z \in fn(P_\alpha, Q_\alpha, y)$, $P_\alpha\{z/y\} \sqsubseteq Q_\alpha\{z/y\}$.

PROOF: Since $Init(P) = Init(Q)$, it is $Ev(P) = Ev(Q)$. Let g_1 and g_2 be the functions associated with P and Q, respectively. By applying the rules C0 for $\alpha \in OUT$ and C1 for $\alpha \in IN$, it is straightforward to show that for each $a \in Ev(P) = Ev(Q)$, $g_1(a) \sqsubseteq g_2(a)$. Then we consider the term $R \equiv P[g_2(a)/g_1(a),\ a \in Ev(P)]$ (where $P[Y/X]$ denotes the agent obtained from P by replacing each (sub)term X in P with Y), and using the rules in C0, we show that $P \sqsubseteq R$. Now the proof that $R \sqsubseteq Q$ proceeds, using τ-laws, exactly like proof of Lemma 4.3.6.i) in [DH84] (just read '$a.q_a$' as 'g_2(a)'.) □

Lemma 3.5.20 *Suppose P is in Ω-hnf and Q is in hnf or Ω-hnf. If $P \underset{\sim}{\sqsubset} Q$ then for each $\alpha \in Init(P) \cap Init(Q)$:*

i. *$\alpha \in OA$ implies $P_\alpha + \Omega \underset{\sim}{\sqsubset}^{+} Q_\alpha + \Omega$, and*

ii. *$\alpha = x(y) \in IA$ implies for each $z \in fn(P_\alpha, Q_\alpha, y)$,*
$P_\alpha\{z/y\} + \Omega \underset{\sim}{\sqsubset}^{+} Q_\alpha\{z/y\} + \Omega$.

PROOF: Similar to, but easier than, that of Lemma 3.5.16. □

Lemma 3.5.21 *Suppose P is in Ω-hnf and Q is in hnf or Ω-hnf. Then $P \sqsubseteq Q$ if $Init(P) \subseteq Init(Q)$ and for each $\alpha \in Init(P)$:*

i. *$\alpha \in OA$ implies $P_\alpha + \Omega \sqsubseteq Q_\alpha + \Omega$, and*

ii. *$\alpha = x(y) \in IA$ implies for each $z \in fn(P_\alpha, Q_\alpha, y), P_\alpha\{z/y\} + \Omega \sqsubseteq Q_\alpha\{z/y\} + \Omega$.*

PROOF: Similar to, but easier than, that of Lemma 3.5.19. □

We can finally give the completeness proof:

Theorem 3.5.22 (Theorem 3.2.11) *$P \underset{\sim}{\sqsubset}^{+} Q$ implies $\mathcal{F} \vdash P \sqsubseteq Q$.*

PROOF: The proof is by induction on $dpc(P)$.

Basis. $dpc(P) = 0$. If $P \Uparrow$ then $\Omega(P) \equiv \Omega$ exists (Proposition 3.2.9 and the thesis follows from law Omega. Otherwise, $P \Downarrow$ and therefore $h(P)$ exists (Proposition 3.2.8). Then it is also $Q \Downarrow$ and thus $h(Q)$ exists. It must be $h(P) \equiv \tau.\mathbf{0}$ or $h(P) \equiv \mathbf{0}$ and, since $Init(h(P)) = Init(h(Q))$, it is the same for $h(Q)$. Now, if $h(P) \equiv \tau.\mathbf{0}$ and $h(Q) \equiv \mathbf{0}$, the thesis follows from N4. Otherwise it must be $h(P) \equiv h(Q) \equiv \mathbf{0}$.

Inductive Step. $dpc(P) > 0$. If $P \Uparrow$, $\Omega(P)$ exists and Q has an (Ω-)hnf, say H. Since $\Omega(P) \underset{\sim}{\sqsubset}_{may} H$, from Lemma 3.5.4 we get $Init(\Omega(P)) \subseteq Init(H)$. By applying Lemma 3.5.20, we have that for each $\alpha \in Init(\Omega(P))$:

i. $\Omega(P)_\alpha + \Omega \underset{\sim}{\sqsubset}^{+} H_\alpha + \Omega$, for $\alpha \in OA$;

ii. $\Omega(P)_\alpha\sigma + \Omega \underset{\sim}{\sqsubset}^{+} H_\alpha\sigma + \Omega$, for each $\alpha = x(y) \in IA$ and each $\sigma = \{z/y\}$ with $z \in fn(\Omega(P)_\alpha, H_\alpha, y)$.

Since for each considered σ: $dpc(\Omega(P)_\alpha\sigma) < dpc(\Omega(P)) = dpc(P)$, we can apply the inductive hypothesis and get that the above relations i. and ii. still hold when replacing " $\sqsubseteq^+_{\sim}$ " with " $\sqsubseteq$ ". By applying Lemma 3.5.21, we obtain the claim. If $P \Downarrow$ then also $Q \Downarrow$ and $h(P) \equiv H_1$ and $h(Q) \equiv H_2$ both exist. Moreover, by possibly applying law D7 ($\alpha.X = \alpha.\tau.X$), we can suppose H_1 is in a form s.t. for each $\alpha \in Init(H_1)$, $H_{1\alpha}$ is of the form $\tau.H'$, for some H'. From Lemma 3.5.18, we get $H_1 \prec H_2$. Moreover, from Lemma 3.5.16, we obtain that for each $\alpha \in Init(H_1) = Init(H_2)$:

a. $H_{1\alpha} \sqsubseteq^+_{\sim} H_{2\alpha}$, $\alpha \in OA$;

b. $H_{1\alpha}\sigma \sqsubseteq^+_{\sim} H_{2\alpha}\sigma$, for $\alpha = x(y) \in IA$ and for each $\sigma = \{z/y\}$ with $z \in fn(H_{1\alpha}, H_{2\alpha}, y)$.

(Recall that $H_{1\alpha} \xrightarrow{\tau}$) By applying the inductive hypothesis, we obtain that relations a. and b. still hold when replacing " $\sqsubseteq^+_{\sim}$ " with " $\sqsubseteq$ ". By applying Lemma 3.5.19 we then obtain the thesis. □

3.5.4 Denotational semantics

Definition 3.5.23 (Finite-Breadth Approximants) *For each P, define the set $FBA(P) \subseteq FSNF$ inductively as follows:*

- $$FBA(op(P_1, ..., P_k)) = \{snf(op(N_1, ..., N_k)) \mid N_i \in FBA(P_i),\ for\ i = 1, ..., k\}$$

  for each k-ary operator op different from input prefix, $k = 0, 1$ or 2.

- $$FBA(x(y).P) = \{x(w).(\textstyle\sum_{z\in F}[w = z]N_z + [w \notin F]\Omega) \mid F \subseteq_{\text{fin}} \mathcal{N},\ N_z \in FBA(P\{z/w\})\ and\ w\ fresh\ \}.$$

Notice that, by construction, $N \in FBA(P)$ implies $N \sqsubseteq_{must} P$.

Proposition 3.5.24 *Let P and Q be processes. Then:*

a. *$N \in FBA(P)$ implies $[\![N]\!] = [N]$;*

b. *$[\![FBA(P)]\!]$ is directed in $\mathcal{D}$ and $[\![P]\!] = sup[\![FBA(P)]\!]$;*

c. *$N \in FSNF$ and $N \sqsubseteq_{must} P$ implies there exists $M \in FBA(P)$ such that $N \sqsubseteq_{must} M$;*

d. *If for each $N \in FBA(P)$ $N \sqsubseteq_{must} Q$ then $P \sqsubseteq_{must} Q$.*

PROOF: (outline) In the proof we use the notion of *finite-breadth term*, a term in which every input prefix $x(w)$. is followed by a subterm of the form $(\sum_{z\in F}[w = z]P_z + [w \notin F]\Omega)$, with $w \notin F$. Fsnf's are of course particular cases of finite-breadth terms.

a. Show, by structural induction, that for each finite-breadth term P, $snf(P) \in FSNF$ and $[\![P]\!] = [snf(P)]$.

b. The proof is a structural induction on P, which exploits the continuity of the functions f_{op}, out and inp. The only difficult case is when $P = x(y).R$ for some process R. We consider only this case. Define for each z, $R_z = R\{^z/_y\}$. We have:

$$\begin{array}{rcl} [\![FBA(P)]\!] & = & \{[x(w).(\sum_{z\in F}[w=z]N_z + [y \notin F]\Omega)] \mid F \subseteq_{\mathrm{fin}} \mathcal{N}, \\ & & N_z \in FBA(R_z), \text{ and } w \text{ fresh }\} \\ & & \text{(Definition of } FBA \text{ and part a.)} \\ & = & \{inp(x,f) \mid f \in K\} \;\; \text{(Definition of } inp\text{)} \end{array}$$

where K is the set of functions $f : \mathcal{N} \longrightarrow PO$ of the kind:

$$\lambda z.\ \mathit{if}\ z \in F\ \mathit{then}\ [N_z]\ \mathit{else}\ [\Omega], \text{ with } F \subseteq_{\mathrm{fin}} \mathcal{N} \text{ and } N_z \in FBA(R_z).$$

Now it is easy to check that K is directed in $(\mathcal{N} \longrightarrow \mathcal{D})$: this is an easy consequence of the fact that for each z, by inductive hypothesis $[\![FBA(R_z)]\!]$ is directed in $\mathcal{D}$. Therefore, due to continuity of $inp(x,-)$, we have:

$$\begin{array}{rl} sup[\![FBA(P)]\!] & = sup\{inp(x,f) \mid f \in K\} \\ & = inp(x, sup\,K). \end{array}$$

On the other side:

$$\begin{array}{rll} [\![P]\!] & = inp(x, \lambda z.[\![R_z]\!]) & \\ & = inp(x, \lambda z.sup\,[\![FBA(R_z)]\!]) & \text{(Inductive hypothesis)} \\ & = inp(x,\ sup\,H) & \end{array}$$

where $H = \{f \mid f$ is a function of the form: $\lambda z.[N_z]$ with $N_z \in FBA(R_z)\}$. We have $K \subseteq H$ and thus $sup\,K \preceq sup\,H$. On the other side, it is easily seen that each function of H is approximable by a chain of functions in K. In fact, given any $f = \lambda z.[N_z]$ in H, consider the chain $C_f = \{f_i \mid i \geq 0\} \subseteq K$, with the functions f_i, $i \geq 0$, defined in this way: fixed any infinite chain of finite sets $F_0 \subseteq F_1 \subseteq F_2 \subseteq \ldots$ s.t. $\cup_{i\geq 0} F_i = \mathcal{N}$, define $f_i = \lambda z.\mathit{if}\ z \in F_i\ \mathit{then}\ [N_z]\ \mathit{else}\ [\Omega]$; it is easy to check that $sup\,C_f = f$. Now apply the following fact from cpo theory: given $X, Y \subseteq D$ cpo, with sup's s_x, s_y respectively, if for each $y \in Y$ there exists $A_y \subseteq X$ s.t. $sup\,A_y = y$, then $s_y \preceq s_x$. We obtain that $sup\,H \preceq sup\,K$, and therefore $sup\,H = sup\,K$; therefore $inp(x, sup\,K) = inp(x, sup\,H)$ and the thesis follows.

c. It is sufficient to prove that for any finite-breadth term P and any process Q it holds that $P \sqsubseteq_{must} Q$ implies there exists $M \in FBA(P)$ such that $P \sqsubseteq_{must} M$. This is proved by induction on the depth of the formal proof of $P \sqsubseteq_{must} Q$ within the proof system for $\sqsubseteq_{must}$. The relevant case is when C1 is the last applied rule. In that case, techniques similar to the ones used in the above item b. can be employed.

d. The claim easily follows from Lemma 3.4.1 and from the following assertion: for any process P and any restriction-free observer o, if P **must** o then there exists $N \in FBA(P)$ such that N **must** o. Thus, it is sufficient to prove the latter assertion. Indeed, it holds that for any process P and any restriction-free observer o, if P **must** o then there exists $M \in FSNF$ such that M **must** o (this can be proved by specializing the proof for the infinite case of Proposition 3.5.28; only notice that, for this finite case, the proof of the latter can be re-written without mentioning any recursion law). Now, since $M \sqsubseteq_{must} P$, from part c. of this proposition we get that there exists $N \in FBA(P)$, $M \sqsubseteq_{must} P$, s.t. $M \sqsubseteq_{must} N$; therefore N **must** o, and the assertion is proven.

□

Now we can prove:

Theorem 3.5.25 (Full abstraction for finite processes) *For each P and Q, $P \sqsubseteq_{must} Q$ iff $[\![P]\!] \preceq [\![Q]\!]$.*

PROOF:

(If) We first prove that given any $N \in FSNF$, $[\![N]\!] \preceq [\![Q]\!]$ implies $N \sqsubseteq_{must} Q$. In fact, $[\![FBA(Q)]\!]$ is directed in $\mathcal{D}$ and $[\![N]\!]$ is compact in $\mathcal{D}$; hence, due to Proposition 3.5.24, part b., there exists $N' \in FBA(Q)$ s.t. $[\![N]\!] \preceq [\![N']\!]$, i.e. $N \sqsubseteq_{must} N'$ for part a. of Proposition 3.5.24. But $N' \sqsubseteq_{must} Q$, and thus $N \sqsubseteq_{must} Q$. Now, $[\![P]\!] \preceq [\![Q]\!]$ implies, for the previous proposition, part b., that for each $N \in FBA(P)$ it is $[\![N]\!] \preceq [\![Q]\!]$ and therefore $N \sqsubseteq_{must} Q$. Applying part d. of Proposition 3.5.24, we obtain then $P \sqsubseteq_{must} Q$.

(Only If) In virtue of Proposition 3.5.24, part c., for each $N \in FBA(P)$, there exists a $N' \in FBA(Q)$ s.t. $N \sqsubseteq_{must} N'$, i.e. $[\![N]\!] \preceq [\![N']\!]$ in $\mathcal{D}$, for part a. of the same proposition. From this fact it follows that $sup[\![FBA(P)]\!] \preceq sup[\![FBA(Q)]\!]$, i.e. $[\![P]\!] \preceq [\![Q]\!]$ in $\mathcal{D}$, for part b. of Proposition 6.7. □

3.5.5 Dealing with infinite agents

All proofs given in the soundness subsection for the finite calculus easily extend to arbitrary agents via the following lemma, that generalizes the assertions after Definition 3.5.2 to the infinite agents. Essentially, it says that, even in the case of infinite agents, if two processes always generate *finite* equivalent sequences of silent moves, when run in parallel with the same observer, then they are must-equivalent.

Lemma 3.5.26 *Given P, P' processes and o, o' observers, suppose $P|o \leq P'|o'$. Then P* **m̸ust** *o implies P'* **m̸ust** *o'.*

PROOF: Consider the possible kinds of failing computation for $P|o$. By definition of $\leq$ between experiments, if a finite failing computation exists for $P|o$, then it must also exist for $P'|o'$.

The only remaining case is therefore $P|o = e_0 \xrightarrow{\tau} e_1 \xrightarrow{\tau} e_2 \xrightarrow{\tau} \cdots$ and $e_i \not\xrightarrow{\omega}$ for $i \geq 0$. Suppose by contradiction that P' **must** o'. Consider the *computation tree* from $P'|o'$ (with τ-actions only), with branches pruned to obtain a tree whose leaves are all those nodes that can give rise to $\xrightarrow{\omega}$ transitions; call this tree $\mathcal{T}$. Since P' **must** o', $\mathcal{T}$ has the property that for each non-leaf node e, $e\downarrow$. It is not difficult to show that if $e\downarrow$ then e has a finitely many τ-derivatives; therefore $\mathcal{T}$ is finitary. By definition of $\leq$ between experiments, for each $k \geq 0$, there exists in $\mathcal{T}$ a path $(P'|o' = f_0, f_1, ..., f_k)$ with the property that $(e_0 \xrightarrow{\tau} ... \xrightarrow{\tau} e_k)$ and $(f_0 \xrightarrow{\tau} ... \xrightarrow{\tau} f_k)$ are equivalent sequences of τ-transitions. By applying Koenig's lemma, there exists in $\mathcal{T}$ an infinite path; but this contradicts P' **must** o'. □

The following lemma, that provides a different characterization of the must preorder, is necessary for proving the next proposition.

Lemma 3.5.27 (Lemma 5.1) *The must testing preorder $<_M$ obtained by taking the set of observers without restriction coincides with $\lesssim_{must}$.*

PROOF: Obviously $P \lesssim_{must} Q$ implies $P <_M Q$.

To show the reverse implication, suppose that $P <_M Q$ and P **must** o, where o is any observer.

By possibly α-renaming o, we can suppose w.l.o.g. that each restriction (y) in o is such that $y \notin fn(P, Q, o)$. Fixed a restriction (y) of o, consider now the observer $r(o, y)$ where, intuitively, all restrictions (y) have been discarded; more precisely $r(o, y)$ is defined inductively in this way: $r(op(P_1, ..., P_k), y) = op(r(P_1, y), ..., r(P_k, y))$, for any k-ary operator *op*, $k = 0, 1$ or 2, different from (y), $r((y)o', y) = r(o', y)$. It can be easily proved, by structural induction on o and by exploiting Lemma 6.4 in case $o = (y)o'$, that $R|o$ and $R|r(o, y)$ are equivalent as experiments, where R is P or Q. By repeatedly applying this result, we can eventually get rid of all the restrictions of o, i.e. obtain a restriction-free observer o', such that $R|o$ and $R|o'$ are equivalent as experiments, where R is P or Q.

Thus P **must** o'; but being $P <_M Q$, we have Q **must** o', and therefore, since $Q|o'$ is equivalent to $Q|o$, also Q **must** o. □

Proposition 3.5.28 (Proposition 3.4.2) *For any process P and for any restriction-free observer o, if P* **must** *o then there exists $D \in FIN(P)$ such that D* **must** *o.*

PROOF: In this proof "$X \sqsubseteq_{must} Y$" will be used as a shorthand for "$\mathcal{A} - \{\omega - induction, REC2\} \vdash X \sqsubseteq_{must} Y$". It is a routine task of algebraic semantics (see [Hen88]) to show that, for any finite D and any P, if $D \sqsubseteq_{must} P$, then there exists $D' \in FIN(P)$ such that $D \sqsubseteq_{must} D'$. Thus, to prove our claim, it suffices to show there exists a finite D s.t. $D \sqsubseteq_{must} P$ and D **must** o. Actually, we will show a little stronger result, i.e.: if P **must** o then there exists a finite strong normal form N s.t. $N \sqsubseteq_{must} P$ and N **must** o. Note that, by hypothesis, o cannot perform any bound output action.

Suppose now that P **must** o and consider the computation tree $\mathcal{T}$ from $P|o$, defined like in the proof of Lemma 3.5.26. Since P **must** o and $\mathcal{T}$ is finitary, it follows from Koenig's lemma that $\mathcal{T}$ is finite. The proof is by induction on the *depth of communication of* $\mathcal{T}$, i.e. the maximal number of communications between the process and the observer in a path from the root to a leaf in $\mathcal{T}$. It is not hard to show that this number does not change if we consider the computation tree generated by a $P'|o$, for any P' must-equivalent to P.

If $P \Uparrow$, then it must be $o \stackrel{\omega}{\longrightarrow}$, and we can take $N \stackrel{def}{=} \Omega$. If $P \Downarrow$, then we can suppose that P is in hnf. There are two cases.

1. $P = \sum_{\alpha \in A} \alpha.P_\alpha$. Agent N is built up out of the finite agents N_α, $\alpha \in A$, defined as follows:

   - $\alpha = x(y)$. Consider the set:

   $$Pairs(\alpha) = \{(z, o')| \quad o \stackrel{\tau}{\longrightarrow} o_1 \stackrel{\tau}{\longrightarrow} \cdots \stackrel{\tau}{\longrightarrow} o_k \stackrel{\overline{x}z}{\longrightarrow} o', \text{ with } o_i \stackrel{\omega}{\not\longrightarrow}, \text{ for } 0 \leq i \leq k\}\,.$$

   For each $(z, o') \in Pairs(\alpha)$, we have $P|o \stackrel{c}{\Longrightarrow} P_\alpha\{z/y\}|o'$, thus $P_\alpha\{z/y\}$ **must** o'; therefore, for each z such that some pair (z, o') is in $Pairs(\alpha)$,

   $$P_\alpha\{z/y\} \textbf{ must } \sum_{(z,o')\in Pairs(\alpha)} \tau.o'.$$

   By inductive hypothesis, there exists a finite strong normal form $N(\alpha, z) \sqsubseteq_{must} P_\alpha\{z/y\}$ such that $N(\alpha, z)$ **must** $\sum_{(z,o')\in Pairs(\alpha)} \tau.o'$, i.e $N(\alpha, z)$ **must** o', for each $(z, o')\in Pairs(\alpha)$. Note that $N(\alpha, z)$ does not depend on y, thus, by possibly α-renaming y in P, we can suppose $y \notin \Pi_1(Pairs(\alpha))$ and $y \notin fn(N(\alpha, z))$ (Π_1 is the projection on the first coordinate). Finally, define N_α to be

   $$\sum_{z\in\Pi_1(Pairs(\alpha))} [y = z]N(\alpha, z) + [y \notin \Pi_1(Pairs(\alpha))]\Omega\,.$$

   By applying Match and Mismatch laws it is easy to check that for each name w, $N_\alpha\{w/y\} \sqsubseteq_{must} P_\alpha\{w/y\}$.

   - $\alpha = \overline{x}y$. If the set

   $$S = \{o'| \quad o \stackrel{\tau}{\longrightarrow} o_1 \stackrel{\tau}{\longrightarrow} \cdots \stackrel{\tau}{\longrightarrow} o_k \text{ and } P|o_k \stackrel{c}{\longrightarrow} P_\alpha|o', \; o_i \stackrel{\omega}{\not\longrightarrow} \text{ for } 0 \leq i \leq k\}$$

   is not empty, then P_α **must** $\sum_{o'\in S} \tau.o'$; by induction hypothesis, there exists a fsnf $N' \sqsubseteq_{must} P_\alpha$ with the property that for each $o' \in S$, N'_α **must** o'; take N_α to be N'_α. If $S = \emptyset$, take N_α to be Ω.

- $\alpha = \overline{x}(y)$. We can suppose, by possibly α-renaming, that $y \notin fn(o)$. If the set

$$S = \{o' | \quad \begin{array}{l} o \xrightarrow{\tau} o_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} o_k \text{ and } P|o_k \xrightarrow{c} (y)(P_\alpha|o'), \\ o_i \not\xrightarrow{\omega} \text{ for } 0 \leq i \leq k\} \end{array}$$

is not empty, then P_α **must** $\sum_{o' \in S} \tau.o'$, and, by induction hypothesis, there exists a fsnf $N' \sqsubseteq_{must} P_\alpha$ with the property that for each $o' \in S$, N'_α **must** o'; take N_α to be N'_α. If $S = \emptyset$, take N_α to be Ω.

Now define N to be the fsnf $\sum_{\alpha \in A} \alpha.N_\alpha$. By using the rules C0 and C1, we see that $N \sqsubseteq_{must} P$ and furthermore by construction N **must** o.

2. $P = \sum_{A \in L} \tau. \sum_{\alpha \in A} \alpha.P_\alpha$. For each $A \in L$, $\sum_{\alpha \in A} \alpha.P_\alpha$ **must** o; repeat the above construction, thus there exists a fsnf N_A s.t. $N_A \sqsubseteq_{must} \sum_{\alpha \in A} \alpha.P_\alpha$ and N_A **must** o; then take N to be $\sum_{A \in L} \tau.N_A$.

$\square$

Theorem 3.5.29 (Theorem 5.1) $[\![P]\!] \preceq [\![Q]\!]$ *in* $\mathcal{D}$ *if and only if* $P \sqsubseteq\!\!\!\sim^{+}_{must} Q$.

PROOF: Recall from [HP80] that:

$$\{[\![D]\!]\rho |\, D \in FIN(P)\} \text{ is directed in } \mathcal{D} \text{ and } [\![P]\!]\rho = sup\,\{[\![D]\!]\rho |\, D \in FIN(P)\}\,. \tag{3.1}$$

Define for each closed non-finite P, $FBA(P) = \{N |\, N \in FBA(D) \text{ for some } D \in FIN(P)\}$. From Proposition 3.5.24, Theorem 3.5.25 and 3.1 above, it easily follows that $[\![FBA(P)]\!] = \{[\![N]\!] | N \in FBA(P)\}$ is directed in $\mathcal{D}$ and:

$$[\![P]\!] = sup\,\{[\![N]\!] | N \in FBA(P)\}\,. \tag{3.2}$$

On the other side, it is easy to check that for each closed P and Q, we have $P \sqsubseteq_{must} Q$ if and only if for each $D \in FIN(P)$ there exists $E \in FIN(Q)$ such that $D \sqsubseteq_{must} E$ (to prove the 'if' part, exploit $\omega - induction$ rule; for the 'only if' part, see the proof of Proposition 3.5.28). From Proposition 3.5.24, it therefore follows that:

$$P \sqsubseteq_{must} Q \text{ iff } \forall\ N \in FBA(P)\ \exists\ M \in FBA(Q) \text{ s.t. } N \sqsubseteq_{must} M\,. \tag{3.3}$$

Recalling that $[\![FBA(P)]\!]$ is a set of compact elements in $\mathcal{D}$, it follows from (3.2): $[\![P]\!] \preceq [\![Q]\!]$ in $\mathcal{D}$ if and only if for each $N \in FBA(P)$ there exists an $M \in FBA(Q)$ such that $[\![N]\!] \preceq [\![M]\!]$ in $\mathcal{D}$. Since it has already been proved for finite agents N and M that $[\![N]\!] \preceq [\![M]\!]$ in $\mathcal{D}$ if and only if $N \sqsubseteq_{must} M$, we have from (3.3) that $[\![P]\!] \preceq [\![Q]\!]$ in $\mathcal{D}$ iff $P \sqsubseteq_{must} Q$, and the thesis follows from the soundness and completeness of $\sqsubseteq_{must}$ w.r.t. $\sqsubseteq\!\!\!\sim^{+}_{must}$. $\square$

Chapter 4

Symbolic Semantics

In this chapter, building on previous work by Hennessy and Lin on static value-passing process algebras [HL92, HL93], we provide a framework that yields alternative "efficient" characterization for various π-calculus equivalences based on bisimulation and a complete proof system to reason about them. An additional advantage of this framework is that it sheds light on the conceptual differences among known different forms of π-calculus bisimulation. Our attention will be confined to strong bisimulations, but we do not see any serious obstacle in extending our results to the weak ones.

The basic theory of bisimulation for the π-calculus has been introduced in [MPW92]. The fundamental notion is that of *ground* bisimulation; it has the same conceptual simplicity of the CCS one and suggests a natural strategy for equivalence checking. However, due to name passing, the definition-based verification technique runs into serious efficiency problems. On input actions, a case analysis on the received names is needed to check that receiving equal names leads to "equivalent" states. To see this, consider the processes $P \stackrel{def}{=} a(y).P'$ and $Q \stackrel{def}{=} a(y).Q'$. To check that P and Q are ground bisimilar, following the definition (see Section 2.3) we would have to verify that P' and Q' are bisimilar for all possible instantiations of y with a name occurring free in P and Q or with a fresh name. Of course, performing multiple checks for each input action may lead to exponential explosion.

As we saw in Chapter 2, input prefix also introduces another problem: it does not preserve ground bisimulation equivalence. This leads to considering the maximal induced congruence, obtained by closing the ground equivalence under name-substitutions [MPW92]. Checking for congruence of two terms by relying on the original definition would require performing several (one for each "relevant" substitution) ground equivalence checks.

A simple example is sufficient to appreciate that, when verifying π-calculus bisimulation with such straightforward techniques, many performed checks are indeed useless.

Consider the two processes

$$P_1 \stackrel{def}{=} a(y).P \text{ and } P_2 \stackrel{def}{=} a(y).([y = z]P + [y \neq z]P)$$

where y and z are distinct.

It should be immediate to establish that P_1 and P_2 are equivalent. But, checking their equivalence by directly relying on its definition, requires checking that P and $[y = z]P + [y \neq z]P$ are bisimilar when y is replaced by z or by each of the free names of P. This could be very costly; it is however evident that, beside y, z is the only name that matters.

A significant gain in efficiency could be obtained by finding a systematic way to prune non-essential cases when performing the case analysis. To this aim, the idea pursued in this chapter is that of setting up a framework where case analysis can be performed *symbolically*. First, a new transition system for the π-calculus is introduced, where the logical constraints that make a transition possible are made explicit. Then, a new bisimulation, which we refer to as *symbolic*, is defined, that performs the case-analysis directly on these logical constraints. It will be proved that symbolic bisimulation can be used to establish the standard ones.

The symbolic transitions are of the form $P \xrightarrow{\phi,\,\alpha} P'$, where ϕ is a *boolean constraint* over names that has to hold for the transition to take place, and α is a standard π-calculus action. A typical symbolic transition is $[x = y]\alpha.P \xrightarrow{[x=y],\,\alpha} P$, saying that action α can be performed under any interpretation of names satisfying $x = y$. A boolean constraint is in general built from the basic constraints $[x = y]$ via the standard boolean connectives.

Symbolic bisimulation is defined on the top of the symbolic transition system, in a way that is parametric over the language of boolean constraints. This leads to distinct symbolic equivalences $\simeq^{\phi}$, depending on the boolean constraint ϕ. Informally, to verify $P \simeq^{\phi} Q$, it is required to find, for each symbolic move $P \xrightarrow{\psi,\,\alpha} P'$ of P, a case-partition of the condition $\phi \wedge \psi$, such that each subcase entails a matching symbolic move for Q. As an example, the equivalence $P_1 \simeq^{true} P_2$ of the above mentioned processes is readily verified by partitioning condition $true$ as $\{[z = y], [z \neq y]\}$; this is sufficient because each of these two conditions entails that $[y = z]P + [y \neq z]P$ is equal to P.

Symbolic bisimulation is related to the standard ones by the following "consistency and adequacy" theorem ($\sim$ denotes as usual ground equivalence):

$$P \simeq^{\phi} Q \text{ iff for each name-substitution } \sigma \text{ satisfying } \phi,\ P\sigma \sim Q\sigma$$

where a name-substitution σ *satisfies* ϕ if the result of applying σ to ϕ is a tautology; here, $P\sigma$ denotes the result of applying σ to P .

The above statement tells us that each symbolic equivalence $\simeq^{\phi}$ is the closure of the ground equivalence under all name-substitutions satisfying ϕ. Thus, for example, the congruence (i.e. the closure w.r.t. *all* substitutions) will be recovered as the symbolic bisimulation $\simeq^{true}$. Ground bisimulation equivalence of two specific processes P and Q will be instead recovered as a symbolic bisimulation $P \simeq^{\phi(P,Q)} Q$, where $\phi(P,Q)$ is a constraint imposing that all free names in P and Q be distinct.

We also present a proof system to reason about symbolic bisimulation. The statements derivable within the system are of the form $\phi \triangleright P = Q$ and the system is sound and complete in the sense that $\phi \triangleright P = Q$ is derivable if and only if $P \simeq^{\phi} Q$. By taking advantage of the symbolic transitional semantics, the proof of completeness is, by and large, a symbolic version of the classical proof for strong bisimulation over CCS [Mil89]. Additional complications are however introduced by the fact that the boolean condition ϕ may also constraint the communication capabilities of processes.

The symbolic characterization of the standard equivalences has also an additional advantage; it sheds new light on the conceptual difference between different forms of bisimulations for the π-calculus. As we saw, two forms of ground bisimulations (each inducing a different congruence) exist, the early form and the late one. Intuitively, they correspond to two different instantiation strategies for the formal parameter of input actions: in the first strategy (early), the instantiation is performed at the moment of inferring the input action, while in the other (late) it is performed later, when a communication is actually inferred. In *open* bisimulation, introduced by Sangiorgi in [San93], the instantiation is delayed as much as possible; this yields an equivalence much stronger than the early and late ones. Our symbolic formulation indicates that each of the mentioned strategies corresponds to a different degree of generality in performing case-analysis.

Besides [San93], this chapter mainly relates to three papers: [HL92], [HL93] and [PS93]. In [HL92], the notion of symbolic bisimulation is introduced within a syntax-free framework, where *symbolic transition graphs* are considered; a verification algorithm is given and, for a version of CCS with value-passing, an adequacy result similar to ours is proved. For the same value-passing language, a sound and complete proof system is presented in [HL93]. In [PS93], for the same name-passing language considered here, late and early ground bisimulation and the induced congruences are equipped with four distinct algebraic proof systems; efficiency considerations are absent.

This chapter may be viewed as the extension of [HL92] and [HL93] to a name-passing setting, for which "efficient" characterizations of different bisimulation-based equivalences are obtained. A more detailed comparison with these and other papers is deferred to the conclusive Chapter 6. Here we want only to stress the main reason that makes this extension non-trivial. The blurring of values and channel names, a distinctive feature of the π-calculus, allows names to appear both in the actions, in the processes and in the constraints; this gives rise to a subtle interplay between name-scoping and boolean constraining. This interplay is best revealed in the symbolic SOS rules for one of the name-binders of the π-calculus, the *restriction* operator (y). In $(y)P$, the name y is declared to be *new*, i.e. different from any other name; therefore, when we have $P \stackrel{\phi,\alpha}{\longrightarrow} P'$ as a premise of an inference rule for $(y)P$, in the conclusion we have to discard from ϕ every assumption requiring y to be equal to other names, thus obtaining a new constraint $(y)\phi$, not containing y.

The rest of the chapter is organized as follows. In Section 4.1, after introducing the standard notions of bisimulation equivalences for the π-calculus, the symbolic transitional semantics and symbolic bisimulation are presented. In Section 4.2, some basic properties relating substitutions to boolean constraints, transitional semantics and bisimulation are established; these properties will be used in later sections. In Section 4.3, the main theorems, i.e. consitency and adequacy of symbolic bisimulation w.r.t. standard bisimulation, are proven. Section 4.4 presents the proof system and the corresponding theorems of soundness and completeness. In Sections 4.1 - 4.4 only early bisimulation is considered, while late bisimulation is treated in full detail in Section 4.5.

4.1 Symbolic Semantics

In this section, after some preliminaries, the new symbolic semantics will be introduced.

4.1.1 Preliminaries

First, we fix the syntax we shall work with in this chapter. Let the language BF of *Boolean Formulae* be defined as:

$\phi ::= true \mid [x = y] \mid \neg\phi \mid \phi \wedge \phi.$

Unless otherwise stated, we will let $\phi, \psi, \ldots$ range over BF. The language $\mathcal{P}_2$ is defined by the following grammar:

$$P ::= X \quad | \quad \mathbf{0} \quad | \quad \alpha.P \quad | \quad P+P \quad | \quad \phi P \quad | \quad (y)P \quad | \quad P|P \quad | \quad recX.P.$$

As usual, α ranges over Act and X over the set of agent variables; agent variables and rec operators are subject to the usual constraints. Note that this is an extension of the basic language considered in Section 2.1; more precisely, the match - mismatch pair is replaced here by a more general form, the *boolean guard* ϕP (indeed, as we shall see, $[x \neq y]$ corresponds to $\neg[x = y]$).

We fix now some basic (somewhat standard) notations for this chapter. We shall use $false$ for $\neg true$, $[x \neq y]$ for $\neg[x = y]$ and $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$. *Evaluation* of a boolean formula ϕ, $Ev(\phi)$, into the set $\{true, false\}$ is defined inductively as expected, once we set that $Ev(true) = true$, $Ev([x = x]) = true$ and $Ev([x = y]) = false$, for distinct names x and y. $n(\phi)$ will denote the set of names occurring in ϕ; we also set $fn(\phi) = n(\phi)$.

Given a substitution σ and $V \subseteq_{\text{fin}} \mathcal{N}$, we define:

- $V\sigma = \{x\sigma |\, x \in V\}$
- $dom(\sigma) = \{x |\, x\sigma \neq x\}$
- $range(\sigma) = dom(\sigma)\sigma$
- $n(\sigma) = dom(\sigma) \cup range(\sigma)$

In the rest of the chapter we confine ourselves to *finite* substitutions, i.e. those σ s.t. $n(\sigma)$ is finite. If t is either an action, a formula or a process term, $t\sigma$ denotes the result of applying the substitution σ to t, i.e. the expression obtained by simultaneously replacing

$$\texttt{Act}\ :\ \frac{-}{\alpha.P \xrightarrow{\alpha} P}$$

$$\texttt{Sum}\ :\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad \texttt{Par}\ :\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2}\ bn(\alpha) \cap fn(P_2) = \emptyset$$

$$\texttt{Com}\ :\ \frac{P_1 \xrightarrow{\overline{x}z} P_1',\ P_2 \xrightarrow{x(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'\{z/y\}}$$

$$\texttt{Close}\ :\ \frac{P_1 \xrightarrow{\overline{x}(y)} P_1',\ P_2 \xrightarrow{x(y)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (y)(P_1' \mid P_2')}$$

$$\texttt{Res}\ :\ \frac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'}\ y \notin n(\alpha) \qquad \texttt{Open}\ :\ \frac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(y)} P'}\ x \neq y$$

$$\texttt{Guard}\ :\ \frac{P \xrightarrow{\alpha} P'}{\phi P \xrightarrow{\alpha} P'}\ Ev(\phi) = true \qquad \texttt{Rec}\ :\ \frac{P[recX.P/X] \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

Symmetrical rules for `Sum` , `Par` , `Com` and `Close` are omitted.

Table 4.1: Standard SOS rules for $\mathcal{P}_2$.

each $x \in fn(t)$ with $x\sigma$ in t, with renaming of bound names possibly involved. We also extend fn to substitutions and let $fn(\sigma)$ to be $n(\sigma)$; in this way, function $fn(\,.\,)$ is defined over both names, actions, processes, formulae and substitutions.

We adopt for $\mathcal{P}_2$ the *late* operational semantics discussed in Chapter 2, Section 2.1. For the sake of completeness and to ease the comparison with the symbolic operational semantics we are going to introduce, the complete standard operational semantics is presented in Table 4.1.

The definition of standard (early and late) ground bisimulation equivalences are the usual ones of Chapter 2, Section 2.3. In particular, as to early bisimulation, it is convenient to refer to Definition 2.3.3. From now on, for this chapter, $\sim$ will denote *early* bisimulation equivalence. The treatment of late bisimulation equivalence is confined in Section 4.5.

$$\phi-\texttt{Act} \;:\; \frac{-}{\alpha.P \overset{true,\,\alpha}{\longrightarrow} P}$$

$$\phi-\texttt{Sum} \;:\; \frac{P_1 \overset{\phi,\,\alpha}{\longrightarrow} P_1'}{P_1+P_2 \overset{\phi,\,\alpha}{\longrightarrow} P_1'} \qquad \phi-\texttt{Par} \;:\; \frac{P_1 \overset{\phi,\,\alpha}{\longrightarrow} P_1'}{P_1 \mid P_2 \overset{\phi,\,\alpha}{\longrightarrow} P_1' \mid P_2} \; \begin{array}{l} bn(\alpha)\cap \\ fn(P_2)=\emptyset \end{array}$$

$$\phi-\texttt{Com} \;:\; \frac{P_1 \overset{\phi_1,\,\overline{x}z}{\longrightarrow} P_1',\; P_2 \overset{\phi_2,w(y)}{\longrightarrow} P_2'}{P_1 \mid P_2 \overset{\phi_1\wedge\phi_2\wedge[x=w],\,\tau}{\longrightarrow} P_1' \mid P_2'\{z/y\}}$$

$$\phi-\texttt{Close} \;:\; \frac{P_1 \overset{\phi_1,\,\overline{x}(y)}{\longrightarrow} P_1',\; P_2 \overset{\phi_2,\,w(y)}{\longrightarrow} P_2'}{P_1 \mid P_2 \overset{\phi_1\wedge\phi_2\wedge[x=w],\,\tau}{\longrightarrow} (y)(P_1' \mid P_2')}$$

$$\phi-\texttt{Res} \;:\; \frac{P \overset{\phi,\,\alpha}{\longrightarrow} P'}{(y)P \overset{(y)\phi,\,\alpha}{\longrightarrow} (y)P'} \; y\notin n(\alpha) \qquad \phi-\texttt{Open} \;:\; \frac{P \overset{\phi,\,\overline{x}y}{\longrightarrow} P'}{(y)P \overset{(y)\phi,\,\overline{x}(y)}{\longrightarrow} P'} \; x\neq y$$

$$\phi-\texttt{Guard} \;:\; \frac{P \overset{\psi,\,\alpha}{\longrightarrow} P'}{\phi P \overset{\phi\wedge\psi,\,\alpha}{\longrightarrow} P'} \qquad \phi-\texttt{Rec} \;:\; \frac{P[recX.P/X] \overset{\phi,\,\alpha}{\longrightarrow} P'}{recX.P \overset{\phi,\,\alpha}{\longrightarrow} P'}$$

Symmetrical rules for $\phi-\texttt{Sum}$, $\phi-\texttt{Par}$, $\phi-\texttt{Com}$ and $\phi-\texttt{Close}$ are omitted.

Table 4.2: Symbolic SOS rules for $\mathcal{P}_2$.

4.1.2 Symbolic semantics

To introduce the symbolic semantics, some specific notation for boolean formulae and substitutions is needed.

Definition 4.1.1 (Basic definitions)

- $\sigma \models \phi$ *stands for* $Ev(\phi\sigma) = true$;
- $\phi \models \psi$ *stands for* $\forall\ \sigma$: $\sigma \models \phi$ *implies* $\sigma \models \psi$;
- $[\alpha = \beta]$ *stands for*

$$\begin{array}{ll} [x=z] & \textit{if } \exists\, y\text{: } (\alpha = x(y) \textit{ and } \beta = z(y)) \textit{ or} \\ & (\alpha = \overline{x}(y) \textit{ and } \beta = \overline{z}(y)); \\ [x=z]\wedge[y=w] & \textit{if } \alpha = \overline{x}y \textit{ and } \beta = \overline{z}w; \\ true & \textit{if } \alpha = \beta = \tau; \\ false & \textit{otherwise.} \end{array}$$

- $\bigvee D$, *where* $D = \{\phi_1, \ldots, \phi_n\} \subseteq_{\text{fin}} BF$, $n > 0$, *is the boolean formula* $\phi_1 \vee \ldots \vee \phi_n$. *A similar notation will be used for* $\bigwedge D$. *Furthermore, we let* $\bigvee \emptyset$ *denote* $false$ *and* $\bigwedge \emptyset$ *denote* $true$.
- *The boolean formula* $(y)\phi$ *is defined by structural induction on* ϕ *as follows:*

$$\begin{array}{rlr} (y)true & = true & \\ (y)[w_1 = w_2] & = [w_1 = w_2] & \textit{if } y \notin \{w_1, w_2\} \\ (y)[y = y] & = true & \\ (y)[y = w] & = (y)[w = y] = false & \textit{if } y \neq w \\ (y)\neg\phi & = \neg(y)\phi & \\ (y)(\phi_1 \wedge \phi_2) & = (y)\phi_1 \wedge (y)\phi_2 & \end{array}$$

The symbolic transitional semantics of $\mathcal{P}_2$ is presented in Table 4.2.

Note that each symbolic rule is the counterpart of a concrete one. Intuitively, the boolean formula ϕ in $P \overset{\phi,\,\alpha}{\longrightarrow} P'$ collects the conditions over the free names of P necessary for α to take place. E.g., rule $\phi-$`Act` says that $\alpha.P$ can perform α unconditionally. In $\phi-$`Com` and $\phi-$`Close` , the condition of matching channels ($[x = w]$) is moved into the boolean condition of the resulting symbolic transition; this is necessary to infer symbolic transitions such as:

$$(x(y).\mathbf{0}) \mid (\overline{w}y.\mathbf{0}) \overset{[x=w],\,\tau}{\longrightarrow} \mathbf{0} \mid \mathbf{0}\,.$$

Rule $\phi-$`Res` reveals the interplay between name-scoping and constraining: in $(y)P$, the name y is declared to be *new*, i.e. different from any other name; thus the rule says that, given a symbolic transition $P \overset{\phi,\,\alpha}{\longrightarrow} P'$ as a premise, under the assumption that y is new, every positive assumption about the identity of y has to be discarded from ϕ in the conclusion; as a result, the new constraint $(y)\phi$ that does not contain y is obtained. An example of derivation using this rule is:

$$\phi - \mathtt{Res}\ \ : \frac{P \overset{[y=z]\vee[z=w],\,\tau}{\longrightarrow} P'}{(y)P \overset{false\vee[z=w],\,\tau}{\longrightarrow} (y)P'}$$

where z and w are different from y (note that $false \vee [z = w]$ is equivalent to $[z = w]$). A similar comment holds for $\phi - \mathtt{Open}$. The other inference rules should be self-explanatory.

We are now set to introduce symbolic bisimulation for the π-calculus. For a variant of CCS with value passing, symbolic bisimulation has already been introduced in [HL92]. The underlying intuition is that of establishing equivalence of, say, P and Q under a condition ϕ, by matching symbolic transitions of P, $P \xrightarrow{\psi,\alpha} P'$, by *sets* of symbolic transitions of Q. More precisely, the condition $\phi \wedge \psi$, is partitioned into a set D of subcases, each of which entails a matching transition for Q. Due to the treatment of restricted names in the π-calculus, here we have to take into account the case when the performed action is a bound output $\overline{x}(y)$. In this case, we also impose that name y be logically different from any known name in subsequent reasoning.

Before defining symbolic bisimulation, we introduce *ϕ-decompositions*; a similar concept (ϕ-partition) was already present in [HL92].

Definition 4.1.2 (ϕ-decomposition) *Given ϕ and a set of formulae $D = \{\phi_1, \ldots, \phi_n\}$, we say that D is a ϕ-*decomposition *iff $\phi \models \bigvee D$.*

Definition 4.1.3 (Symbolic early bisimulation)

- *A family $\mathcal{F} = \{R_\phi \,|\, \phi \in BF\}$ of symmetric binary relations over $\mathcal{P}_2$, indexed over the set BF of boolean formulae, is a* family of symbolic early bisimulations (FSEB) *iff $\forall\ \phi$ and $(P, Q) \in R_\phi$, $P \xrightarrow{\psi,\alpha} P'$, with $bn(\alpha) \cap fn(P, Q, \phi) = \emptyset$, implies:*

  there exists a χ-decomposition D, such that for all $\phi' \in D$, there is a transition $Q \xrightarrow{\psi',\beta} Q'$ with $\phi' \models (\psi' \wedge [\alpha = \beta])$ and $(P', Q') \in R_{\phi'}$, where:

  $$\chi \stackrel{def}{=} \begin{cases} \phi \wedge \psi \wedge \bigwedge_{z \in fn(P,Q,\phi)} [y \neq z] & \text{if } \alpha \text{ is of the form } \overline{x}(y) \\ \phi \wedge \psi & \text{otherwise.} \end{cases}$$

- *$P \simeq^\phi Q$ iff there exists a FSEB $\{R_\psi |\, \psi \in BF\}$ such that $(P, Q) \in R_\phi$.*

We now discuss informally the reason why symbolic bisimulation is more amenable to efficient automatic verification than the concrete one, leaving for future work a precise complexity analysis. Note that, on input actions, no multiple instantiation of the formal parameter is required; instead, a single instantiation with a *fresh* name suffices (the "freshness" condition is $bn(\alpha) \cap fn(P, Q, \phi) = \emptyset$ [1]). The case analysis on the received value is now embodied in the decomposition D; by choosing this decomposition in an appropriate way, the number of cases to deal with is in general significantly smaller than that arising with the original definition. This is shown in the following example.

[1] $fn(\psi)$ need not to be considered because, as it will turn out (Lemma 4.2.6), $n(\psi) \subseteq fn(P)$.

Example 4.1.4 Consider the processes P_1 and P_2 given in the Introduction:

$$P_1 \stackrel{def}{=} x(y).P \text{ and } P_2 \stackrel{def}{=} x(y).([y = z]P + [y \neq z]P)$$

where $y \neq z$. $P_1 \sim^c P_2$ can be established by showing $P_1 \simeq^{true} P_2$, or by relying on the original definition. Below, we make use of both methods.

$P_1 \simeq^{true} P_2$ can be established as follows. Let $P' = [y = z]P + [y \neq z]P$ and Id denote the identity relation. It is easy to see that the family of relations below :

$$\begin{array}{rcl}\mathcal{R}_{true} & = & \{(P_1, P_2), (P_2, P_1)\} \\ \mathcal{R}_{[z=y]} & = & \{(P, P'), (P', P)\} \\ \mathcal{R}_{[z\neq y]} & = & \{(P, P'), (P', P)\} \\ \mathcal{R}_{\phi} & = & Id \ \text{ for } \phi \notin \{true, [z = y], [z \neq y]\}\end{array}$$

is a FSEB, corresponding to decomposing $true$ into $\{[z = y], [z \neq y]\}$. Note that, formally, we have to exhibit infinitely many relations (one for each $\phi \in BF$); but it turns out that only finitely many are different from Id.

To establish $P_1 \sim^c P_2$ relying on the definition, we should prove that for each substitution σ, $P_1\sigma \sim P_2\sigma$; the relation to exhibit for each σ is (here we assume for the sake of simplicity that $y \notin n(\sigma)$):

$$\mathcal{R} = \{(P_1\sigma, P_2\sigma)\} \cup \{(P\sigma\{w/y\}, P'\sigma\{w/y\}) \mid w \in fn(P'\sigma, P\sigma)\} \cup Id\,.$$

Actually, the appropriate decomposition can be determined automatically for a large class of processes. In [HL92], an algorithm is presented to check symbolic bisimulation between two finite *standard* symbolic transition graphs. A symbolic graph is standard if the bound names of any transition does not occur free in its ancestor-nodes. Given two finite standard graphs G_1 and G_2, the algorithm calculates, in a time polynimial in the sizes of the graphs, the *most general boolean expression* under which G_1 and G_2 are equivalent, i.e. it calculates a boolean ϕ such that if $G_1 \simeq^{\psi} G_2$ then $\phi \models \psi$. Therefore, the equivalence problem for graphs is reduced to the implication problem for boolean expressions.

By introducing minor modifications (that take into account the extra conditions due to bound output when determining the decomposition), Hennessy and Lin' algorithm can be used to calculate the most general boolean expression of two π-calculus processes represented by finite standard symbolic transition graphs. If we consider, e.g., finite (i.e. rec-free) processes, it is sufficient to generate their symbolic graphs by considering one transition for each equivalence class of transitions and to use fresh names for input and bound output actions. We want also to point out that, as we shall see in Section 4.2, the implication problem between formulae is decidable for the considered language (BF).

4.2 Basic Properties of Substitutions

In this section, some basic facts about substitutions will be proved. The section is divided in two parts, where substitutions are related to properties of boolean formulae and of operational semantics, respectively. The most relevant results of this section are Lemma 4.2.2 and Lemma 4.2.9. The former enables us to verify the satisfaction relation $\models$ by only checking finitely many substitutions; an easy consequence of this (Corollary 4.2.3) is that $\models$ is decidable. The latter relates concrete and symbolic transitional semantics, and is crucial for proving both the consistency and the adequacy theorems.

4.2.1 Substitutions and boolean formulae

Lemma 4.2.1 states some elementary properties of substitutions and boolean formulae.

Lemma 4.2.1

1. *If t is a name, a boolean formula or a $\mathcal{P}_2$-term, $\widetilde{w}$ is a vector of pairwise distinct names and $\widetilde{w} \cap fn(t) = \emptyset$, then for any $\widetilde{z}$: $t = t\{\widetilde{w}/\widetilde{z}\}\{\widetilde{z}/\widetilde{w}\}$.*
2. *If $Ev(\phi) = true$ and σ is injective over $n(\phi)$, then also $Ev(\phi\sigma) = true$.*
3. *Let σ be a substitution such that $y \notin n(\sigma)$. Then $\sigma \models \phi$ if and only if $\sigma \models (y)\phi$.*

PROOF: By structural induction on t and ϕ. □

Next lemma is the core of the adequacy proof. It will enable us to verify the logical implication between any two formulae, $\phi \models \psi$, by testing a *finite* set S of substitutions. These are essentially the substitutions whose names range in $n(\phi, \psi)$ *plus* a reserve of 'fresh' names F. The intuitive reason this works is that, given any substitution ρ over ϕ and ψ, the names of $range(\rho)$ can be injectively renamed to names in F, to obtain a new substitution in S "equivalent" to the original ρ. The actual statement of the lemma is complicated by the presence of a distinct name y that represents the input formal parameter. In order to define injective renamings, set F has to be chosen large enough[2].

We adopt, in the sequel, the following notation: given a substitution σ, $\sigma_{|V}$ denotes the *restriction* of σ to V, i.e. the substitution defined as follows: for each $x \in \mathcal{N}$, if $x \in V$ then $x(\sigma_{|V}) = x\sigma$, $x(\sigma_{|V}) = x$ otherwise.

Lemma 4.2.2 *Let $y \in \mathcal{N}$, $V \subseteq_{\mathrm{fin}} \mathcal{N} - \{y\}$ and $V_y \stackrel{def}{=} V \cup \{y\}$; moreover suppose $n(\phi, \psi) \subseteq V_y$, where ϕ, $\psi \in BF$. Fix any $F \subseteq_{\mathrm{fin}} \mathcal{N} - V_y$ such that $|F| > 2 * |V_y|$ and consider the set of substitutions: $S \stackrel{def}{=} \{\sigma |\, n(\sigma) \subseteq V \cup F$ and $\sigma \models \phi\}$. We have $\phi \models \psi$ if:*

[2] We have used here a lower-bound of two multiplied by the number of all the other involved names; we could have considered a set F just as large as the the involved names, at the cost of complicating the proof.

a. *$y \notin n(\phi)$ implies that $\sigma\{^z/_y\} \models \psi$, for each $\sigma \in S$ and for each $z \in V_y\sigma$, and*

b. *$y \in n(\phi)$ implies that $\phi \models \bigwedge_{x \in V}[x \neq y]$ and $\sigma \models \psi$, for each $\sigma \in S$.*

PROOF: Let ρ be any substitution such that $\rho \models \phi$. We have to show that $\rho \models \psi$. Suppose $V = \{x_1, \ldots, x_n\}$, with the x_i's pairwise distinct and $n \geq 0$. For each $i \in \{1, \ldots, n\}$, define $z_i \stackrel{def}{=} x_i\rho$. Thus

$$V\rho = \{z_1, \ldots, z_n\}$$

(where the z_i's, $1 \leq i \leq n$, are not necessarily pairwise distinct). Furthermore define $z_0 \stackrel{def}{=} y\rho$.

Fix now a set of names $\widetilde{w} \subseteq F$, s.t. $|\widetilde{w}| = |V\rho|$ and $\widetilde{w} \cap V_y\rho = \emptyset$: such a $\widetilde{w}$ does exist because F has been chosen big enough. Clearly it is $y \notin \widetilde{w}$, since $y \notin F$. Suppose $\widetilde{w} = \{w_1, \ldots, w_n\}$ (where the w_i's are not necessarily pairwise distinct). Define

$$\sigma_0 = \{w_1/z_1, \ldots, w_n/z_n\}$$

in such a way that $w_i = w_j$ iff $z_i = z_j$: this is possible because $|\widetilde{w}| = |V\rho|$. Thus σ_0 is an injective substitution over $V_y\rho$. Therefore, since $Ev(\phi\rho) = true$, it is also (Lemma 4.2.1(2)):

$$Ev(\phi\rho\sigma_0) = true. \tag{4.1}$$

Now, it is easily checked that:

$$\forall x \in V_y : \; x\rho\sigma_0 = x(\rho\sigma_0)_{|V}\{^{z_0\sigma_0}/_y\}. \tag{4.2}$$

Letting $w_0 \stackrel{def}{=} z_0\sigma_0$ and $\sigma \stackrel{def}{=} (\rho\sigma_0)_{|V}$, we have now two cases:

- $z_0 \in V\rho$. Thus $\rho \not\models \bigwedge_{x \in V}[x \neq y]$. Recalling that $\rho \models \phi$, we have then that $\phi \not\models \bigwedge_{x \in V}[x \neq y]$, and this in turn implies $y \notin n(\phi)$ (condition a. of the hypotheses). Let us now check that $\sigma \in S$. Indeed, from $y \notin n(\phi)$, it follows $\phi\sigma = \phi\rho\sigma_0$, hence (from (4.1)): $Ev(\phi\sigma) = true$; furthermore we have $n(\sigma) = V \cup \widetilde{w} \subseteq V \cup F$. Thus we are in case a. of the statement of the lemma. From $z_0 \in V\rho$, it follows $w_0 \in V\sigma$. Therefore, by hypothesis, it is $\sigma\{^{w_0}/_y\} \models \psi$. But, since $n(\psi) \subseteq V_y$, from (4.2) it follows that:

  $$Ev(\psi\rho\sigma_0) = Ev(\psi\sigma\{^{w_0}/_y\}) = true.$$

  Therefore, since $\sigma_0^{-1} = \{z_1/w_1, \ldots, z_n/w_n\}$ is injective over $V_y\rho\sigma_0 = V\rho\sigma_0 = \widetilde{w}$, it is also (Lemma 4.2.1 (1) and (2)):

  $$Ev(\psi\rho) = Ev(\psi\rho\sigma_0\sigma_0^{-1}) = true.$$

  that is the thesis for this case.

- $z_0 \notin V\rho$. Hence $w_0 = z_0$ and, from (4.1) and (4.2), we have

$$Ev(\phi\sigma\{z_0/y\}) = Ev(\phi\rho\sigma_0) = true.$$

Recall now that $z_0 \notin \widetilde{w}$, thus either $z_0 = y$ or $z_0 \notin V_y\sigma = \widetilde{w} \cup \{y\}$. From these facts, relation above and Lemma 4.2.1 (1) and (2), it therefore follows:

$$Ev(\phi\sigma) = Ev(\phi\sigma\{z_0/y\}\{y/z_0\}) = true.$$

Therefore, since $n(\sigma) \subseteq V \cup F$, it is $\sigma \in S$. From the hypotheses it follows that $\sigma \models \psi$ and hence (Lemma 4.2.1(2)) also $\sigma\{z_0/y\} \models \psi$. But from (4.2) we get that $\psi\sigma\{z_0/y\} = \psi\rho\sigma_0$. Let σ_0^{-1} be defined as in the previous case and note that σ_0^{-1} is injective over $V_y\rho\sigma_0 = \widetilde{w} \cup \{z_0\}$; therefore it is also (Lemma 4.2.1 (1) and (2)):

$$Ev(\psi\rho) = Ev(\psi\rho\sigma_0\sigma_0^{-1}) = true$$

that is the thesis for this case. □

As a corollary of the above lemma, we obtain the decidability of the relation $\models$.

Corollary 4.2.3 *The relation $\models \subseteq BF \times BF$ is decidable.*

PROOF: Given any ϕ and ψ, assume we want to decide whether $\phi \models \psi$. Define $V \stackrel{def}{=} n(\phi, \psi)$, fix any $y \in \mathcal{N} - V$ and $F \subseteq_{\text{fin}} \mathcal{N} - (V \cup \{y\})$ s.t. $|F| > 2 * |V \cup \{y\}|$ and define $S \stackrel{def}{=} \{\sigma |\, n(\sigma) \subseteq V \cup F \text{ and } \sigma \models \phi\}$; S is clearly finite. We check syntactically (by applying the definition) whether for each $\sigma \in S$ $\sigma \models \psi$. If this is the case, then according to the previous lemma, part a., $\phi \models \psi$ (note that y plays no role here); otherwise $\phi \not\models \psi$. □

Next definition is useful to express the fact that a substitution is a *variant* of another one, i.e. that they are the same up to some injective renaming of names; $\rho \models \chi(\sigma, V)$ formalizes the fact that ρ is a variant of σ over the set V.

Definition 4.2.4 (Characteristic boolean formula) *Given $V \subseteq_{\text{fin}} \mathcal{N}$ and a substitution σ, we define the* characteristic boolean formula of σ over V *as:*

$$\chi(\sigma, V) \stackrel{def}{=} \bigwedge_{x,y \in V} m_{xy}$$

where $m_{xy} \stackrel{def}{=} [x = y]$ if $x\sigma = y\sigma$, $m_{xy} \stackrel{def}{=} [x \neq y]$ otherwise.

Lemma 4.2.5 below asserts that variant substitutions “behave the same” w.r.t. $\models$.

Lemma 4.2.5 *Let σ, ρ be substitutions, $V \subseteq_{\text{fin}} \mathcal{N}$ and suppose $\rho \models \chi(\sigma, V)$. If $\sigma \models \phi$ and $n(\phi) \subseteq V$ then $\rho \models \phi$.*

PROOF: Let $V = \{x_1, \ldots, x_n\}$, with the x_i's pairwise distinct, and $n \geq 0$ and let $\chi(\sigma, V) = \bigwedge_{x_i, x_j \in V} m_{x_i x_j}$. For each $i \in \{1, \ldots, n\}$ define $y_i = x_i\sigma$ and $z_i = x_i\rho$. We will show that, taken any $i, j \in \{1, \ldots, n\}$, it is $z_i = z_j$ iff $y_i = y_j$. Indeed: $z_i = z_j$ iff $x_i\rho = x_j\rho$ iff (recalling that $\rho \models \chi(\sigma, V)$) $m_{x_i x_j} = [x_i = x_j]$ iff $x_i\sigma = x_j\sigma$ iff $y_i = y_j$.

Therefore the set $\sigma_0 = \{z_1/y_1, \ldots, z_n/y_n\}$ is a substitution, and furthermore:

1. for each $x \in V$, $x\rho = x\sigma\sigma_0$
2. σ_0 is injective over $V\sigma$.

From 1 it follows that $\phi\rho = \phi\sigma\sigma_0$, and since from 2 σ_0 is injective and $Ev(\phi\sigma) = true$, it follows from Lemma 4.2.1 that also $Ev(\phi\rho) = true$. □

4.2.2 Substitutions and operational semantics

It is easy to see that Lemma 2.1.1, 2.1.2 and 2.1.4 of Chapter 2 extend to the more general language we are considering in this chapter (where the match - mismatch pair is replaced by the ϕP construct). Furthermore, they extend to symbolic transition as well. These facts are stated in the following three lemmata. All of them are easily proven by transition induction.

Lemma 4.2.6

1. *If* $P \xrightarrow{\alpha} P'$ *then:* $fn(\alpha) \subseteq fn(P)$ *and* $fn(P') \subseteq fn(P) \cup bn(\alpha)$.
2. *If* $P \xrightarrow{\phi, \alpha} P'$ *then:* $n(\phi) \subseteq fn(P)$, $fn(\alpha) \subseteq fn(P)$ *and* $fn(P') \subseteq fn(P) \cup bn(\alpha)$.

Lemma 4.2.7 *Let σ be an injective substitution over $fn(P)$ and suppose $bn(\alpha) \cap n(\sigma) = \emptyset$.*

1. *If* $P \xrightarrow{\alpha} P'$ *then equally* $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.
2. *If* $P \xrightarrow{\phi, \alpha} P'$ *then equally* $P\sigma \xrightarrow{\phi\sigma, \alpha\sigma} P'\sigma$.

Lemma 4.2.8

1. *If* $P \xrightarrow{\alpha} P'$, *with* $bn(\alpha) = \{y\}$ *and* $y_0 \notin fn(P)$, *then equally* $P \xrightarrow{\beta} P'\{y_0/y\}$, *with* β *α-equivalent to* α *and* $bn(\beta) = \{y_0\}$.
2. *If* $P \xrightarrow{\phi, \beta} P'$, *with* $bn(\alpha) = \{y\}$ *and* $y_0 \notin fn(P)$, *then equally* $P \xrightarrow{\phi, \beta} P'\{y_0/y\}$, *with* β *α- equivalent to* α *and* $bn(\beta) = \{y_0\}$.

As a consequence of the previous three lemmata, Remark 2.1.5 extends to the language we are considering, both for concrete and for symbolic transitions.

The main result of this part is Lemma 4.2.9, that relates symbolic and standard transitional semantics via substitutions. It will be crucial for the proof of both the consistency and the adequacy theorems of symbolic bisimulation.

Lemma 4.2.9 (Relating symbolic and concrete SOS)

1. *If* $P \xrightarrow{\phi,\alpha} P'$, *with* $bn(\alpha) \cap fn(P,\sigma) = \emptyset$ *and* $\sigma \models \phi$, *then* $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.
2. *If* $P\sigma \xrightarrow{\alpha} P'$, *with* $bn(\alpha) \cap fn(P,\sigma) = \emptyset$, *then there exists a symbolic transition* $P \xrightarrow{\phi,\beta} P''$, *with* $\sigma \models \phi$, $\beta\sigma = \alpha$ *and* $P''\sigma = P'$.

PROOF: The proof of each of the two statements goes by transition induction.

1. We only deal with the cases when the last applied rule is $\phi - \texttt{Res}$ or $\phi - \texttt{Open}$. The other cases are easier or can be handled similarly.

- ($\phi - \texttt{Res}$)

$$\phi - \texttt{Res} \; : \frac{P \xrightarrow{\phi,\,\alpha} P'}{(y)P \xrightarrow{(y)\phi,\,\alpha} (y)P'} \; y \notin n(\alpha) \, .$$

By hypothesis we have $\sigma \models (y)\phi$ and $bn(\alpha) \cap fn((y)P,\sigma) = \emptyset$. By Remark 2.1.5, we assume $y \notin n(\sigma)$. To apply the inductive hypothesis to the transition $P \xrightarrow{\phi,\,\alpha} P'$, we have to check that $bn(\alpha) \cap fn(P,\sigma) = \emptyset$ and that $\sigma \models \phi$. By hypothesis, $bn(\alpha) \cap fn((y)P,\sigma) = \emptyset$; since $fn(P,\sigma) \subseteq fn((y)P,\sigma,y)$ and $y \notin n(\alpha)$, it is also $bn(\alpha) \cap fn(P,\sigma) = \emptyset$. We now check that $\sigma \models \phi$. Indeed, from $\sigma \models (y)\phi$, $y \notin n(\sigma)$ and Lemma 4.2.1 (3), it follows that $\sigma \models \phi$. Thus, the inductive hypothesis gives us:

$$P\sigma \xrightarrow{\alpha\sigma} P'\sigma \, .$$

Since $y \notin n(\alpha,\sigma)$, it is also $y \notin n(\alpha\sigma)$. We can therefore apply $\texttt{Res}$ with y to the above transition:

$$(y)P\sigma \xrightarrow{\alpha\sigma} (y)P'\sigma.$$

Since $((y)P)\sigma = (y)P\sigma$ and $((y)P')\sigma = (y)P'\sigma$, we have the thesis.

- ($\phi - \texttt{Open}$)

$$\phi - \texttt{Open} \; : \frac{P \xrightarrow{\phi,\overline{x}y} P'}{(y)P \xrightarrow{(y)\phi,\overline{x}(y)} P'} \; y \neq x \, .$$

By hypothesis $\sigma \models (y)\phi$ and $y \notin n(\sigma)$. Thus, it is $((y)P)\sigma = (y)P\sigma$; furthermore, from Lemma 4.2.1 (3), we have then that $\sigma \models \phi$. Since $bn(\overline{x}y) = \emptyset$, we can apply the inductive hypothesis to get:

$$P\sigma \xrightarrow{\overline{x\sigma}y} P'\sigma.$$

Since $x \neq y$ and $y \notin n(\sigma)$, it is $x\sigma \neq y$. We can therefore apply the `Open` rule with y to obtain:

$$(y)P\sigma \xrightarrow{\overline{x\sigma}(y)} P'\sigma.$$

Since $((y)P)\sigma = (y)P\sigma$, we have the thesis for this case.

2. Again, the only non-trivial cases are those concerning the restriction operator. We show only the `Res` case, since the `Open` can be handled in a similar way.

- (`Res`) Consider the term $((y)P)\sigma$; we assume by Remark 2.1.5 that $y \notin n(\sigma)$, so that $((y)P)\sigma = (y)P\sigma$. Therefore we have:

$$\texttt{Res} \;:\; \frac{P\sigma \xrightarrow{\alpha} P'}{(y)P\sigma \xrightarrow{\alpha} (y)P'\sigma}\; y \notin n(\alpha)$$

where, by hypothesis, $bn(\alpha) \cap fn((y)P, \sigma) = \emptyset$. Since $fn(P, \sigma) \subseteq fn((y)P, \sigma, y)$ and $y \notin n(\alpha)$, it is $bn(\alpha) \cap fn(P, \sigma) = \emptyset$. We can therefore apply the inductive hypothesis, obtaining that for some ϕ:

$$P \xrightarrow{\phi,\beta} P''$$

with $\sigma \models \phi$, $\beta\sigma = \alpha$ and $P''\sigma = P'$. Now, $y \notin n(\alpha, \sigma)$ implies $y \notin n(\beta)$. Therefore we can apply the rule $\phi - \texttt{Res}$ to the above transition obtaining:

$$(y)P \xrightarrow{(y)\phi,\beta} (y)P''.$$

Since $y \notin n(\sigma)$ and $\sigma \models \phi$, from Lemma 4.2.1 (3) we get $\sigma \models (y)\phi$; furthermore $((y)P'')\sigma = (y)P''\sigma = (y)P'$; this is the thesis for present case.

□

We come now to two lemmata about bisimulation. The following one says that ground bisimulation $\sim$ is closed under injective substitutions.

Lemma 4.2.10 *If $P \sim Q$ and σ is injective over $fn(P, Q)$, then also $P\sigma \sim Q\sigma$.*

Proof: Routine, exploiting Lemma 4.2.7 (See also [MPW92]). □

Next lemma asserts that variant substitutions "behave the same" w.r.t. $\sim$.

Lemma 4.2.11 *Let σ, ρ be substitutions, $V \subseteq_{\text{fin}} \mathcal{N}$ and suppose $\rho \models \chi(\sigma, V)$. If $P\sigma \sim Q\sigma$ and $fn(P, Q) \subseteq V$ then $P\rho \sim Q\rho$.*

PROOF: Let $V = \{x_1, \ldots, x_n\}$, with the x_i's pairwise distinct, and $n \geq 0$ and let $\chi(\sigma, V) = \bigwedge_{x_i, x_j \in V} m_{x_i x_j}$. For each $i \in \{1, \ldots, n\}$ define $y_i = x_i\sigma$ and $z_i = x_i\rho$. Proceeding exactly like in the proof of Lemma 4.2.5, we show that the set $\sigma_0 = \{z_1/y_1, \ldots, z_n/y_n\}$ is a substitution, and furthermore:

1. for each $x \in V$, $x\rho = x\sigma\sigma_0$,
2. σ_0 is injective over $V\sigma$.

From 1 it follows that $P\rho = P\sigma\sigma_0$ and $Q\rho = Q\sigma\sigma_0$. Furthermore, from $P\sigma \sim Q\sigma$, from 2 above and from Lemma 4.2.10, we get that $P\sigma\sigma_0 \sim Q\sigma\sigma_0$, from which our claim follows. $\square$

4.3 Consistency and Adequacy of Symbolic Bisimulation

For stating both the consistency and the adequacy theorems, it is useful to fix the following definition:

Definition 4.3.1 (Closing $\sim$ under ϕ) *For each $\phi \in BF$, let relation $\sim^\phi$ be:*

$$P \sim^\phi Q \text{ iff } \forall\sigma \text{ such that } \sigma \models \phi,\ P\sigma \sim Q\sigma.$$

Theorem 4.3.2 (Consistency of symbolic bisimulation) *$P \simeq^\phi Q$ implies $P \sim^\phi Q$.*

PROOF: We will show that the relation:

$$\mathcal{R} \stackrel{def}{=} \{(P\sigma, Q\sigma) \,|\, \exists\, \phi.\, \sigma \models \phi \;\text{ and }\; P \simeq^\phi Q\}$$

is a ground bisimulation. Suppose $P \simeq^\phi Q$ and $\sigma \models \phi$. Let us see how the moves of $P\sigma$ are matched by $Q\sigma$. Suppose that $P\sigma \xrightarrow{\alpha} P'$, with $bn(\alpha) \cap fn(P\sigma, Q\sigma) = \emptyset$; by Remark 2.1.5, we assume that $bn(\alpha) \cap fn(P, Q, \phi, \sigma) = \emptyset$, as well. We distinguish the possible cases for α (α silent, input, free output or bound output action) and confine ourselves to the input and the bound output cases, since the others are easier.

- $\alpha = x(y)$. It suffices to show that for each $z \in fn(P\sigma, Q\sigma, y)$ there exists Q' such that $Q\sigma \xrightarrow{\alpha} Q'$ and $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$. Thus fix $z \in fn(P\sigma, Q\sigma, y)$. From $P\sigma \xrightarrow{\alpha} P'$ and Lemma 4.2.9 we deduce that there is a symbolic transition:

  $$P \stackrel{\psi, w(y)}{\longrightarrow} P''$$

  where $\sigma \models \psi$, $w\sigma = x$ and $P''\sigma = P'$. Since $P \simeq^\phi Q$, there exists a $\phi \wedge \psi$-decomposition $D = \{\phi_1, \ldots, \phi_n\}$ s.t. for each $i \in \{1, \ldots, n\}$ there exists a symbolic transition:

  $$Q \stackrel{\psi_i, w_i(y)}{\longrightarrow} Q_i$$

where $\phi_i \models \psi_i \wedge [w = w_i]$ and $P'' \simeq^{\phi_i} Q_i$. Now, $\sigma \models \phi \wedge \psi$. But, since $n(\phi \wedge \psi) \subseteq fn(P, Q, \phi)$ (Lemma 4.2.6) and $y \notin fn(P, Q, \phi, \sigma)$, we have that $(\phi \wedge \psi)\sigma\{z/y\} = (\phi \wedge \psi)\sigma$, hence $\sigma\{z/y\} \models (\phi \wedge \psi)$. Therefore, it is $\sigma\{z/y\} \models \phi_j$, for some $j \in \{1, \ldots, n\}$. Applying again Lemma 4.2.9 to transition $Q \stackrel{\psi_j, w_j(y)}{\longrightarrow} Q_j$, we get the matching transition for $Q\sigma$:

$$Q\sigma \stackrel{x(y)}{\longrightarrow} Q_j\sigma.$$

Indeed, define $Q' \stackrel{def}{=} Q_j\sigma$. Thus we have $P'\{z/y\} = P''\sigma\{z/y\}$ and that $Q'\{z/y\} = Q_j\sigma\{z/y\}$. Finally, recall that $\sigma\{z/y\} \models \phi_j$ and $P'' \simeq^{\phi_j} Q_j$; from these facts, by definition, it follows $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$.

- $\alpha = \overline{x}(y)$. We will show that there exists Q' s.t. $Q\sigma \stackrel{\alpha}{\longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$. From $P\sigma \stackrel{\alpha}{\longrightarrow} P'$ and Lemma 4.2.9, we deduce that there exists a symbolic transition:

$$P \stackrel{\psi, \overline{w}(y)}{\longrightarrow} P_1$$

where $\sigma \models \psi$, $w\sigma = x$ and $P_1\sigma = P'$. Since $P \simeq^{\phi} Q$, there exists a $\phi \wedge \psi \wedge \bigwedge_{z \in fn(P,Q,\phi)}[y \neq z]$ - decomposition $D = \{\phi_1, \ldots, \phi_n\}$ s.t. for each $i \in \{1, \ldots, n\}$ there exists a symbolic transition:

$$Q \stackrel{\psi_i, \overline{w_i}(y)}{\longrightarrow} Q_i$$

with $\phi_i \models \psi_i \wedge [w = w_i]$ and $P_1 \simeq^{\phi_i} Q_i$. Now, since $y \notin fn(P, Q, \phi, \sigma)$, it is $\sigma \models \bigwedge_{z \in fn(P,Q,\phi)}[y \neq z]$; hence, since $\sigma \models \phi \wedge \psi$, also $\sigma \models \phi \wedge \psi \wedge \bigwedge_{z \in fn(P,Q,\phi)}[y \neq z]$. Therefore there exists $\phi_j \in D$ s.t. $\sigma \models \phi_j$. Applying Lemma 4.2.9 to the transition $Q \stackrel{\psi_j, \overline{w_j}(y)}{\longrightarrow} Q_j$, we get the matching transition for $Q\sigma$:

$$Q\sigma \stackrel{\overline{x}(y)}{\longrightarrow} Q_j\sigma.$$

Indeed, define $Q' \stackrel{def}{=} Q_j\sigma$; recall that $P' = P_1\sigma$, that $P_1 \simeq^{\phi_j} Q_j$ and that $\sigma \models \phi_j$; thus by definition $(P', Q') \in \mathcal{R}$ and we are done.

□

We now come to prove that symbolic bisimulation is adequate for expressing concrete bisimulations, i.e. that to prove that whenever $P \sim^{\phi} Q$ then $P \simeq^{\phi} Q$. To prove this adequacy theorem, we shall rely on the fact that only a suitable finite set of name-substitutions is "relevant" when working with fixed collection of processes and formulae. All remaining substitutions are, in fact, variants of the considered ones, i.e. they can be obtained by injective renaming. This is a distinctive property of the π-calculus, since it relies on the blurring of names and variables.

An informal account of the proof might be the following. We show that $\{\sim^\phi \mid \phi \in BF\}$ is a family of symbolic bisimulations. Thus, given P, Q with $P \sim^\phi Q$ and $P \overset{\psi,\alpha}{\longrightarrow} P'$, we have to find a decomposition of $\phi \wedge \psi$ (here for the sake of simplicity we assume that α is not a bound output), such that each subcase entails a matching symbolic transition for Q. Letting $V = fn(P, Q, \phi)$, the idea is to determine a *finite* set of substitutions, $S = \{\sigma_1, \dots, \sigma_k\}$, such that each substitution satisfying $\phi \wedge \psi$ is a variant over $V \cup bn(\alpha)$ of some σ_i. By relying on Lemma 4.2.2, $\phi \wedge \psi$ is decomposed into a set $D = \{\phi_1, \dots, \phi_k\}$, where we have one subcase for each $\sigma_i \in S$. Each $\phi' \in D$ is chosen to entail a symbolic matching transition for Q.

Theorem 4.3.3 (Adequacy of symbolic bisimulation) *$P \sim^\phi Q$ implies $P \simeq^\phi Q$.*

PROOF: We will show that the the family of relations

$$\mathcal{F} \overset{def}{=} \{\sim^\phi \mid \phi \in BF\}$$

is a family of symbolic (early) bisimulations. Thus let $\phi \in BF$ and $P \sim^\phi Q$ and suppose $P \overset{\psi,\alpha}{\longrightarrow} P'$ with $bn(\alpha) \cap fn(P, Q, \phi) = \emptyset$. If $\alpha = \overline{x}(y)$, define $\chi \overset{def}{=} \bigwedge_{z \in fn(P,Q,\phi)} [y \neq z]$, otherwise define $\chi \overset{def}{=} true$. We have to find a $\phi \wedge \psi \wedge \chi$-decomposition $D = \{\phi_1, \dots, \phi_n\}$ s.t. for each $i \in \{1, \dots, n\}$ there exists a symbolic transition $Q \overset{\psi_i,\beta_i}{\longrightarrow} Q_i$ with $\phi_i \models \psi_i \wedge [\alpha = \beta_i]$ and furthermore $P' \sim^{\phi_i} Q_i$.

Let $V \overset{def}{=} fn(P, Q, \phi)$ and $V_y \overset{def}{=} V \cup bn(\alpha)$. Take any $F \subseteq_{\text{fin}} \mathcal{N} - V_y$ such that $|F| > 2 * |V_y|$. Define the set of substitutions:

$$S \overset{def}{=} \{\sigma \mid n(\sigma) \subseteq F \cup V \text{ and } \sigma \models \phi \wedge \psi \wedge \chi\}\,.$$

S is clearly finite. We now distinguish the possible alternatives for α (α equal to τ, $x(y)$, $\overline{x}y$ or $\overline{x}(y)$) and stick ourselves to $\alpha = x(y)$ and $\alpha = \overline{x}(y)$, since the others are easier.

- $\alpha = x(y)$. We let $D \overset{def}{=} \{\chi(\sigma\{z/y\}, V_y) \mid \sigma \in S,\ z \in V_y\sigma\}$. The fact that this is a $\phi \wedge \psi \wedge \chi$-decomposition follows from Lemma 4.2.2, case a. (read the "ϕ" of the assertion of the lemma as $\phi \wedge \psi \wedge \chi$ and the "ψ" as $\bigvee D$). We now come to show that each $\chi(\sigma\{z/y\}, V_y) \in D$ entails a matching symbolic transition for Q. Thus fix $\sigma \in S$ and $z \in V_y\sigma$ and consider the following chain of implications:

$$\begin{array}{rl}
P \overset{\psi,\alpha}{\longrightarrow} P' & \text{implies, since } \sigma \models \psi \text{ and } y \notin n(\sigma) \\
 & \text{and in virtue of Lemma 4.2.9,} \\
P\sigma \overset{\gamma}{\longrightarrow} P'\sigma & \text{with } \alpha\sigma = \gamma\text{; from } P \sim^\phi Q \text{ and } \sigma \models \phi\text{, we have} \\
Q\sigma \overset{\gamma}{\longrightarrow} Q' & \text{for some } Q' \text{ with } P'\sigma\{z/y\} \sim Q'\{z/y\}. \\
 & \text{Due to Lemma 4.2.9, there exists a symbolic transition} \\
Q \overset{\psi',\beta}{\longrightarrow} Q'' & \text{with } \sigma \models \psi' \wedge [\beta = \alpha] \text{ and } Q''\sigma = Q'\,.
\end{array}$$

We check now that that $\chi(\sigma\{z/y\}, V_y) \models \psi' \wedge [\beta = \alpha]$ and that $P' \sim^{\chi(\sigma\{z/y\}, V_y)} Q''$.

We prove the former fact. It is $n(\psi' \wedge [\beta = \alpha]) \subseteq fn(P, Q) \subseteq V$ (Lemma 4.2.6) and $y \notin n(\sigma)$; furthermore $\sigma \models \psi' \wedge [\beta = \alpha]$; these facts imply $\sigma\{z/y\} \models \psi' \wedge [\beta = \alpha]$; hence, from Lemma 4.2.5, given any ρ with $\rho \models \chi(\sigma\{z/y\}, V_y)$, we have $\rho \models \psi' \wedge [\beta = \alpha]$.

We show now that $P' \sim^{\chi(\sigma\{z/y\}, V_y)} Q''$. Consider any substitution ρ s.t. $\rho \models \chi(\sigma\{z/y\}, V_y)$; we have to show that $P'\rho \sim Q''\rho$. From above, we know that $P'\sigma\{z/y\} \sim Q''\sigma\{z/y\}$. But $fn(P', Q'') \subseteq V_y$ (Lemma 4.2.6), thus from Lemma 4.2.11, $P'\rho \sim Q''\rho$.

- $\alpha = \overline{x}(y)$. Define $D \stackrel{def}{=} \{\chi(\sigma, V_y) | \sigma \in S\}$. The fact that this D is a $\phi \wedge \psi \wedge \chi$-decomposition follows from Lemma 4.2.2, case b. (read the "ϕ" of the assertion of the lemma as $\phi \wedge \psi \wedge \chi$ and the "ψ" as $\bigvee D$). We now come to show that each $\chi(\sigma, V_y) \in D$ entails a matching symbolic transition for Q. Thus fix $\sigma \in S$ and consider the following chain of implications:

$$
\begin{array}{rl}
P \xrightarrow{\psi, \alpha} P' & \text{implies, in virtue of Lemma 4.2.9 and } \sigma \models \psi, \\
P\sigma \xrightarrow{\gamma} P'\sigma & \text{with } \alpha\sigma = \gamma. \\
& \text{From } P \sim^{\phi} Q \text{ and } \sigma \models \phi, \text{ for some } Q' \text{ it is} \\
Q\sigma \xrightarrow{\gamma} Q' & \text{with } P'\sigma \sim Q'. \\
& \text{From Lemma 4.2.9, there exist } \psi', \beta, Q'' \text{ s.t.:} \\
Q \xrightarrow{\psi', \beta} Q'' & \text{with } \sigma \models \psi' \wedge [\beta = \alpha] \text{ and } Q''\sigma = Q'.
\end{array}
$$

We now show that $\chi(\sigma, V_y) \models \psi' \wedge [\beta = \alpha]$ and that $P' \sim^{\chi(\sigma, V_y)} Q''$. Consider any ρ s.t. $\rho \models \chi(\sigma, V_y)$; since $\sigma \models \psi' \wedge [\beta = \alpha]$ and $n(\psi' \wedge [\beta = \alpha]) \subseteq V$ (Lemma 4.2.6), from Lemma 4.2.5 we have $\rho \models \psi' \wedge [\beta = \alpha]$ as well.

We show now that $P' \sim^{\chi(\sigma, V_y)} Q''$. Consider any substitution ρ s.t. $\rho \models \chi(\sigma, V_y)$; we have to show that $P'\rho \sim Q''\rho$. From above, we know that $P'\sigma \sim Q''\sigma$. But $fn(P', Q'') \subseteq V_y$ (Lemma 4.2.6) and by hypothesis $\rho \models \chi(\sigma, V_y)$, thus from Lemma 4.2.11, $P'\rho \sim Q''\rho$.

□

We end the section by showing that also ground bisimulation $\sim$ can be characterized in terms of the symbolic one.

Theorem 4.3.4 *$P \sim Q$ iff $P \simeq^{\phi} Q$, where* $\phi = \bigwedge_{x,y \in fn(P,Q),\, x,y \text{ distinct}} [x \neq y]$.

Proof: The theorem follows immediately from the consistency and adequacy theorems for symbolic bisimulation and from the fact that $\sim$ is closed under injective substitutions (Lemma 4.2.10). □

4.4 The Proof System

Let us now consider the *finite* fragment of $\mathcal{P}_2$, i.e. the calculus without the $recX$. operator and discuss an equational axiomatization of symbolic bisimulation over it. It is well known that decidable axiomatizations cannot exist for the full language.

The statements derivable within the proof system are guarded equations of the form "$\phi \rhd P = Q$", to be read as "under ϕ, P equals Q". In the sequel, we will write "$\phi \rhd P = Q$" to mean that the equation $\phi \rhd P = Q$ is derivable within the proof system. Furthermore, we will abbreviate "$true \rhd P = Q$" simply as "$P = Q$". In this section, symbol $\equiv$ will be used for identity (up to α-conversion), in order to distinguish it from the proof-theoretic equality ($=$).

The inference rules of the system, that we refer to as $\mathcal{S}$ are presented in Table 4.3 (the standard inference rules for reflexivity, symmetry and transitivity have been omitted), while the axioms are in Table 4.4. Note that the new relevant axioms are $Subst$, $Res2$ and Exp, while the laws for Summation and Restriction are the usual ones, that we have borrowed from [MPW92].

Our proof system can be viewed as the result of merging that of [HL93] and [PS93]. More precisely, all of the inference rules, but the Res rule, are taken from [HL93], while the axioms, but $Res2$, are taken from [PS93].

The Cut rule permits case analysis on ϕ: it says that if ϕ can be split into two subcases ϕ_1 and ϕ_2, and for each subcase we can prove $P = Q$, then $P = Q$ is derivable under ϕ. The Res and $Res2$ rules exhibit the same kind of logical "hiding" of the bound name y as the rules $\phi - \mathtt{Res}$ and $\phi - \mathtt{Open}$ of symbolic transitional semantics. The other rules and axioms should be self-explanatory; anyway, we refer the reader to [PS93, HL93] for explanations on their intuitive meaning.

We give below some derived inference rules and axioms that will be useful to manipulate terms.

Lemma 4.4.1 (Derived laws)

1. *(Guard2)* $\phi \models \psi$ *implies* $\phi \rhd \psi P = P$.
2. *(Guard3)* $\phi \models \neg\psi$ *implies* $\phi \rhd \psi P = \mathbf{0}$.
3. *(Guard4)* $\phi(\psi P) = (\phi \wedge \psi)P$.
4. *(Guard5)* $\phi \models \psi$ *and* $\psi \models \phi$ *imply* $\phi P = \psi P$.
5. *(Guard6)* $\phi \models false$ *implies* $\phi P = \mathbf{0}$.
6. *(Subst2)* $\phi \models [x = y]$ *implies* $\phi \rhd \alpha.P = \alpha\{y/x\}.P$.
7. *(Cons)* $\psi \models \phi$ *and* $\phi \rhd P = Q$ *imply* $\psi \rhd P = Q$.
8. *(Sum1)* $\phi(P + Q) = \phi P + \phi Q$.

$$(Congr)\quad \frac{\phi \rhd P = Q}{\phi \rhd P' = Q'}\quad \text{where } P' = Q' \text{ stands for either of: } \tau.P = \tau.Q,\ \overline{x}y.P = \overline{x}y.Q,\ \psi P = \psi Q,\ P + R = Q + R,\ P|R = Q|R.$$

$$(Res)\quad \frac{\phi \rhd P = Q}{(y)\phi \rhd (y)P = (y)Q}$$

$$(Inp)\quad \frac{\phi \rhd \sum_{i\in I} \tau.P_i = \sum_{i\in I} \tau.Q_i}{\phi \rhd \sum_{i\in I} x(y).P_i = \sum_{i\in I} x(y).Q_i}\quad y \notin n(\phi)$$

$$(Guard)\quad \frac{\phi \wedge \psi \rhd P = Q,\ \phi \wedge \neg\psi \rhd Q = \mathbf{0}}{\phi \rhd \psi P = Q}$$

$$(False)\quad \frac{-}{false \rhd P = Q}$$

$$(Cut)\quad \frac{\phi_1 \rhd P = Q,\ \phi_2 \rhd P = Q}{\phi \rhd P = Q}\quad \phi \models \phi_1 \vee \phi_2$$

$$(Axiom)\quad \frac{-}{true \rhd P = Q}\quad \text{for each axiom } P = Q$$

Table 4.3: Inference rules of the proof system $\mathcal{S}$.

Summation Laws
$(S0)$ $P + \mathbf{0} = P$
$(S1)$ $P + P = P$
$(S2)$ $P + Q = Q + P$
$(S3)$ $P + (Q + R) = (P + Q) + R$

Restriction Laws
$(R0)$ $(x)P = P$, if $x \notin fn(P)$
$(R1)$ $(x)(y)P = (y)(x)P$
$(R2)$ $(x)(P + Q) = (x)P + (x)Q$
$(R3)$ $(x)\alpha.P = \alpha.(x)P$, if $x \notin n(\alpha)$
$(R4)$ $(x)\alpha.P = \mathbf{0}$, if $x = subj(\alpha)$

Axioms from [MPW92]

$(Subst)$ $[x = y]\alpha.P = [x = y]\alpha\{x/y\}.P$

$(Res2)$ $(y)(\phi P) = ((y)\phi)(y)P$

(Exp)
If $P \equiv \sum_{i \in I} \phi_i \alpha_i.P_i$ and $Q \equiv \sum_{j \in J} \psi_i \beta_i.Q_i$ and no α_i (resp. β_j) binds a name free in Q (resp. P):

$$\begin{aligned} P \mid Q = {} & \textstyle\sum_{i \in I} \phi_i \alpha_i.(P_i \mid Q) + \sum_{j \in J} \psi_j \beta_j.(P \mid Q_j) + \\ & + \textstyle\sum_{\alpha_i \, opp \, \beta_j} (\phi_i \wedge \psi_j \wedge [x_i = y_j])\tau.R_{ij} \end{aligned}$$

where $\alpha_i \, opp \, \beta_j$ and R_{ij} are defined by:
1. $\alpha_i = \overline{x_i}z$ and $\beta_j = y_j(y)$; then $R_{ij} = P_i \mid Q_j\{z/y\}$
2. $\alpha_i = \overline{x_i}(y)$ and $\beta_j = y_j(y)$; then $R_{ij} = (y)(P_i \mid Q_j)$
3. The "converse" of 1.
4. The "converse" of 2.

New Axioms

Table 4.4: Axioms of the proof system $\mathcal{S}$.

9. *(Sum2)* $\phi P + \psi P = (\phi \vee \psi)P$.

PROOF: Routine (see also [HL93]). □

Before proving its soundness and completeness, we give an elementary application of the proof system.

Example 4.4.2 We show that $P_1 \stackrel{def}{=} x(y).P = x(y).([y = z]P + [y \neq z]P) \stackrel{def}{=} P_2$:

$$\begin{array}{rcll} P & = & true\, P & (Guard2) \\ & = & ([y = z] \vee [y \neq z])P & (Guard5) \\ & = & [y = z]P + [y \neq z]P & (Sum2). \end{array}$$

From the latter equation, applying $Congr - \tau$, we get $\tau.P = \tau.([y = z]P + [y \neq z]P)$; finally applying Inp the wanted result follows.

Soundness is straightforward to prove by exploiting consistency and adequacy of symbolic bisimulation.

Theorem 4.4.3 (Soundness of the proof system) *$\phi \triangleright P = Q$ implies $P \simeq^{\phi} Q$.*

PROOF: Relying on the definition of $\sim^{\phi}$, check soundness of the inference rules. All the cases are straightforward. As an example we consider *Res*. Suppose $P \sim^{\phi} Q$; we have to show that $(y)P \sim^{(y)\phi} (y)Q$ as well. Let $\sigma \models (y)\phi$; we show that $((y)P)\sigma \sim ((y)Q)\sigma$.

By Remark 2.1.5, we assume that $y \notin n(\sigma)$; from Lemma 4.2.6 we get that $\sigma \models \phi$ as well. From $P \sim^{\phi} Q$ it follows that $P\sigma \sim Q\sigma$; hence, from the fact that restriction preserves $\sim$, we have:

$$((y)P)\sigma = (y)(P\sigma) \sim (y)(Q\sigma) = ((y)P)\sigma$$

that is the wanted claim. □

The actual proof of completeness relies on a "customized" notion of head normal form.

Definition 4.4.4 (Head normal forms) *A process P is in* head normal form (HNF) *if it is of the form $\sum_{i \in I} \phi_i S_i$, where:*

- *$\{\phi_i | \, i \in I\}$ is a true-decomposition such that $\phi_i \wedge \phi_j \models false$ for each $i, j \in I$ with $i \neq j$;*
- *each S_i, $i \in I$, is of the form $\sum_{j \in J_i} \alpha_j.P_j$.*

Lemma 4.4.5 *For each process P, there exists a HNF H s.t. $P = H$.*

PROOF: The proof is by structural induction on P. The most relevant case is when $P = P_1 + P_2$. We check this case only. By inductive hypothesis, there exist HNF's $H_1 \equiv \sum_{i\in I} \phi_i S_i$ and $H_2 \equiv \sum_{j\in J} \psi_j T_i$ that are provably equivalent to, respectively, P_1 and P_2; hence $P = H_1 + H_2$. Now, for each $I' \subseteq I$ and $J' \subseteq J$, define the formula

$$\chi_{I'J'} \stackrel{def}{=} \bigwedge_{i\in I', j\in J'} (\phi_i \wedge \psi_j) \wedge \bigwedge_{i\in I-I', j\in J-J'} (\neg\phi_i \wedge \neg\psi_j)\,.$$

It is easy to see that the set

$$D = \{\chi_{I'J'} \,|\, I' \subseteq I,\ J' \subseteq J\}$$

is a *true*-decomposition and furthermore that for any two distinct $\chi_1, \chi_2 \in D$, $\chi_1 \wedge \chi_2 = false$. Now we have

$$\begin{aligned} H_1 + H_2 &= true\,(H_1 + H_2) && (Guard2)\\ &= (\textstyle\bigvee D)(H_1 + H_2) && (Guard5)\\ &= \textstyle\sum_{\chi_{I'J'}\in D}(\chi_{I'J'}(H_1 + H_2)) && (Sum2)\,. \end{aligned} \tag{4.3}$$

Now observe that, from $Guard4$, $Guard5$ and $Guard6$, it follows that $\chi_{I'J'}\phi_i S_i = \chi_{I'J'} S_i$ if $i \in I'$, $\chi_{I'J'}\phi_i S_i = \mathbf{0}$ otherwise. From this, it follows that

$$\begin{aligned} \chi_{I'J'} H_1 &= \textstyle\sum_{i\in I} \chi_{I'J'}\phi_i S_i && (Sum1)\\ &= \textstyle\sum_{i\in I'} \chi_{I'J'} S_i && \text{(from the above fact and Summation laws)}\\ &= \chi_{I'J'}(\textstyle\sum_{i\in I'} S_i) && (Sum2)\,. \end{aligned}$$

In a similar manner, one shows that $\chi_{I'J'} H_2 = \chi_{I'J'}(\sum_{j\in J'} T_j)$. Hence, applying $Sum1$, we have:

$$\begin{aligned} \chi_{I'J'}(H_1 + H_2) &= \chi_{I'J'}(\textstyle\sum_{i\in I'} S_i) + \chi_{I'J'}(\textstyle\sum_{j\in J'} T_j)\\ &= \chi_{I'J'} U_{I'J'} \end{aligned}$$

where $U_{I'J'}$ is defined as $\sum_{i\in I'} S_i + \sum_{j\in J'} T_j$. From (4.3) it follows $H_1 + H_2 = \sum_{\chi_{I'J'}\in D} \chi_{I'J'} U_{I'J'}$; the latter process is a HNF. $\square$

We need some other notions about substitutions. Next definition introduces the concepts of completeness for formulae and of equivalence relation induced by a formula. The former can be explained by saying that if ϕ is complete over V then under ϕ names in V can be, in a sense, treated as *constants*. The definition of the latter is self-explanatory.

Definition 4.4.6 *Let* $V \subseteq_{\mathrm{fin}} \mathcal{N}$ *and* $\phi \in BF$.

1. *We say that* ϕ *is* complete over V *if for each* $x, y \in V$, *either* $\phi \models [x = y]$ *or* $\phi \models [x \neq y]$.

2. *We define* $\mathcal{R}(\phi, V)$, *the* equivalence relation induced by ϕ over V, *as: for each* $x, y \in V$ $(x, y) \in \mathcal{R}(\phi, V)$ *iff* $\phi \models [x = y]$.

In the sequel we adopt the following notation: given a term P of the form $\sum_{i\in I}\alpha_i.P_i$ and any set $V \subseteq_{\text{fin}} \mathcal{N}$, P_V is $\sum_{i|i\in I,\, fn(\alpha_i)\subseteq V}\alpha_i.P_i$.

Lemma 4.4.7 *Suppose that $P \equiv \sum_{i\in I}\alpha_i.P_i \simeq^\phi \sum_{j\in J}\beta_j.Q_j \equiv Q$ and that ϕ is complete over $V = \cup_{i\in I} fn(\alpha_i) \cup \cup_{j\in J} fn(\beta_j)$. Then, for any equivalence class C of $\mathcal{R}(\phi, V)$, we have $P_C \simeq^\phi Q_C$.*

PROOF: As a direct consequence of the fact that ϕ is complete over V, we have that for each $x, y \in V$:

$$(x, y) \notin \mathcal{R}(\phi, V) \text{ implies } \phi \models [x \neq y]. \tag{4.4}$$

Fix an equivalence class C of $\mathcal{R}(\phi, V)$ and any σ s.t. $\sigma \models \phi$; it is sufficient to show that $P_C\sigma \sim Q_C\sigma$. Indeed, since $P\sigma \sim Q\sigma$, for any summand $(\alpha_i.P_i)\sigma$ of $P_C\sigma$, there must exist a "matching" summand $(\beta_j.Q_j)\sigma$ of $Q\sigma$; but it must be that $fn(\beta_j) \subseteq C$, because of (4.4), i.e $(\beta_j.Q_j)$ is a summand of Q_C. By symmetry we conclude that $P_C\sigma \sim Q_C\sigma$. □

We now define a measure that will be used as induction parameter in the proof of the theorem.

Definition 4.4.8 (ϕ-depth) *Given P and ϕ, we define the* ϕ-depth of P *as:*

$$\begin{aligned} depth(\phi, P) = max\{k| \quad & \textit{for some } \sigma \models \phi \textit{ and } \alpha_1, \cdots, \alpha_n \textit{ different from } \tau, \\ & (P|\alpha_1.\cdots.\alpha_n.\mathbf{0})\sigma = R_0 \xrightarrow{\tau} R_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} R_k \ \}. \end{aligned}$$

This parameter has been defined to satisfy the three properties stated in the following lemma:

Lemma 4.4.9

1. *$depth(\phi, P) < depth(\phi, \alpha.P)$, for any action α;*
2. *$P \sim^\phi Q$ implies $depth(\phi, P) = depth(\phi, Q)$;*
3. *$\phi \models \psi$ implies $depth(\phi, P) \leq depth(\psi, P)$.*

PROOF: Straightforward from the definition. □

We are now set for proving the main theorem of the section. First, we give an informal account of the proof; it goes by induction on $depth(\phi, P)$. If $P \simeq^\phi Q$, we can assume that both P and Q are in HNF, in virtue of Lemma 4.4.5. Condition ϕ can be split into a decomposition D such that for each subcase $\psi \in D$:

1. under ψ, both $P = \sum_{i\in I}\alpha_i.P_i$ and $Q = \sum_{j\in J}\beta_j.Q_j$, that is P and Q are equal to some head normal form in the sense of [Mil89];
2. ψ is complete over the set V of names occurring free in the α_i's and β_j's.

By Lemma 4.4.7, we have that for each equivalence class C of $\mathcal{R}(\phi, V)$, $(\sum_{i\in I}\alpha_i.P_i)_C \simeq^{\psi} (\sum_{j\in J}\beta_j.Q_j)_C$. Exploiting the symbolic transitional semantics and the inductive hypothesis, one easily shows that the latter two terms are provably equivalent under ψ; the proof is a symbolic version of the classical one of [Mil89]. From this it easily follows that also P and Q are provably equivalent under ψ. Since the latter holds for each $\psi \in D$, we can conclude that $\phi \triangleright P = Q$ by applying the Cut rule. We come now to the actual proof.

Theorem 4.4.10 (Completeness of the proof system) *$P \simeq^{\phi} Q$ implies $\phi \triangleright P = Q$.*

PROOF: The proof goes by induction on $k = depth(\phi, P) = depth(\phi, Q)$. Suppose $k > 0$; thus we can assume that the thesis holds for each P', Q' and ϕ' s.t. $P' \simeq^{\phi'} Q'$ and $depth(\phi', P') < k$.

In virtue of Lemma 4.4.5, of the correctness of the proof system and of Lemma 4.4.9(2), we can suppose w.l.o.g. that both P and Q are in HNF.

It is therefore

$$P \equiv \sum_{i\in K}\phi_i R_i \quad\text{and}\quad Q \equiv \sum_{j\in H}\psi_j S_j.$$

Since $\phi \models \bigvee_{i,j}\phi\wedge\phi_i\wedge\psi_j$ (recall that by definition of HNF both $\{\phi_i|\, i\in K\}$ and $\{\psi_j|\, j\in H\}$ are $true$-decompositions), it will suffice to show that for each $i \in K$ and $j \in H$

$$\phi\wedge\phi_i\wedge\psi_j \triangleright P = Q \tag{4.5}$$

and then to apply the Cut rule. Thus fix $i \in K, j \in H$ and let $\chi \stackrel{def}{=} \phi\wedge\phi_i\wedge\psi_j$. Recall that (by definition of HNF) for $i' \neq i$ it is $\phi_i \models \neg\phi_{i'}$, and, for $j' \neq j$, $\psi_j \models \neg\psi_{j'}$; thus applying repeatedly $Guard2$, $Guard3$ and the Summation Axioms, we can write:

$$\chi \triangleright P = R_i \quad\text{and}\quad \chi \triangleright Q = S_j. \tag{4.6}$$

Thus it will suffice to show that:

$$\chi \triangleright R_i = S_j. \tag{4.7}$$

By definition of HNF, for suitable non-negative integers M, N it is:

$$R_i \equiv \sum_{l=1}^{M}\alpha_l.P_l \quad\text{and}\quad S_j \equiv \sum_{l=1}^{N}\beta_l.Q_l.$$

Define $\Gamma \stackrel{def}{=} \{\tau,\, input,\, free-output,\, bound-output\}$ and for each $\gamma \in \Gamma$

$$R_\gamma \stackrel{def}{=} \sum\{\alpha_l.P_l|\, \alpha_l \text{ is of kind } \gamma\} \text{ and } S_\gamma \stackrel{def}{=} \sum\{\beta_l.Q_l|\, \beta_l \text{ is of kind } \gamma\}.$$

Thus

$$R_i = \sum_{\gamma\in\Gamma}R_\gamma \quad\text{and}\quad S_j = \sum_{\gamma\in\Gamma}S_\gamma.$$

To show (4.7), it will suffice to show that, for each $\gamma \in \Gamma$

$$\chi \rhd R_\gamma = S_\gamma. \tag{4.8}$$

Now, we can write, for suitable I and J:

$$R_\gamma \equiv \sum_{i \in I} \alpha_i.P_i \quad \text{and} \quad S_\gamma \equiv \sum_{j \in J} \beta_j.Q_j.$$

Define $V \stackrel{def}{=} \cup_{i \in I} fn(\alpha_i) \cup \cup_{j \in J} fn(\beta_j)$ and assume $V = \{x_i, \ldots, x_k\}$, with the x_i's distinct. For each $x \in V$ and $V' \subseteq V$, define the formula:

$$\eta(x, V') \stackrel{def}{=} \bigwedge_{y \in V'} [x = y] \wedge \bigwedge_{x \in V - V'} [x \neq y].$$

Now, given k sets $V_1, \ldots, V_k \subseteq V$, define

$$\xi(V_1, \ldots, V_k) \stackrel{def}{=} \bigwedge_{x_i \in V} \eta(x_i, V_i)\,.$$

and let

$$\Xi \stackrel{def}{=} \{\xi(V_1, \ldots, V_k) |\, \text{with } V_1, \ldots, V_k \subseteq V\ \}.$$

It is easy to see that each $\xi \in \Xi$ is complete over V and that Ξ is a *true*-partition; the latter implies

$$\chi \models \bigvee\{\xi \wedge \chi |\, \xi \in \Xi\}\,. \tag{4.9}$$

Thus, to show (4.8), it will suffice to show that for each $\xi \in \Xi$:

$$\xi \wedge \chi \rhd R_\gamma = S_\gamma \tag{4.10}$$

and then to apply the *Cut* rule.

We will show now (4.10). Consider the equivalence relation $\mathcal{R}\,(\xi \wedge \chi, V)$: it will consist of t non-empty equivalence classes $C_1, \ldots, C_t$, thus, abbreviating $R_{\gamma C}$ as R_C, we can write:

$$R_\gamma = R_{C_1} + \cdots + R_{C_t} \quad \text{and} \quad S_\gamma = S_{C_1} + \cdots + S_{C_t}.$$

We will prove now that for each equivalence class C:

$$\xi \wedge \chi \rhd R_C = S_C \tag{4.11}$$

and this will establish (4.10) and hence the theorem. Fix C. Note that, since ξ is complete over V, so is $\xi \wedge \chi$. From $P \sim^{\phi} Q$, it follows $P \sim^{\chi} Q$; from this, (4.6) and from the correctness of the proof system we get $R_i \sim^{\chi} S_j$; this in turn implies $R_\gamma \sim^{\chi} S_\gamma$; hence $R_\gamma \sim^{\xi \wedge \chi} S_\gamma$. From the latter fact and Lemma 4.4.7, it follows:

$$R_C \sim^{\chi \wedge \xi} S_C\,. \tag{4.12}$$

At this stage, we have to distinguish the possible cases for γ; we only deal with the case $\gamma = bound - output$, since it is the most interesting. By α-equivalence, we can assume w.l.o.g. that for each $i \in I$ and $j \in J$, $bn(\alpha_i) = bn(\beta_j) = y$, with y fresh. Choose now $w \in C$. Since $\chi \wedge \xi \models [x = w]$ for each $x \in C$, by repeatedly applying $Subst2$, we can write

$$\begin{array}{ll} \chi \wedge \xi \rhd & R_C = \sum_{i \in I \mid subj(\alpha_i) \in C} \overline{w}(y).P_i \stackrel{def}{=} F \quad \text{and} \\ \chi \wedge \xi \rhd & S_C = \sum_{j \in J \mid subj(\beta_j) \in C} \overline{w}(y).Q_j \stackrel{def}{=} G\,. \end{array} \tag{4.13}$$

Now it is easy to show (4.11). We will show now that for each summand $\overline{w}(y).Q_j$ of G, it is

$$\chi \wedge \xi \rhd F = F + \overline{w}(y).Q_j. \tag{4.14}$$

From this, applying the congruence laws, it will follow $\chi \wedge \xi \rhd F = F + G$; symmetrically, it will be also $\chi \wedge \xi \rhd G = F + G$, from which it will follow $\chi \wedge \xi \rhd F = G$, that, together with (4.13), establishes (4.11).

We now come to show (4.14) for an arbitrary $\overline{w}(y).Q_j$. Observe that $G \stackrel{true, \overline{w}(y)}{\longrightarrow} Q_j$ and $F \simeq^{\chi \wedge \xi} G$ (that is implied by (4.13) and (4.12)) imply, by definition of symbolic bisimulation, that there exists a $(\chi \wedge \xi \wedge \bigwedge_{z \in fn(F,G,\chi \wedge \xi)} [y \neq z])$-decomposition D s.t. for each $\zeta \in D$ there exists a transition $F \stackrel{true, \overline{w}(y)}{\longrightarrow} P_{i_j}$ with

$$P_{i_j} \simeq^{\zeta} Q_j$$

hence also

$$P_{i_j} \simeq^{\zeta \wedge \chi \wedge \xi} Q_j.$$

Observe that $\zeta \wedge \chi \wedge \xi \models \chi \wedge \xi$, thus exploiting Lemma 4.4.9:

$$depth(\zeta \wedge \chi \wedge \xi, P_{i_j}) \leq depth(\chi \wedge \xi, P_{i_j}) < depth(\chi \wedge \xi, \overline{w}(y).P_{i_j}) \leq k$$

thus by inductive hypothesis:

$$\zeta \wedge \chi \wedge \xi \rhd P_{i_j} = Q_j$$

and hence:

$$\zeta \wedge \chi \wedge \xi \rhd \overline{w}y.P_{i_j} = \overline{w}y.Q_j.$$

Applying the congruence laws and the axioms for $+$, we obtain:

$$\zeta \wedge \chi \wedge \xi \rhd F = F + \overline{w}y.Q_j.$$

Since the above relation holds for each $\zeta \in D$ and $\{\zeta \wedge \chi \wedge \xi \mid \zeta \in D\}$ is a $(\chi \wedge \xi \wedge \bigwedge_{z \in fn(F,G,\chi \wedge \xi)} [y \neq z])$-decomposition, an application of the Cut rule yields:

$$\chi \wedge \xi \wedge \bigwedge_{z \in fn(F,G,\chi \wedge \xi)} [y \neq z] \rhd F = F + \overline{w}y.Q_j.$$

Applying the Res inference rule with y and the law $R3$, we obtain

$$(y)(\chi \wedge \xi) \wedge (y)(\bigwedge_{z \in fn(F,G,\chi\wedge\xi)} [y \neq z]) \rhd (y)F = (y)F + \overline{w}(y).Q_j. \tag{4.15}$$

Now, since $y \notin fn(F, G, \chi \wedge \xi)$, it is $(y)F = F$ (law $R0$) and $(y)(\chi \wedge \xi) = \chi \wedge \xi$; furthermore $(y)(\bigwedge_{z \in fn(F,G,\chi\wedge\xi)} [y \neq z])$ is equivalent to $true$; from these facts and (4.15) the desired (4.14) follows. □

4.5 Dealing with Late Bisimulation

We report in this section the definition of symbolic late bisimulation and the related theorems of consistency and adequacy. After that, we discuss a symbolic characterization of ground bisimulation and a proof system for symbolic late bisimulation.

In the sequel, $\sim_l$ will denote the standard ground late bisimulation and $\sim_l{}^{\phi}$ the closure of $\sim_l$ under ϕ. We will only deal with the input case of each theorem, since the other cases are formally the same as early bisimulation.

Symbolic late bisimulation is obtained by simply adding the condition $bn(\alpha) \cap n(\bigvee D) = \emptyset$ to the first item of the Definition 4.1.3: this amounts to imposing that no alternative of the decomposition depends on the "value" of the formal parameter $bn(\alpha)$ (i.e. to forbidding case-analysis on the actual value of $bn(\alpha)$).

Definition 4.5.1 (Symbolic late bisimulation)

- *A family $\mathcal{F} = \{R_\phi \,|\, \phi \in BF\}$ of symmetric binary relations over $\mathcal{P}$, indexed over the set BF of boolean formulae, is a* family of symbolic late bisimulations (FSLB) *iff* $\forall$ ϕ *and* $(P, Q) \in R_\phi$, $P \xrightarrow{\psi,\alpha} P'$, *with* $bn(\alpha) \cap fn(P, Q, \phi) = \emptyset$, *implies:*

  there exists a χ-decomposition D, with $bn(\alpha) \cap n(\bigvee D) = \emptyset$, such that for all $\phi' \in D$, there is a transition $Q \xrightarrow{\psi',\beta} Q'$ with $\phi' \models (\psi' \wedge [\alpha = \beta])$ and $(P', Q') \in R_{\phi'}$, where:

  $$\chi \stackrel{def}{=} \begin{cases} \phi \wedge \psi \wedge \bigwedge_{z \in fn(P,Q,\phi)} [y \neq z] & \textit{if } \alpha \textit{ is of the form } \overline{x}(y) \\ \phi \wedge \psi & \textit{otherwise.} \end{cases}$$

- $P \simeq_l^\phi Q$ *iff there exists a FSEB* $\{R_\psi | \, \psi \in BF\}$ *such that* $(P, Q) \in R_\phi$.

Theorem 4.5.2 (Consistency of symbolic late bisimulation) $P \simeq_l^\phi Q$ *implies* $P {\sim_l}^{\phi} Q$.

Proof: We will show that the relation:

$$\mathcal{R} \stackrel{def}{=} \{(P\sigma, Q\sigma) | \, \exists \, \phi. \, \sigma \models \phi \text{ and } P \simeq_l^\phi Q\}$$

is a ground late bisimulation. Suppose $P \simeq_l^{\phi} Q$ and $\sigma \models \phi$. Let us see how the moves of $P\sigma$ are matched by $Q\sigma$. Suppose that $P\sigma \xrightarrow{\alpha} P'$, with $bn(\alpha) \cap fn(P\sigma, Q\sigma) = \emptyset$; by Remark 2.1.5, we also assume $y \notin n(\sigma)$. We analyze the case when α is an input, $\alpha = x(y)$. It suffices to show that there exists Q' such that $Q\sigma \xrightarrow{\alpha} Q'$ and for each $z \in fn(P\sigma, Q\sigma, y)$ $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$. From $P\sigma \xrightarrow{\alpha} P'$ and Lemma 4.2.9, we deduce that for some ψ and some w, it is:

$$P \xrightarrow{\psi, w(y)} P''$$

where $\sigma \models \psi$, $w\sigma = x$ and $P''\sigma = P'$. Since $P \simeq_l^{\phi} Q$, there exists a $\phi \wedge \psi$-decomposition $D = \{\phi_1, \ldots, \phi_n\}$ s.t. $y \notin n(\bigvee D)$ and for each $i \in \{1, \ldots, n\}$ there exists a symbolic transition:

$$Q \xrightarrow{\psi_i, w_i(y)} Q_i$$

where $\phi_i \models \psi_i \wedge [w = w_i]$ and $P'' \simeq_l^{\phi_i} Q_i$. Now, $\sigma \models \phi \wedge \psi$ implies that for some $j \in \{1, \ldots, n\}$, $\sigma \models \phi_j$. Applying again Lemma 4.2.9 to the transition $Q \xrightarrow{\psi_j, w_j(y)} Q_j$, we get the matching transition for $Q\sigma$:

$$Q\sigma \xrightarrow{x(y)} Q_j\sigma.$$

Fix any $z \in fn(P\sigma, Q\sigma, y)$ and let $Q' \stackrel{def}{=} Q_j\sigma'$. We have $P'\{z/y\} = P''\sigma\{z/y\}$ and $Q'\{z/y\} = Q_j\sigma\{z/y\}$. Recall that $\sigma \models \phi_j$; since $y \notin n(\phi_j, \sigma)$, it is also $\phi_j\sigma\{z/y\} = \phi_j\sigma$ hence $\sigma\{z/y\} \models \phi_j$ as well. Recall also that $P'' \simeq^{\phi_j} Q_j$. Thus, by definition, $(P'\{z/y\}, Q'\{z/y\}) \in \mathcal{R}$. $\square$

We come now to the adequacy theorem. Lemma 4.2.10 and Lemma 4.2.11 extend of course to the late case (i.e., they still hold if replacing $\sim$ with $\sim_l$).

Theorem 4.5.3 (Adequacy of symbolic late bisimulation) *$P {\sim_l}^{\phi} Q$ implies $P \simeq_l^{\phi} Q$.*

PROOF: We will show that the the family of relations

$$\mathcal{F} \stackrel{def}{=} \{{\sim_l}^{\phi} \mid \phi \in BF\}$$

is a family of symbolic late bisimulations. Thus let $\phi \in BF$ and $P {\sim_l}^{\phi} Q$ and suppose $P \xrightarrow{\psi, \alpha} P'$ with $bn(\alpha) \cap fn(P, Q, \phi) = \emptyset$. We deal only with the case when α is an input. Thus we have to find a $\phi \wedge \psi$-decomposition $D = \{\phi_1, \ldots, \phi_n\}$ s.t. for each $i \in \{1, \ldots, n\}$ $y \notin n(\phi_i)$ and there exists a symbolic transition $Q \xrightarrow{\psi_i, \beta_i} Q_i$ with $\phi_i \models \psi_i \wedge [\alpha = \beta_i]$ and furthermore $P' {\sim_l}^{\phi_i} Q_i$.

Let $V \stackrel{def}{=} fn(P, Q, \phi)$ and $V_y \stackrel{def}{=} V \cup bn(\alpha)$. Take any $F \subseteq_{\mathrm{fin}} \mathcal{N} - V_y$ s.t. $|F| > 2 * |V_y|$ and define the set of substitutions:

$$S \stackrel{def}{=} \{\sigma \mid n(\sigma) \subseteq F \cup V \text{ and } \sigma \models \phi \wedge \psi\}.$$

S is clearly finite We define $D \stackrel{def}{=} \{\chi(\sigma, V) \mid \sigma \in S\}$. The fact that D is a $\phi \wedge \psi$-decomposition follows from Lemma 4.2.2, case a. (read the "ϕ" of the statement of the lemma as $\phi \wedge \psi$ and the "ψ" as $\bigvee D$)[3]. We now prove that each $\chi(\sigma, V) \in D$ entails a matching symbolic transition for Q. Thus fix $\sigma \in S$ and consider the following chain of implications:

$$\begin{array}{rl}
P \xrightarrow{\psi,\alpha} P' & \text{implies, in virtue of Lemma 4.2.9, since } \sigma \models \psi \text{ and } y \notin n(\sigma) \\
P\sigma \xrightarrow{\gamma} P'\sigma & \text{with } \alpha\sigma = \gamma\text{; from } P{\sim_l}^{\phi}Q \text{ and } \sigma \models \phi\text{, for some } Q' \\
Q\sigma \xrightarrow{\gamma} Q' & \text{where for each } z \in fn(P\sigma, Q\sigma, y) : P'\sigma\{z/y\} \sim_l Q'\{z/y\}. \\
 & \text{From Lemma 4.2.9, there exists a symbolic transition:} \\
Q \xrightarrow{\psi',\beta} Q'' & \text{with } \sigma \models \psi' \wedge [\beta = \alpha] \text{ and } Q''\sigma = Q'\ .
\end{array}$$

We now prove that $\chi(\sigma, V) \models \psi' \wedge [\beta = \alpha]$ and that $P'{\sim_l}^{\chi(\sigma,V)}Q''$.

Consider any ρ s.t. $\rho \models \chi(\sigma, V)$; we have to show that $\rho \models \psi' \wedge [\beta = \alpha]$; in virtue of Lemma 4.2.6, $n(\psi' \wedge [\beta = \alpha]) \subseteq V$; since $\sigma \models \psi' \wedge [\beta = \alpha]$, by Lemma 4.2.5 we get $\rho \models \psi' \wedge [\beta = \alpha]$ as well.

We show now that $P'{\sim_l}^{\chi(\sigma,V)}Q''$. Consider any substitution ρ s.t. $\rho \models \chi(\sigma, V)$; we have to show that $P'\rho \sim_l Q''\rho$. From above, we know that for each $z \in fn(P'\sigma, Q''\sigma, y)$, it is $P'\sigma\{z/y\} \sim_l Q''\sigma\{z/y\}$. Taken any $w \in V$ (note that $V \neq \emptyset$, in that $x \in V$), it follows that $(w(y).P')\sigma \sim_l (w(y).Q'')\sigma$. Since $fn(w(y).P', w(y).Q'') \subseteq V$ and by hypothesis $\rho \models \chi(\sigma, V)$, it follows from Lemma 4.2.11 that $(w(y).P')\rho \sim_l (w(y)Q'')\rho$. From the latter it follows $P'\rho \sim Q''\rho$. $\square$

The discussion of Section 4.1 on automatic verification extends to the late case as well, by considering the late version of Hennessy and Lin's algorithm.

Also ground late bisimulation $\sim_l$ can be characterized in terms of the symbolic late one.

Theorem 4.5.4 *$P \sim_l Q$ iff $P \simeq_l^{\phi} Q$, where $\phi = \bigwedge_{x,y \in fn(P,Q),\, x,y \text{ distinct}} [x \neq y]$.*

A sound and complete proof system for late bisimulation can be obtained by replacing the Inp rule of the system considered in Section 4.4 with the simpler rule:

$$(Inp - L) : \frac{\phi \rhd P = Q}{\phi \rhd x(y).P = x(y).Q}\ y \notin n(\phi).$$

The corresponding provability relation is denoted by $\rhd_l$. The proofs of soundness and completeness can be obtained by slightly modifying those for the early case, thus we omit them and state:

Theorem 4.5.5 *$\phi \rhd_l P = Q$ iff $P \simeq_l^{\phi} Q$.*

[3]Observe that in this case y plays essentially no role.

Chapter 5

Causal Semantics

Most theoretical research on the π-calculus has dealt with interleaving semantics [MPW92, BD92a, PS93, San92, San93, Wal94]; a major concern has been investigation of its expressiveness [Mil92, San92]. The non-interleaving side of the π-calculus remains largely unexplored; only very recently, the analysis of a location semantics has appeared [San94a].

The main goal of this chapter is to investigate the the relationship between interleaving and causal semantics for the π-calculus. We focus on the interleaving observation equivalence, written $\approx$ [Mil89, MPW92], and on the non-interleaving *causal bisimulation*, written $\approx_c$ [DDM88, DD89, Kie91]; these are among the most studied behavioural equivalences in the respective groups. W.r.t. observation equivalence, causal bisimulation makes additional distinctions on processes on the basis of the causal dependencies for their actions. Our major result shows that causal bisimulation can be reconducted to, or implemented into, observation equivalence. This allows us to use the simpler theory of the interleaving semantics to reason about the causal one; the result also evidences the expressive power of the π-calculus.

Causal bisimulation represents quite a stable relation. Several approaches to causality have independently led to it. Examples include Degano, De Nicola and Montanari' *history preserving bisimulation* [DDM88], Darondeau and Degano' *causal-tree bisimulation* [DD89] and Kiehn's *global-cause bisimulation* [Kie91]. (Comparisons of causal bisimulation with other interleaving and non-interleaving equivalences can be found in [Ace92b, Ace92a, DDM93].) Known descriptions of causal bisimulation over CCS-like languages are based on operational transition rules more complex than the standard interleaving ones [Mil89], because they carry information about causal dependencies, in form of pointer-like objects among actions. This extra structure also has to be lifted at the syntactic level, forcing an extension of the basic language; existing algebraic theories and proof systems of causal bisimulation [DD89, Kie93] heavily rely on the added causal operators. A motivation for trying to implement causal bisimulation at the interleaving level is to seek ways for avoiding the introduction of this extra structure.

Another goal of this chapter is to investigate the meaning of causality in calculi for mobile processes. We have individuated two forms of dependencies between actions, which we have called *subject* and *object* dependencies. Subject dependencies originate from the nesting of prefixes and are propagated through internal communications of processes. For example, the processes $a.b.\mathbf{0} + b.a.\mathbf{0}$ can perform actions a and b in either order. In either case, there is a dependency between the two actions — the firing of the first enables the firing of the second. The propagation of subject dependencies through interactions is shown in $\nu\, c\,(a.c.\mathbf{0} \mid \overline{c}.b.\mathbf{0})$ (we use the π-calculus notation $\nu\, a\, P$ for the restriction of name a in process P). In this process, the interaction at c creates a dependency of b from a; indeed, it holds that $\nu\, c\,(a.c.\mathbf{0} \mid \overline{c}.b.\mathbf{0})$ is causally equivalent with $a.b.\mathbf{0}$. In CCS-like languages, i.e., languages in which actions represent pure synchronization, subject dependencies are the only form of causal dependencies. Subject dependencies cannot be detected at the interleaving level. For example, one cannot detect the dependency between the actions a and b of the process $a.b.\mathbf{0} + b.a.\mathbf{0}$. Hence, in an interleaving semantics this process is equated to the process $a.\mathbf{0} \mid b.\mathbf{0}$; this equality is rejected in a causal semantics because in $a.\mathbf{0} \mid b.\mathbf{0}$ actions a and b are independent.

The other form of dependencies, namely the object dependencies, are determined by π-calculus binding mechanisms on actions; therefore they do not exist in CCS. Approximately, an action λ is object dependent from a previous action μ if μ acts as binder for some name of λ; as such, μ can be thought of as supplying some names to λ. There are two binding actions in the π-calculus, namely input and bound output — the latter denoting the output of a restricted, i.e., private, name.

By contrast with subject dependencies, the possible object dependencies in a process can be detected at the interleaving level, simply revealed by the occurrences of names in actions. For instance, consider the bound output transition $P_1 \xrightarrow{(\nu\, b)\overline{a}\langle b\rangle} P_2$, which says that P_1 can perform the output at a of the private name b and become P_2 in doing so; the object dependencies from the action $(\nu\, b)\overline{a}\langle b\rangle$ are given by those actions in the successive behaviour of P_2 in which b occurs free. A similar consideration can be made for input transitions $P_1 \xrightarrow{a(b)} P_2$ — which says that P_1 can perform the input at a of b so becoming P_2 — if name b is not already present in P_1. In other words, observationally equivalent processes show the same object dependencies.

In most of the cases, an object dependency is also a subject dependency (for instance, this is always the case for the object dependencies arising from input actions). There are, however, exceptions. For instance, the process $P \stackrel{def}{=} \nu\, b\,\big(\overline{a}\langle b\rangle.\,\mathbf{0} \mid \overline{b}\langle y\rangle.\,\mathbf{0}\big)$ can perform the output at a of the restricted name b and then an output at b; the former action enables the latter, because it "opens" the restriction at b. There is an object dependency of the action at b from the action at a, but no subject dependency.

Here, we have chosen to separate subject dependencies from object dependencies, because they represent logically different forms of causality. Hence we shall want to distinguish processes which differ for the subject or for the object dependencies. For instance, we shall discriminate between the process P above and the process $Q \stackrel{def}{=} \nu\, b\,(\overline{a}\langle b\rangle.\,\overline{b}\langle y\rangle.\,\mathbf{0})$.

Both processes can perform an action at a and then an action at b. However, in Q there is a double causal dependency between the two actions: one given by a prefix-nesting (i.e., a subject dependency), and the other given by a name-binding (i.e., an object dependency). By contrast, in P there is a single dependency between the two actions, given by name-binding.

To define a causal bisimulation which reflects these discriminations on processes, we enrich the standard (i.e., interleaving) transition system of the π-calculus with information about the subject dependencies. This is enough because the object dependencies can already be detected in the standard system. Our enriched system is defined following Kiehn's system [Kie91], which describes causal bisimulation in CCS (as noted above, in CCS all causal dependencies are subject dependencies). This implies adding explicit causes to π-calculus syntax and operational semantics. Causal bisimulation is defined on top of the enriched system using familiar bisimulation techniques.

We then show that the sophisticated observational machinery of causal bisimulation ($\approx_{\mathrm{c}}$) can be represented from within the interleaving observation equivalence ($\approx$). We define a compositional encoding $[\![\,.\,]\!]$ from the enriched calculus to the basic one, and we prove that it is fully abstract w.r.t. $\approx_{\mathrm{c}}$ and $\approx$; i.e., for each pair of terms A and B in the enriched calculus we have:

$$A \approx_{\mathrm{c}} B \text{ if and only if } [\![A]\!] \approx [\![B]\!] .$$

Through various examples, we show how this result and the algebraic theory of $\approx$ can be used to prove properties about $\approx_{\mathrm{c}}$. When restricted to the sublanguage with no object dependencies, that is CCS, our encoding yields a characterisation of the classic causal bisimulation of [DD89, Kie91] in terms of the observation equivalence of the monadic π-calculus.

In the definition of encoding, we exploit an important feature of the π-calculus: the calculus naturally permits the description of *data structures* which can be created and combined at run time. These data structures are modeled as standard processes accessible via private names, which can be passed around and used to form more complex structures. We use data structures called *wires* to encode the observability of the explicit causes used in the definition of causal bisimulation. A wire receives names at an *entrance-point* and retransmit them at a bunch of *end-points*. Several wires can be connected together, connection representing union of causes. Reachability among wires, i.e. if an end-point of a wire can be reached from the entrance-point of another wire, encodes the subject dependencies between actions. When a target process $[\![A]\!]$ performs a visible action μ, a wire is generated and a pointer to the wire — under the form of a fresh name — is emitted. This pointer can be used to 'explore' the wire and determine the subject dependencies for μ. Transmission of causes across a parallel composition, a distinguishing feature of causal bisimulation, is rendered as an exchange of a private name, by which two previously separated wires get connected. The proof of our full abstraction theorem exploits a few properties of concatenation of wires that permit symbolic manipulations of these structures.

This chapter mainly relates to [San94a]. There, a programme similar to that of the present chapter is carried out for *location bisimulation* [BCHK94], one of the most convincing spatial-sensitive semantics. An encoding from an enriched π-calculus of *located processes* to the standard π-calculus is given and proved to be fully abstract w.r.t. location bisimulation and observation equivalence. The treatment of causes in communications is what distinguishes causal bisimulation from location bisimulation: in the former, the causes of the interacting actions must be appropriately merged, whereas in the latter they remain separate. This explains the difference between the data structures (the wires) used in the encoding of the two equivalences, here and in [San94a]. In the case of causal bisimulation, the connections among wires, established as the computation proceeds, may create *branching* chains; by contrast, in the case of location bisimulation connections among wires may only create *linear* chains. The difference also shows up in the proofs: although the overall structure of the proofs of full abstraction is similar, the technical details are quite different, sometimes strikingly so. A thorough comparison between causal and location bisimulation via their encodings into observation equivalence presented here and in [San94a] is in Section 5.9. The outcomes of this comparison are certainly not new. Careful studies of the relationship between causal- and spatial-sensitive equivalences can be found in the literature [Ace92b, CD93, MY92, Kie91, Kie93, DP92]. The use of the encodings gives us a refreshing perspective on the issue.

The remainder of the chapter is organized as follows. In Section 2, we fix the syntax we shall work with and review the definition of observation equivalence. In Section 3 we present the proof techniques (mainly various forms of "bisimulation up-to") that will be extensively used in later sections. In Section 5.3 we introduce the enriched language of causal terms, and we define causal bisimulation. In Section 5.4 we present the encoding $[\![\,.\,]\!]$. Its full abstraction is proved in Sections 5.5 through 5.7, which are organized as follows. In Section 5.5, some important properties of wire processes are shown. These properties are used in Section 5.6, to establish the operational correspondence between transitions of causal terms and transitions of encoded processes, and in Section 5.7, to derive the main theorems. In Section 5.8 we show some examples of applications of the encoding. In Section 5.9 causal bisimulation and location bisimulation are compared using their encodings into observation equivalence presented here and in [San94a].

5.1 Preliminaries

We start by fixing the syntax of the variant of the π-calculus we shall work with: it is the polyadic π-calculus discussed in Chapter 2, Section 2.2. In the actual development of the chapter, it will be essential the fact that our language is parametric with the chosen class of names $\mathcal{N}$. To emphasize this fact, we shall denote the language by $\mathcal{P}_3(\mathcal{N})$. Its actual syntax is recalled below:

$$P \ ::= \ \mathbf{0} \ | \ a(\widetilde{b}).\,P \ | \ \overline{a}\langle\widetilde{b}\rangle.\,P \ | \ \tau.\,P \ | \ P_1 + P_2 \ | \ P_1 \,|\, P_2 \ | \ \nu\, a\, P \ | \,!P\,.$$

The definitions and conventions on syntax, sorting and operational semantics are those fixed in Chapter 2, Section 2.2. In particular, we assume that all processes are well-sorted for some fixed but arbitrary sorting Ob. We only adopt *early* operational semantics (the one we described in Section 2.2), that naturally leads to early bisimulation. The early system permits a more uniform treatment of bisimulation, but this choice is not critical at all: all results can be rephrased for the late case. From now on we simply omit the adjective “early”.

Notations

We write $\Pi_{i\in I}P_i$ as an abbreviation for $P_{i_1} \,|\, \ldots \,|\, P_{i_n}$, $I = \{i_1, \ldots, i_n\}$. Symbol I will range over finite indexing sets. In an assertion, we say that a name is *fresh* to mean that it it is different from any other name which is nominated in the assertion or which occurs in a process nominated in the assertion.

Remark 5.1.1 The above grammar does not have the operator of match, and uses replication rather than recursive definitions of agents. We have omitted matching not because of difficulties in its treatment, but because we think that causal bisimulation has interesting congruence properties in the calculus without matching. A discussion on this is deferred to the concluding section. We adopted replication because it suffices to code up recursion (the encoding showed by Milner [Mil91, section 3.1] for the ordinary interleaving bisimilarity also works well for causal bisimulation) and is easier to handle algebraically. $\Box$

In subsequent sections, the following interleaving bisimulation-based relations over $\mathcal{P}_3(\mathcal{N})$ will be considered: strong bisimulation equivalence ($\sim$), strong bisimulation congruence ($\sim^c$) and weak observation equivalence ($\approx$). Definitions and a few properties of these relations are reported in Chapter 2, Section 2.3.

5.2 Proof Techniques

For the proof of the results in this chapter, we appeal to the expansion relation and to various “up-to” techniques.

5.2.1 The expansion relation

The expansion relation $\lesssim$ [AKH92, SM92] is an asymmetric variant of $\approx$ which allows us to count the number of τ-actions performed by the processes. Thus, $P \lesssim Q$ holds if

$P \approx Q$ but also Q has at least as many τ-moves as P. The expansion relation provides us with better 'control' on τ-moves than the ordinary $\approx$.

Definition 5.2.1 (Expansion) *A relation $\mathcal{R}$ is an* expansion *if $P \,\mathcal{R}\, Q$ implies:*

1. *Whenever $P \xrightarrow{\mu} P'$, there exists Q' s.t. $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \,\mathcal{R}\, Q'$;*
2. *whenever $Q \xrightarrow{\mu} Q'$, there exists P' s.t. $P \xrightarrow{\widehat{\mu}} P'$ and $P' \,\mathcal{R}\, Q'$.*

We say that Q expands *P, written $P \lesssim Q$, if $P \,\mathcal{R}\, Q$, for some expansion $\mathcal{R}$.*

Note that, in contrast with the other relations considered in the chapter, $\lesssim$ is not symmetric. For instance, $P \lesssim \tau. P$, but $\tau. P \not\lesssim P$ since $\tau. P$ has to perform more τ-actions than P in order to mimic its actions. Relation $\lesssim$, which is indeed a preorder, enjoys the same congruence properties as $\approx$. The following proposition sums up the relationship among the interleaving behavioural relations considered in the chapter.

Proposition 5.2.2 *The following is a chain of strict inclusions: $\sim^c \subset \sim \subset \lesssim \subset \approx$.* $\square$

5.2.2 The "up-to" techniques

The "up-to" techniques allow us to reduce the size of a relation $\mathcal{R}$ to exhibit for proving bisimilarities [Mil89, SM92, San94b]. The up-to techniques which we use in the chapter allow us to manipulate the derivatives of a pair of processes in two ways: first, we can replace such derivatives with behaviourally-equivalent processes; secondly, we can cut common contexts of the derivatives. The contexts we cut are of the form $\nu\, \widetilde{c}\, (R \mid [\cdot])$; they are called *static contexts*.

Definition 5.2.3 ($\approx$-bisimulation up-to context and up-to $\gtrsim$) *A symmetric relation $\mathcal{R}$ is a* $\approx$-bisimulation up-to context and up-to $\gtrsim$ *if $P \,\mathcal{R}\, Q$ and $P \stackrel{\mu}{\Longrightarrow} P''$ imply that there are a static context $C[\cdot]$ and processes P' and Q' s.t. $P'' \gtrsim C[P']$, $Q \stackrel{\widehat{\mu}}{\Longrightarrow}\ \gtrsim C[Q']$ and $P' \,\mathcal{R}\, Q'$.*

Proposition 5.2.4 *If $\mathcal{R}$ is a $\approx$-bisimulation up-to context and up-to $\gtrsim$, then $\mathcal{R} \subseteq \approx$.*

PROOF: By showing that

$$\mathcal{R}' = \{(P,Q) : \begin{array}{l}\text{for some static context } C[\cdot] \text{ and processes } P', Q' \\ \text{it holds that } P \gtrsim C[P'], Q \gtrsim C[Q'] \text{ and } P' \,\mathcal{R}\, Q' \quad \}\end{array}$$

is a $\approx$-bisimulation. This proves the thesis because $\mathcal{R} \subseteq \mathcal{R}'$. Details of the proof can be found in [San94a]. $\square$

We do not know whether the above soundness theorem holds if Definition 5.2.3 is weakened by replacing all occurrences of $\lesssim$ with $\approx$.

The next up-to technique is new; it is the corresponding for expansion of Definition 5.2.3. Note the occurrence of $\sim$ in clause (1) of Definition 5.2.5: we do not know whether the replacement of $\sim$ with $\lesssim$ preserves the soundness of the technique.

Definition 5.2.5 (Expansion up-to context and up-to $\gtrsim$) *A relation $\mathcal{R}$ is an* expansion up-to context and up-to $\gtrsim$ *if $P \mathcal{R} Q$ implies:*

1. *Whenever $P \xrightarrow{\mu} P''$, there are a static context $C[\cdot]$ and processes P' and Q' s.t. $P'' \sim C[P']$, $Q \stackrel{\mu}{\Longrightarrow} \gtrsim C[Q']$ and $P' \mathcal{R} Q'$;*
2. *whenever $Q \xrightarrow{\mu} Q''$, there are a static context $C[\cdot]$ and processes P' and Q' s.t. $Q'' \gtrsim C[Q']$, $P \xrightarrow{\widehat{\mu}} \lesssim C[P']$ and $P' \mathcal{R} Q'$.*

Proposition 5.2.6 *If $\mathcal{R}$ is an expansion up-to context and up-to $\gtrsim$, then $\mathcal{R} \subseteq \lesssim$.*

Proof: By showing that

$$\{(P,Q) : \begin{array}{l}\text{for some static context } C[\cdot] \text{ and processes } P', Q' \\ \text{it holds that } P \lesssim C[P'], Q \gtrsim C[Q'] \text{ and } P' \mathcal{R} Q' \quad \}\end{array}$$

is an expansion relation. The proof is similar to the proof of Proposition 5.2.4. □

5.3 Causal Bisimulation

The section begins with a discussion on the way object and subject dependencies in processes can be detected. Whereas the former are detectable with the standard transition system, the latter require a richer system, with explicit causes. We show a few properties of the richer system and then define causal bisimulation on top of it.

Object dependencies are the dependencies determined by the binding mechanisms on names. To know the object dependencies in a process, we look at its *runs*. A run of a process is a sequence of transitions starting from that process.

Definition 5.3.1 *A run $P_1 \xrightarrow{\mu_1} P_2 \ldots P_n \xrightarrow{\mu_n} P_{n+1}$, $n \geq 1$, is* fresh *when for all $1 \leq i \leq n$ if $\mu_i = \nu \widetilde{b}\, \overline{a}\langle \widetilde{c} \rangle$ or $\mu_i = a\langle \widetilde{b} \rangle$, then $\widetilde{b} \cap fn(P_i) = \emptyset$ and for all $j < i$, $\widetilde{b} \cap n(\mu_j) = \emptyset$; in this case, we say that names $\widetilde{b}$ are* introduced *in μ_i.*

In a fresh run, names exported in an output or imported in an input are all fresh. Fresh runs reveal all possible name dependencies in processes in a simple way.

Definition 5.3.2 *Given a fresh run $P_1 \xrightarrow{\mu_1} P_2 \ldots P_n \xrightarrow{\mu_n} P_{n+1}$, we say that μ_i is* object dependent *on μ_j, $1 \leq j < i \leq n$, if there is a name introduced in μ_j which is among the free names of μ_i.*

For instance, in the run $\nu\, b\,(b \mid a(x).\,\overline{x}\langle b\rangle) \xrightarrow{a\langle c\rangle} \xrightarrow{\nu\, b\, \overline{c}\langle b\rangle} \xrightarrow{b} \mathbf{0} \mid \mathbf{0}$, the second action is object dependent on the first and the third is object dependent on the second. Object dependencies can already be observed at the interleaving level, since they are shown by the occurrences of names in actions: names themselves act as pointers among actions. Thus, observationally equivalent processes exhibit the same object dependencies.

Proposition 5.3.3 *If $P_1 \approx Q_1$ and P_1 has a fresh run $P_1 \xrightarrow{\mu_1} P_2 \ldots P_n \xrightarrow{\mu_n} P_{n+1}$ where μ_i is object dependent on μ_j, then also Q_1 has a fresh run $Q_1 \xrightarrow{\mu_1} Q_2 \ldots Q_n \xrightarrow{\mu_n} Q_{n+1}$ where μ_i is object dependent on μ_j.* $\square$

The other form of dependencies among actions, namely the *subject dependencies*, are determined by the nesting of prefixes. By contrast with the object dependencies, the subject dependencies are not detected in an interleaving semantics (see discussion in Section 5). To capture them, we add explicit causal information to the standard syntax and operational semantics of the calculus. For this, we follow Kiehn's system [Kie91], which describes the causal dependencies for CCS processes, and transport it onto the π-calculus.

For the extra causal information, we use an auxiliary set $\mathcal{K}$ of *causes*. In the enriched system, visible transitions are of the form $A \xrightarrow[K;\,k]{\mu} A'$. The visible action μ is associated with a unique cause $k \in \mathcal{K}$. The set of actions from which μ is causally related is revealed by means of the set $K \subseteq_{\mathrm{fin}} \mathcal{K}$ of their associated causes. We call K the *cause-set* for μ. If a successive action μ' is caused by μ, then μ' will have k (i.e., the cause associated with μ) in its cause-set. In the term syntax, the explicit use of causes is accompanied by the introduction of a causal prefix $K :: A$, for $K \subseteq_{\mathrm{fin}} \mathcal{K}$, which says that the cause-set of any action of the process A must contain K.

By contrast, silent actions are not observable and do not exhibit causes. However, causes do play an important role in the communication rule, for the causes of two interacting actions must be appropriately merged. Consider the sequence of transitions:

$$
\begin{array}{lcl}
\nu\, b\,\big(a.\,b.\,c.\,\mathbf{0} \mid d.\,\overline{b}.\,\mathbf{0}\big) & \xrightarrow[\emptyset\,;\,k_1]{a} & \nu\, b\,\big(\{k_1\} :: b.\,c.\,\mathbf{0} \mid d.\,\overline{b}.\,\mathbf{0}\big) \\
& \xrightarrow[\emptyset\,;\,k_2]{d} & \nu\, b\,\big(\{k_1\} :: b.\,c.\,\mathbf{0} \mid \{k_2\} :: \overline{b}.\,\mathbf{0}\big) \\
& \xrightarrow{\tau} & \nu\, b\,\big(\{k_1, k_2\} :: c.\,\mathbf{0} \mid \{k_1, k_2\} :: \mathbf{0}\big) \\
& \xrightarrow[\{k_1, k_2\};\,k_3]{c} & \nu\, b\,\big(\{k_1, k_2\} :: \{k_3\} :: \mathbf{0} \mid \{k_1, k_2\} :: \mathbf{0}\big)\,.
\end{array}
$$

They show that the actions at a and d have no cause; and that the action at c is causally dependent on a and d. The latter makes sense: c could fire only because both a and d fired. Note that in the τ-transition, the cause-sets $\{k_1\}$ and $\{k_2\}$ of the consumed actions are merged.

5.3.1 Syntax and transitional semantics of causal terms

The notations and conventions introduced on standard processes — including the identification of alpha equivalent processes — extend to the enriched language of causal processes below, and will not be repeated.

We let k and h range over causes (i.e., elements of $\mathcal{K}$), K and H over finite subsets of $\mathcal{K}$, and A and B over causal processes. We denote by $\mathcal{K}(A)$ the set of causes which appear in the causal process A. To improve readability, in the syntax of our terms and of our operational rules we abbreviate a union of sets of causes $K \cup K'$ as K, K' and we write a singleton set $\{k\}$ simply as k. Thus, $A \xrightarrow[K,K';k]{\mu} A'$ stands for $A \xrightarrow[K \cup K';k]{\mu} A'$, and $k :: A$ stands for $\{k\} :: A$. [1]

Definition 5.3.4 (Causal terms) *Consider an infinite countable set $\mathcal{N}$ of names, and an infinite countable set $\mathcal{K}$ of* causes *with $\mathcal{K}$, $\mathcal{N}$ and $\{\tau\}$ pairwise disjoint. The language of* polyadic causal processes over causes $\mathcal{K}$ and names $\mathcal{N}$, *written $\mathcal{P}_c(\mathcal{N},\mathcal{K})$, is given by the following grammar:*

$$A := K :: A \quad | \quad A \,|\, A \quad | \quad \nu\, a\, A \quad | \quad P$$

where P is a $\mathcal{P}_3(\mathcal{N})$ term (i.e., a standard term) and $K \subseteq_{\text{fin}} \mathcal{K}$.

We do not allow the presence of causes underneath dynamic operators (prefixes, sums and replications), because we are only interested in derivatives of standard processes, for which these cases may never arise. This is clear from the transition rules for causal terms, in Table 5.1. The rules for the prefixes and for communication express the essence of the whole semantics: `C-Inp` and `C-Out`, for input and output prefixes, introduce causes into processes; `C-Cau`, for the causal prefix $K :: A$, makes any action performed by A dependent upon K. Rule `C-Com` implements the cause exchange discussed at the beginning of this section. In the rule, the notation $A[k \rightsquigarrow K]$ indicates the replacement of k with the elements of K, in any set of causes in A. For instance, $(\{k_1, k_2\} :: \alpha.\mathbf{0})[k_2 \rightsquigarrow \{k_3, k_4\}]$ is $\{k_1, k_3, k_4\} :: \alpha.\mathbf{0}$. The rules for the remaining operators are formally identical to the standard rules of the polyadic calculus.

Our rules, when restricted to the sublanguage of finite 0-adic (i.e. CCS) causal terms, coincide with Kiehn's rules [Kie91]. The only variation is in rule `C-Com`: in our case, processes A_1 and A_2 use the same fresh cause k, whereas the obvious π-calculus's adaptation of Kiehn's rule, called `C-Com'` below, uses two distinct causes:

$$\texttt{C-Com}' \;:\; \frac{A_1 \xrightarrow[K_1;k_1]{(\nu\,\widetilde{b}')\overline{a}\langle\widetilde{b}\rangle} A_1' \quad A_2 \xrightarrow[K_2;k_2]{a\langle\widetilde{b}\rangle} A_2'}{A_1 \mid A_2 \xrightarrow{\tau} \nu\,\widetilde{b}'\,(A_1'[k_1 \rightsquigarrow K_2] \mid A_2'[k_2 \rightsquigarrow K_1])} \quad \begin{array}{l} k_1 \notin \mathcal{K}(A_1), \\ k_2 \notin \mathcal{K}(A_2), \\ \widetilde{b}' \cap fn(A_2) = \emptyset\ . \end{array}$$

[1] The prefix $k :: A$ is taken as a primitive operator in [Kie91].

Rules for visible transitions:

$$\texttt{C-Out} : \overline{a}\langle\widetilde{b}\rangle.\,A \xrightarrow[\emptyset\,;\,k]{\overline{a}\langle\widetilde{b}\rangle} k :: A \qquad \texttt{C-Inp} : a(\widetilde{b}).\,A \xrightarrow[\emptyset\,;\,l]{a(\widetilde{c})} k :: A\{\widetilde{c}/\widetilde{b}\}$$

$$\texttt{C-Cau} : \frac{A \xrightarrow[K;k]{\mu} A'}{K' :: A \xrightarrow[K,K';k]{\mu} K' :: A'} \qquad \texttt{C-Par} : \frac{A_1 \xrightarrow[K;k]{\mu} A_1'}{A_1 \mid A_2 \xrightarrow[K;k]{\mu} A_1' \mid A_2}\; bn(\mu)\cap fn(A_2) = \emptyset$$

$$\texttt{C-Res} : \frac{A \xrightarrow[K;k]{\mu} A'}{\nu\, c\, A \xrightarrow[K;k]{\mu} \nu\, c\, A'},\; c \notin n(\mu) \qquad \texttt{C-Open} : \frac{A \xrightarrow[K;k]{(\nu\,\widetilde{b'})\overline{a}\langle\widetilde{b}\rangle} A'}{\nu\, c\, A \xrightarrow[K;k]{(\nu\,\widetilde{b'}c)\overline{a}\langle\widetilde{b}\rangle} \nu\, c\, A'},\; \begin{array}{l} c \neq a, \\ c \in \widetilde{b} - \widetilde{b'} \end{array}$$

$$\texttt{C-Sum} : \frac{A_1 \xrightarrow[K;k]{\mu} A_1'}{A_1 + A_2 \xrightarrow[K;k]{\mu} A_1'} \qquad \texttt{C-Rep} : \frac{A \mid !\,A \xrightarrow[K;k]{\mu} A'}{!\,A \xrightarrow[K;k]{\mu} A'}$$

Rules for silent transitions:

$$\texttt{T-pre} : \tau.A \xrightarrow{\tau} A \qquad \texttt{T-sum} : \frac{A_1 \xrightarrow{\tau} A_1'}{A_1 + A_2 \xrightarrow{\tau} A_1'}$$

$$\texttt{T-par} : \frac{A_1 \xrightarrow{\tau} A_1'}{A_1 \mid A_2 \xrightarrow{\tau} A_1' \mid A_2} \qquad \texttt{T-res} : \frac{A \xrightarrow{\tau} A'}{\nu\, c\, A \xrightarrow{\tau} \nu\, c\, A'}$$

$$\texttt{C-Com} : \frac{A_1 \xrightarrow[K_1;k]{(\nu\,\widetilde{b'})\overline{a}\langle\widetilde{b}\rangle} A_1' \qquad A_2 \xrightarrow[K_2;k]{a(\widetilde{b})} A_2'}{A_1 \mid A_2 \xrightarrow{\tau} \nu\,\widetilde{b'}\,(A_1'[k \rightsquigarrow K_2] \mid A_2'[k \rightsquigarrow K_1])}\; \begin{array}{l} \widetilde{b'} \cap fn(A_2) = \emptyset\,, \\ k \notin \mathcal{K}(A_1, A_2) \end{array}$$

$$\texttt{T-cau} : \frac{A \xrightarrow{\tau} A'}{K :: A \xrightarrow{\tau} K :: A'} \qquad \texttt{T-rep} : \frac{A \mid !\,A \xrightarrow{\tau} A'}{!\,A \xrightarrow{\tau} A'}$$

Table 5.1: SOS rules for causal terms.

Anyhow, the two rules `C-Com` and `C-Com'` are interderivable, as can be proved using Lemma 5.3.5 below to rename causes.

Weak causal transitions are defined in the usual manner, therefore $A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$ is $A \Longrightarrow \overset{\mu}{\underset{K;k}{\longrightarrow}} \Longrightarrow A'$. The definition of well-sortedness for causal terms is given as for standard processes (all input and output prefixes in a term must respect the discipline imposed by the chosen sorting). All causal terms considered in the chapter are supposed to be well-sorted.

5.3.2 Basic properties of the transitional semantics of causal terms

A *cause substitution* ρ is a function from $\mathcal{K}$ to $\mathcal{K}$. We write $k\rho = k'$ if ρ maps cause k onto k'; and $K\rho$ for the set of causes $\{k\rho \ : \ k \in K\}$. Also, $A\rho$ is the process obtained from A by replacing each of its causes k with $k\rho$. The cause substitution that maps k onto k_0 and is the identity elsewhere, will be sometimes written as a replacement $[k \rightsquigarrow k_0]$.

Lemma 5.3.5 allows us to rename the cause k of a transitions $A \overset{\mu}{\underset{K;k}{\longrightarrow}} A'$. Later, Lemma 5.3.8 will extend this result to weak transitions. Lemmata 5.3.6 and 5.3.7 relate the actions of causal terms A and $A\rho$.

Lemma 5.3.5 *Let $k, k_0 \in \mathcal{K}$, with $k_0 \notin \mathcal{K}(A)$.*

1. *If $A \overset{\mu}{\underset{K;k}{\longrightarrow}} A'$, then $A \overset{\mu}{\underset{K;k_0}{\longrightarrow}} A''$ and $A''[k_0 \rightsquigarrow k] = A'$.*
2. *If $A \overset{\mu}{\underset{K;k_0}{\longrightarrow}} A'$, then $A \overset{\mu}{\underset{K;k}{\longrightarrow}} A''$ and $A'[k_0 \rightsquigarrow k] = A''$.*

PROOF: Easy transition inductions. □

Lemma 5.3.6 *Let ρ be a cause substitution. Then*

1. *$A \overset{\mu}{\underset{K;k}{\longrightarrow}} A'$ implies $A\rho \overset{\mu}{\underset{K\rho;k\rho}{\longrightarrow}} A'\rho$.*
2. *$A \overset{\tau}{\longrightarrow} A'$ implies $A\rho \overset{\tau}{\longrightarrow} A'\rho$.*
3. *$A \Longrightarrow A'$ implies $A\rho \Longrightarrow A'\rho$.*
4. *$A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$ implies $A\rho \overset{\mu}{\underset{K\rho;k\rho}{\Longrightarrow}} A'\rho$.*

PROOF: (1) is proved by transition induction; (2) is also proved by transition induction, but in the case when `C-Com` is the last rule applied, we also need part (1) of this lemma and Lemma 5.3.5. Assertion (3) is proven by iterating (2) and finally (4) is a consequence of (1) and (3). □

Lemma 5.3.7 *Let ρ be a cause substitution. Then*

1. *If $A\rho \overset{\mu}{\underset{K;k}{\longrightarrow}} A'$ then there are K', k' and A'' s.t. $A \overset{\mu}{\underset{K';k'}{\longrightarrow}} A''$ with $K'\rho = K$, $k'\rho = k$ and $A''\rho = A'$.*

2. *If* $A\rho \xrightarrow{\tau} A'$ *then there is* A'' *s.t.* $A \xrightarrow{\tau} A''$ *with* $A''\rho = A'$*.*

3. *If* $A\rho \Longrightarrow A'$ *then there is* A'' *s.t.* $A \Longrightarrow A''$ *with* $A''\rho = A'$*.*

4. *If* $A\rho \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$ *then there are* K'*,* k' *and* A'' *s.t.* $A \overset{\mu}{\underset{K';k'}{\Longrightarrow}} A''$ *with* $K'\rho = K$*,* $k'\rho = k$ *and* $A''\rho = A'$*.*

Proof: Similar to that of Lemma 5.3.6. □

By combining Lemmata 5.3.5, 5.3.6(3) and 5.3.7(3), we obtain the analogue of Lemma 5.3.5 for weak transitions:

Lemma 5.3.8 *Let* $k, k_0 \in \mathcal{K}$*, with* $k_0 \notin \mathcal{K}(A)$*.*

1. *If* $A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$*, then* $A \overset{\mu}{\underset{K;k_0}{\Longrightarrow}} A''$ *and* $A''[k_0 \rightsquigarrow k] = A'$*.*
2. *If* $A \overset{\mu}{\underset{K;k_0}{\Longrightarrow}} A'$*, then* $A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A''$ *and* $A'[k_0 \rightsquigarrow k] = A''$*.* □

5.3.3 Causal bisimulation

Weak causal bisimulation is defined on top of weak causal transitions by means of familiar bisimulation techniques. We shall omit the adjective "weak" and simply call it *causal bisimulation.*

Definition 5.3.9 (causal bisimulation) *A symmetric relation* $\mathcal{R} \subseteq \mathcal{P}_c(\mathcal{N},\mathcal{K}) \times \mathcal{P}_c(\mathcal{N},\mathcal{K})$ *is a* causal bisimulation *(or* $\approx_c$-bisimulation*) if* $A \,\mathcal{R}\, B$ *implies:*

1. *whenever* $A \Longrightarrow A'$ *there exists* B *s.t.* $B \Longrightarrow B'$ *and* $A' \,\mathcal{R}\, B'$*;*
2. *whenever* $A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$ *with* $k \notin \mathcal{K}(A, B)$*, there exists* B' *s.t.* $B \overset{\mu}{\underset{K;k}{\Longrightarrow}} B'$ *and* $A' \,\mathcal{R}\, B'$*.*

Two causal terms A, B *are* causally bisimilar*, written* $A \approx_c B$*, if* $A \,\mathcal{R}\, B$ *for some causal bisimulation* $\mathcal{R}$ *.* □

Lemma 5.3.10 *For any cause substitution* ρ*, if* $A \approx_c B$ *then* $A\rho \approx_c B\rho$*.*

Proof: We show that

$$\mathcal{R} = \{(A\rho, B\rho) \ : \ \rho \text{ is a cause substitution and } A \approx_c B\}$$

is a $\approx_c$-bisimulation. Suppose $A \approx_c B$, and consider clause (2) of Definition 5.3.9. If

$$A\rho \overset{\mu}{\underset{K;k}{\Longrightarrow}} A' \text{ with } k \notin \mathcal{K}(A\rho) \tag{5.1}$$

then from Lemma 5.3.7, for some K', k' and A'', we have

$$A \overset{\mu}{\underset{K';k'}{\Longrightarrow}} A'' \quad \text{with } A''\rho = A',\ K'\rho = K \text{ and } k'\rho = k. \tag{5.2}$$

Let $k_0 \notin \mathcal{K}(A, B)$; by Lemma 5.3.8,

$$A \overset{\mu}{\underset{K';\, k_0}{\Longrightarrow}} A''' \quad \text{with } A'''[k_0 \rightsquigarrow k'] = A''$$

and, since $A \approx_{\mathrm{c}} B$, also

$$B \overset{\mu}{\underset{K';\, k_0}{\Longrightarrow}} B''' \quad \text{with } A''' \approx_{\mathrm{c}} B''' .$$

Reversing, the steps, we get

$$B \overset{\mu}{\underset{K';\, k'}{\Longrightarrow}} B'' \quad \text{with } B'''[k_0 \rightsquigarrow k'] = B''$$

and

$$B\rho \overset{\mu}{\underset{K'';\, k''}{\Longrightarrow}} B' \quad \text{with } K'' = K'\rho,\ k'' = k'\rho \text{ and } B' = B''\rho . \tag{5.3}$$

The action in (5.3) matches the one in (5.1): the conditions in (5.2) and (5.3) imply $K = K''$ and $k'' = k$; moreover, it holds that $B' = B'''[k_0 \rightsquigarrow k']\rho$, $A' = A'''[k_0 \rightsquigarrow k']\rho$ and $A''' \approx_{\mathrm{c}} B'''$. We conclude that $(A', B') \in \mathcal{R}$.

The other clause of the definition of $\approx_{\mathrm{c}}$ can be dealt with similarly. $\square$

The following is a characterisation of causal bisimulation which will be exploited in the proof of completeness for the encoding. The difference from Definition 5.3.9 is that the requirement $k \notin \mathcal{K}(A, B)$ in clause (2), input case, is dropped.

Definition 5.3.11 *A symmetric relation* $\mathcal{R} \subseteq \mathcal{P}_{\mathrm{c}}(\mathcal{N}, \mathcal{K}) \times \mathcal{P}_{\mathrm{c}}(\mathcal{N}, \mathcal{K})$ *is a* loose causal bisimulation *(or* $\approx'_{\mathrm{c}}$-bisimulation*) if* $A \mathcal{R} B$ *implies:*

1. *whenever* $A \Longrightarrow A'$ *there exists* B *s.t.* $B \Longrightarrow B'$ *and* $(A', B') \in \mathcal{R}$*;*
2. *whenever* $A \overset{a\langle\widetilde{b}\rangle}{\underset{K;\, k}{\Longrightarrow}} A'$ *there exists* B' *s.t.* $B \overset{a\langle\widetilde{b}\rangle}{\underset{K;\, k}{\Longrightarrow}} B'$ *and* $(A', B') \in \mathcal{R}$*;*
3. *whenever* $A \overset{\nu\, \widetilde{b}\, \overline{a}\langle\widetilde{b}\rangle}{\underset{K;\, k}{\Longrightarrow}} A'$ *with* $k \notin \mathcal{K}(A, B)$*, there exists* B' *s.t.* $B \overset{\nu\, \widetilde{b}\, \overline{a}\langle\widetilde{b}\rangle}{\underset{K;\, k}{\Longrightarrow}} B'$ *and* $(A', B') \in \mathcal{R}$*.*

Two causal terms A *and* B *are* loose cause bisimilar*, written* $A \approx'_{\mathrm{c}} B$*, if* $A \mathcal{R} B$*, for some loose causal bisimulation* $\mathcal{R}$*.*

Proposition 5.3.12 *The relations* $\approx_{\mathrm{c}}$ *and* $\approx'_{\mathrm{c}}$ *coincide.*

Proof: We only have to show that $\approx_{\mathrm{c}} \subseteq \approx'_{\mathrm{c}}$; the converse follows immediately because the defining clauses for $\approx'_{\mathrm{c}}$ are equal or stronger (in the input case) than those for $\approx_{\mathrm{c}}$.

We proof that $\approx_{\mathrm{c}}$ is a $\approx'_{\mathrm{c}}$-bisimulation. Let $A \approx_{\mathrm{c}} B$. We check clause (2) of Definition 5.3.11. Suppose $A \overset{a\langle\widetilde{b}\rangle}{\underset{K;\, k}{\Longrightarrow}} A'$, and take a fresh cause k_0. Then from Lemma 5.3.8, letting $\mu = a\langle\widetilde{b}\rangle$,

$$A \overset{\mu}{\underset{K;\, k_0}{\Longrightarrow}} A'', \quad \text{with } A''[k_0 \rightsquigarrow k] = A',$$

which implies, since $A \approx_c B$,

$$B \underset{K;k_0}{\overset{\mu}{\Longrightarrow}} B'', \quad \text{with } A'' \approx_c B''.$$

Again, by Lemma 5.3.8, we get

$$B \underset{K;k}{\overset{\mu}{\Longrightarrow}} B', \quad \text{with } B''[k_0 \rightsquigarrow k] = B'.$$

From Lemma 5.3.10, $A'' \approx_c B''$ implies $A' = A''[k_0 \rightsquigarrow k] \approx_c B''[k_0 \rightsquigarrow k] = B'$, which concludes the case. $\square$

5.4 The Encoding

In this section we present and discuss the encoding of causal bisimulation into observation equivalence.

We use two new sorts $\mathcal{C}$ and $\mathcal{T}$ of names, 'new' meaning that these sorts do not occur in the original sorting of causal terms. There will be a one-to-one correspondence between causes — i.e. elements of the set $\mathcal{K}$ — in a causal process, and names of sort $\mathcal{C}$ in the standard process encoding it. Hence, for notational convenience, we identify the sets $\mathcal{C}$ and $\mathcal{K}$ (any ambiguity is resolved by the different classes of processes in which elements of these sets occur as causes and as names, and by the different use of causes and names), and call a name in $\mathcal{C}$ a *cause*, or also a *cause name*. Names in $\mathcal{C}$ carry names of sort $\mathcal{T}$. We sometimes call a name in $\mathcal{T}$ a *token*.

The intuition underlying the encoding is as follows. Causes in a causal term A are encoded in the standard term $[\![A]\!]$ as standard processes called *wires*. A wire $k \triangleright K$ takes a token in input at the *entrance point* k and emits it, as an output, at each *end-point* $k' \in K$. Two wires are connected when an end-point of the first is the entrance-point of the other; this connection represents union of causes. When $[\![A]\!]$ performs a visible action, a name k is emitted and a wire with entrance-point k is created. Intuitively, k represents the unique cause associated to that transition, whose cause-set is represented by all end-points of all wires reachable from k. An observer that receives k can determine this cause-set by sending a token at k and and observing *where* (i.e., at which end-points) the token can be retrieved.

As an example, the encoding of the causal process $K :: a.b.\mathbf{0}$ will have the following transitions:

$$[\![K :: a.b.\mathbf{0}]\!] \xrightarrow{a\langle k\rangle} k \triangleright K \mid [\![k :: b.\mathbf{0}]\!] \xrightarrow{b\langle h\rangle} k \triangleright K \mid h \triangleright k \mid [\![h :: \mathbf{0}]\!].$$

(This shows how *linear* chains are generated.) A crucial point is the modeling of silent transitions. The "merging of causes" discussed at the beginning of Section 5.3 is implemented as an exchange of a bound name, by which the appropriate wires get connected. As an example, we have:

$$[\![K :: a.P \mid H :: \overline{a}.Q]\!] \xrightarrow{\tau} \nu\, k\, (\, k \triangleright K \mid k \triangleright H \mid [\![k :: P \mid k :: Q]\!])$$

(This shows how *branching* chains arise.) After the communication, P and Q have a common cause-set $K \cup H$, referred to by k. Indeed, the derivative of the above transition will turn out to be observationally equivalent to $[\![K \cup H :: P \mid K \cup H :: Q]\!]$.

We come to the actual definition of wire processes. In the sequel, we use v to range over $\mathcal{T}$, and we abbreviate $\mathcal{N} \cup \mathcal{C} \cup \mathcal{T}$ as $\mathcal{N}^+$.

Definition 5.4.1 (Wire processes) *The standard process* $k \rhd K \in \mathcal{P}_3(\mathcal{N}^+)$, *called* wire process, *is so defined:*

$$k \rhd K \stackrel{def}{=} !\, k(v).\, \Pi_{k' \in K}\, !\, \overline{k'}\langle v \rangle\,.$$

Name k is the entrance point *of the wire, and names in K are the* end-points.

We explain the presence of the replication operators in wire processes. In general, once a wire is created, an unbounded number of other wires may get connected to it, both at its entrance point, and — due to the branching structure of causal chains — at each of its end-points. The outermost replication in the definition of $k \rhd K$ is to make the wire *persistent*, so that it can be used an arbitrary number of times. The innermost replications in the definition of $k \rhd K$ make a token availability at an end-point persistent. This ensures that a token which has reached an end-point be transmitted to all unboundedly many wires connected with it.

The encoding is presented in Table 5.2. It acts as a homomorphism, from $\mathcal{P}_{\rm c}(\mathcal{N}, \mathcal{K})$ to $\mathcal{P}_3(\mathcal{N}^+)$, homomorphism everywhere but on prefixes. The encoding adds adds extra components (the wires) into processes, thus increasing the number of their states. The problem, however, is strongly alleviated by Lemmata 5.6.1 and 5.6.2 and the Cancelation Lemma 5.7.4, in Section 5.5, which allow us to get rid of wires when they appear at the outermost level of processes.

The encoding has to be defined on sortings too, since an extra component (of sort $\mathcal{C}$) is added to the arities of names in the target processes. To ease the notation, we abbreviate $k \rhd (K \cup H)$ as $k \rhd (K, H)$, and $k \rhd (\{k'\}, K)$ as $k \rhd (k', K)$. Similarly, we abbreviate $[\![A]\!](K \cup K')$ as $[\![A]\!](K, K')$ and $[\![A]\!](\{k\}, K')$ as $[\![A]\!](k, K')$. Note that when the source term are 0-adic terms (i.e., CCS terms), Table 5.2 gives us an encoding of the classical causal bisimulation of CCS [DD89, Kie91] into the observation equivalence of the monadic π-calculus. It is straightforward to see that there is agreement between the definitions of the encoding on processes and on sortings:

Proposition 5.4.2 *If A is well-sorted for Ob then $[\![A]\!]$ is well-sorted for $[\![Ob]\!]$.* $\square$

5.5 Fundamental Lemmata

All processes in this section are standard processes from $\mathcal{P}_3(\mathcal{N}^+)$.

Encoding of the sorting:

$$[\![Ob]\!] \stackrel{def}{=} \{s \mapsto (\widetilde{s}, \mathcal{C}) \;:\; s \mapsto (\widetilde{s}) \in Ob\} \;\bigcup$$
$$\{\mathcal{C} \mapsto (\mathcal{T}),\; \mathcal{T} \mapsto ()\}$$

where $\mathcal{C}$ and $\mathcal{T}$ are new sorts, i.e. sorts which do not appear in Ob.

....

Encoding of causal processes:

We assume $k : \mathcal{C}$, $K \subseteq_{\text{fin}} \mathcal{C}$ and $v : \mathcal{T}$; we set $[\![A]\!] \stackrel{def}{=} [\![A]\!]\emptyset$, where $[\![A]\!]K$ is defined by induction on the structure of A as follows:

$$\begin{array}{rcl} [\![a(\widetilde{b}).\, A]\!]K & \stackrel{def}{=} & a(\widetilde{b}k).\,(\, k \rhd K \;\mid [\![A]\!]\{k\}),\, k \in \mathcal{K} - K \\ [\![\overline{a}\langle\widetilde{b}\rangle.\, A]\!]K & \stackrel{def}{=} & \nu\, k\, \overline{a}\langle\widetilde{b}k\rangle.\,(\, k \rhd K \;\mid [\![A]\!]\{k\}),\, k \in \mathcal{K} - K \end{array}$$

$$\begin{array}{rclrcl} [\![K' :: A]\!]K & \stackrel{def}{=} & [\![A]\!](K', K) & [\![\tau.A]\!]K & \stackrel{def}{=} & \tau.[\![A]\!]K \\ [\![A_1 \mid A_2]\!]K & \stackrel{def}{=} & [\![A_1]\!]K \mid [\![A_2]\!]K & [\![A_1 \,+\, A_2]\!]K & \stackrel{def}{=} & [\![A_1]\!]K \,+\, [\![A_2]\!]K \\ [\![\nu\, a\, A]\!]K & \stackrel{def}{=} & \nu\, a\, [\![A]\!]K & [\![\mathbf{0}]\!]K & \stackrel{def}{=} & \mathbf{0} \\ [\![!\, A]\!]K & \stackrel{def}{=} & ![\![A]\!]K & & & \end{array}$$

Table 5.2: The encoding of causal terms

5.5.1 Properties of the replication operator

We recall some basic properties of the replication operator, which will be used several times in this and in the next section.

Lemma 5.5.1

1. $!\,P \mid !\,P \sim^c !\,P$.
2. $!\,\alpha.P \sim^c \alpha.(!\,\alpha.P \mid P)$, *if* $bn(\alpha) \cap fn(\alpha.\,P) = \emptyset$. □

Lemma 5.5.2 asserts the distributivity of restricted replications over parallel composition and replication. It generalizes an original result by Milner [Mil91] to the case in which there is a *finite set* of restricted replications — rather than one only.

Lemma 5.5.2 *Suppose name* k *occurs free in the processes* P_1, P_2, P *and* R_i $(i \in I)$ *only in output subject position. Then*

1. $\nu\, k\,\big(P_1 \mid P_2 \mid \Pi_{i\in I}\,!\,k(v).\,R_i\big) \;\sim^c\; \nu\, k\,\big(P_1 \mid \Pi_{i\in I}\,!\,k(v).\,R_i\big) \mid \nu\, k\,\big(P_2 \mid \Pi_{i\in I}\,!\,k(v).\,R_i\big)$.
2. $\nu\, k\,(\,!\,P \mid \Pi_{i\in I}\,!\,k(v).\,R_i) \;\sim^c\; !\,\nu\, k\,(P \mid \Pi_{i\in I}\,!\,k(v).\,R_i)$.

Proof: Analogous to the proofs of Milner's distributivity laws in [Mil91]: one has to define strong bisimulations that are preserved by name substitutions. Some simplification can be achieved by working "up-to static contexts", as shown in [San94b]. □

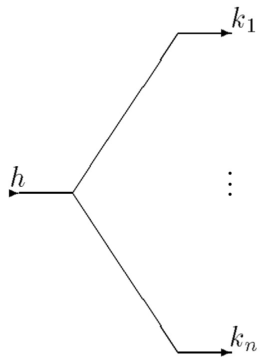
Lemma 5.5.3

1. $\nu\, k\,(\,!\,\alpha.P \mid \Pi_{i\in I}!k(v).R_i) \sim^c\; !\,\alpha.\,(\nu\, k\,(P \mid \Pi_{i\in I}\,!\,k(v).\,R_i))$,
   if $k \notin n(\alpha)$, $bn(\alpha) \cap fn(\Pi_{i\in I}\,!\,k(v).\,R_i) = \emptyset$, *and* k *occurs free in* P *and* R_i $(i \in I)$ *only in output subject position;*
2. $\nu\, k\,(\Pi_{i\in I}!k(v).\Pi_{k'\in K_i}\,!\,\overline{k'}\langle v\rangle \mid !\,\overline{k}\langle v\rangle) \gtrsim \Pi_{k'\in \cup_{i\in I}K_i}\,!\,\overline{k'}\langle v\rangle$, *if* $k \notin \cup_{i\in I}K_i$.

Proof: (1) is proved by first using Lemma 5.5.2(2), to distribute the restricted replications $\Pi_{i\in I}!k(v).R_i$ over the replication $!\,\alpha.P$, and then using expansion and restriction laws (Table 2.2), in the order; (2) is proved by exhibiting the appropriate expansion relation. □

5.5.2 Properties of wire-processes

Wire processes $k \rhd K$ are the core of our implementation of causal bisimulation. To prove the correctness of the implementation, we first need to establish a few properties of compositions of such wires (Lemmata 5.5.4 and 5.5.6). To begin with, two wires $k' \rhd (k, K')$ and $k \rhd K$, in which the entrance point of the second is among the end-points



The wire process $h \rhd K$, with $K = \{k_1, \ldots, k_n\}$.

Figure 5.1: Graphical representation of wire processes (I)

of the first, are, so to speak, 'connected': hence the end-points K of the second can be added to the end-points K' of the first. This is, roughly, the content of Lemma 5.5.4.

If in addition to the above hypothesis, we assume that the common extreme k of the two wires is inaccessible, then their composition yields a single wire $k' \rhd (K, K')$, whose end-points include all end-points of the given wires, except the inaccessible k. This is, roughly, the content of Lemma 5.5.6 (the lemma is slightly more general, in that it allows a product $\Pi_{i \in I}\, k \rhd K_i$ of wires in place of the single wire $k \rhd K$.)

Figure 5.1 contains a graphical representation of a wire process. The content of Lemmata 5.5.4 and 5.5.6 is depicted in Figures 5.2 and 5.3, respectively.

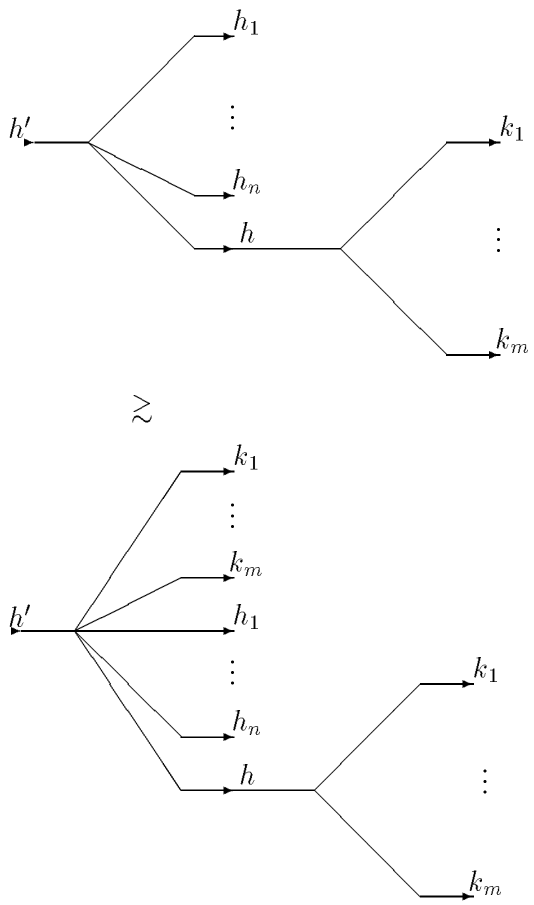
Lemma 5.5.4 $k' \rhd (k, K') \mid k \rhd K \;\; \gtrsim \;\; k' \rhd (k, K', K) \mid k \rhd K$.

Proof: Fixed K, K', k and k', let

$$\begin{aligned} P &\stackrel{def}{=} k \rhd K \mid k' \rhd (k, K', K) \\ Q &\stackrel{def}{=} k \rhd K \mid k' \rhd (k, K') \;. \end{aligned}$$

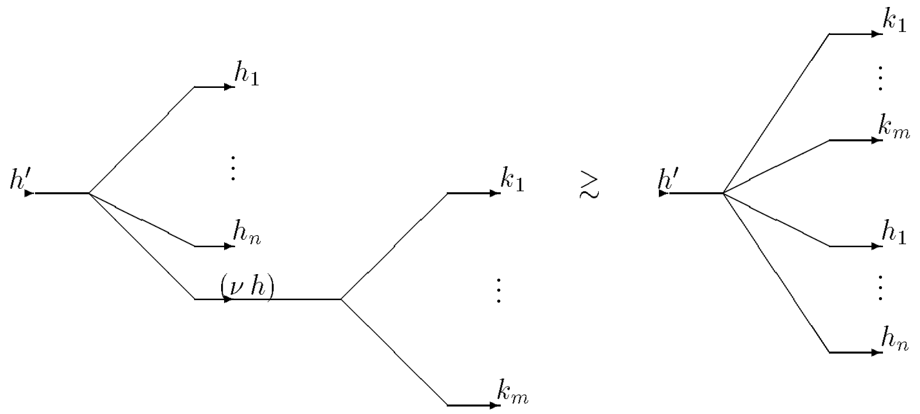
We prove that the singleton $\{(P, Q)\}$ is an expansion up to context and up to $\sim$ (This technique is just a particular case of the expansion up to context and up to $\gtrsim$ (Definition 5.2.5), since $\sim \subset \gtrsim$). We show how the moves of P are matched by Q; the converse is similar. The only non-trivial case is when the move of P originates from $k' \rhd (k, K', K)$, as follows:

$$P \stackrel{k'\langle w\rangle}{\longrightarrow} \sim \Pi_{k'' \in \{k\} \cup K' \cup K} \,!\, \overline{k''}\langle w\rangle \mid P\,.$$



The wire equation $h' \rhd (H, h) \mid h \rhd K \;\gtrsim\; h' \rhd (K, H, h) \mid h \rhd K$, with $H = \{h_1, \ldots, h_n\}$ and $K = \{k_1, \ldots, k_m\}$.

Figure 5.2: Graphical representation of wire processes (II)



The wire equation $(\nu\, h)(\ h' \triangleright (H, h) \mid h \triangleright K\ ) \gtrsim\ h' \triangleright (K, H)$, with $H = \{h_1, \ldots, h_n\}$, $K = \{k_1, \ldots, k_m\}$ and $h \notin K \cup K$.

Figure 5.3: Graphical representation of wire processes (III)

This can be matched by Q thus:

$$\begin{array}{rll} Q & \xrightarrow{k'\langle w\rangle}\sim & k \rhd K \mid \Pi_{k''\in\{k\}\cup K'} \,!\,\overline{k''}\langle w\rangle \mid\ k' \rhd (k, K') \\ & \xrightarrow{\tau}\sim & \Pi_{k''\in K} \,!\,\overline{k''}\langle w\rangle \mid \Pi_{k''\in\{k\}\cup K'} \,!\,\overline{k''}\langle w\rangle \mid Q \\ & \sim & \Pi_{k''\in\{k\}\cup K'\cup K} \,!\,\overline{k''}\langle w\rangle \mid Q \end{array}$$

where the last step, in the case when $K \cap (\{k\} \cup K') \neq \emptyset$, Lemma 5.5.1 is needed. This is enough because $\{(P, Q)\}$ is an expansion up-to context and up-to $\sim^c$. □

To prove Lemma 5.5.6, we use the following lemma:

Lemma 5.5.5

1. *$\nu\, k\,(\,!\,\alpha.P \mid \Pi_{i\in I}\, k \rhd K_i\,) \sim^c\ !\,\alpha.(\nu\, k\,(P \mid \Pi_{i\in I}\, k \rhd K_i\,))$, if $k \notin n(\alpha)$ and k can occur free in P only in output subject position;*

2. *$\nu\, k\,(\Pi_{i\in I}\, k \rhd K_i\ \mid !\overline{k}\langle v\rangle) \gtrsim \Pi_{k'\in\cup_{i\in I}K_i} \,!\,\overline{k'}\langle v\rangle$, if $k \notin \cup_{i\in I}K_i$.*

Proof: The clauses of this lemma are instances of clauses (1) and (2), respectively, of Lemma 5.5.3. □

Lemma 5.5.6 *If $k \notin \big(\{k'\} \cup K' \cup (\cup_{i\in I}K_i)\big)$, then*

$$\nu\, k\,\big(\ k' \rhd (K', k)\ \mid \Pi_{i\in I}\, k \rhd K_i\ \big) \gtrsim\ k' \rhd (K', \cup_{i\in I}K_i)\ .$$

Proof: Let $Q \stackrel{def}{=} \nu\, k\,\big(\ k' \rhd (K', k)\ \mid \Pi_{i\in I}\, k \rhd K_i\ \big)$, $P \stackrel{def}{=}\ k' \rhd (K', \cup_{i\in I}K_i)$ and $R \stackrel{def}{=} \Pi_{k''\in K'} \,!\,\overline{k''}\langle v\rangle \mid\, !\,\overline{k}\langle v\rangle$. Then, recalling that $k' \rhd (K', k)\ = !\,k'(v).R$, we have:

$$\begin{array}{rll} Q & \sim^c & !\,k'(v).\nu\, k\,(R \mid \Pi_{i\in I}\, k \rhd K_i\,) \\ & & \quad\text{Lemma 5.5.5 (1)} \\ & \sim^c & !\,k'(v).\big(\Pi_{k''\in K'} \,!\,\overline{k''}\langle v\rangle \mid \nu\, k\,(\,!\,\overline{k}\langle v\rangle \mid \Pi_{i\in I}\, k \rhd K_i\,)\big) \\ & & \quad\text{Laws for restriction in Table 2.2} \\ & \gtrsim & !\,k'(v).\big(\Pi_{k''\in K'} \,!\,\overline{k''}\langle v\rangle \mid \Pi_{k''\in\cup_{i\in I}K_i} !\overline{k''}\langle v\rangle)\big) \\ & & \quad\text{Lemma 5.5.5 (2)} \\ & \sim & P \end{array}$$

where in the last step, in the case when $K' \cap (\cup_{i\in I}K_i) \neq \emptyset$, Lemma 5.5.1 is needed. □

5.6 Operational Correspondences

This section is devoted to showing the close operational correspondence between the actions of a causal process A and those of the encoding standard process $[\![A]\!]$. In subsection 5.6.1 we study the operational correspondence on strong visible transitions (Proposition 5.6.7), and on strong silent transitions (Proposition 5.6.8). These results are used in subsection 5.6.2 to establish the correspondences on weak transitions.

5.6.1 Operational correspondence on strong transitions

In the proof of the operational correspondence between A and $[\![A]\!]$ on strong transitions, we use Lemma 5.6.1 and Corollary 5.6.6. To prove the latter result, we rely on Lemmata 5.6.2 through 5.6.5. Lemma 5.6.1 and Lemma 5.6.2 are consequences of Lemma 5.5.4 and Lemma 5.5.6, dealing with compositions of wires. To see the relationship between Lemma 5.6.1 and Lemma 5.5.4, and between Lemma 5.6.2 and Lemma 5.5.6, remember that the parameter (K_0, k) is used in $[\![A]\!](K_0, k)$ to define wires of the form $k \rhd K'$, for some K'. Lemmata 5.6.1 and 5.6.2 are also useful when reasoning about processes returned by the encoding, in that they avoid us having to 'unfold' the definition of wire processes. The proofs of Lemmata 5.6.1 and 5.6.2 are reported in Appendix A.

Lemma 5.6.1 $[\![A]\!](K_0, k) \mid k \rhd K \;\gtrsim\; [\![A]\!](k, K_0, K) \mid k \rhd K$ $\square$

Lemma 5.6.2 *If $k \notin K_0 \cup \big(\cup_{i \in I} K_i\big)$, then*

$$\nu\, k\,\big([\![A]\!](K_0, k) \mid \Pi_{i \in I}\, k \rhd K_i\,\big) \;\gtrsim\; [\![A]\!](K_0, \cup_i K_i) \qquad \square$$

The next three lemmata are used to prove Corollary 5.6.6. We have decomposed the corollary in this way to help understanding its meaning. (The corollary can also be proved directly, using Lemma 5.6.2 and transition induction; all cases of the induction would be fairly simple.) Lemma 5.6.3 is a variation of Lemma 5.6.2. The intuition of Lemma 5.6.4 is the following: any cause in $[\![A]\!]K_0$ depends on K_0 (in the sense that there is a wire-connection from this cause to the causes in K_0); therefore, in the process $\nu\, k\big([\![A]\!]K_0 \mid \Pi_{i \in I}\; k \rhd (K_i, K_0)\;\big)$, the dependency of k on K_0, expressed in the wires $k \rhd (K_0, K_i)$, is superfluous. In Lemma 5.6.5, the transition $A \;\xrightarrow[K;\,k]{\mu}\; A'$ shows that k depends on K in A': therefore, the wire $k \rhd K$ can be safely removed. The first two lemmata are easily proved by exhibiting appropriate expansion relations, in the same style as Lemma 5.6.2. Lemma 5.6.5 is proved by transition induction on $A \;\xrightarrow[K;\,k]{\mu}\; A'$.

Lemma 5.6.3 *If $k \notin K \cup K_0$, then $\nu\, k\,([\![A]\!]K_0 \mid k \rhd K\,) \gtrsim [\![A[k \rightsquigarrow K]]\!]K_0$.* $\square$

Lemma 5.6.4 *If $k \notin (\cup_{i \in I} K_i) \cup K_0$, then*

$$\nu\, k\,\big([\![A]\!]K_0 \mid \Pi_{i \in I}\, k \rhd (K_i, K_0)\,\big) \;\gtrsim\; \nu\, k\,\big([\![A]\!]K_0 \mid \Pi_{i \in I}\, k \rhd K_i\,\big)\,. \qquad \square$$

Lemma 5.6.5 *If $A \;\xrightarrow[K;\,k]{\mu}\; A'$ with $k \notin \mathcal{K}(A)$, and $k \notin K_0 \cup K'$, then*

$$\nu\, k\,\big([\![A']\!]K_0 \mid k \rhd K \mid k \rhd K'\,\big) \gtrsim \nu\, k\,\big([\![A']\!]K_0 \mid k \rhd K'\,\big)\,. \qquad \square$$

Corollary 5.6.6 *If $A \;\xrightarrow[K;\,k]{\mu}\; A'$ with $k \notin \mathcal{K}(A)$, and $k \notin K_0 \cup K'$, then*

$$\nu\, k\,\big([\![A']\!]K_0 \mid k \rhd (K_0, K) \mid k \rhd (K_0, K')\,\big) \gtrsim [\![A'[k \rightsquigarrow K']]\!]K_0\,.$$

PROOF: Using Lemmata 5.6.4, 5.6.5 and 5.6.3, in the order. □

Propositions 5.6.7 and 5.6.8 below are slightly stronger than we shall need. They relate the transitions of a causal process A with those of a standard process $[\![A]\!]K$, for a generic parameter K, while we shall only need the results for $K=\emptyset$. These more general statements are easier to handle in the inductive proofs (especially when dealing with rule `C-Cau`).

Proposition 5.6.7 (Correspondence on strong visible transitions)

1. (a) *If* $A \xrightarrow[H;k]{a\langle\widetilde{b}\rangle} A'$ *then* $[\![A]\!]K \xrightarrow{a\langle\widetilde{bk}\rangle} \gtrsim k \rhd (H,K) \mid [\![A']\!]K$.

   (b) *If* $k \notin \mathcal{K}(A) \cup K$ *and* $A \xrightarrow[H;k]{\nu\,\widetilde{b'}\,\overline{a}\langle\widetilde{b}\rangle} A'$ *then* $[\![A]\!]K \xrightarrow{(\nu\,\widetilde{b'}k)\overline{a}\langle\widetilde{bk}\rangle} \gtrsim k \rhd (H,K) \mid [\![A']\!]K$.

2. *The converse of (1), on the actions from* $[\![A]\!]K$:

   (a) *If* $[\![A]\!]K \xrightarrow{a\langle\widetilde{bk}\rangle} P$ *then there exist* A', H *s.t.* $A \xrightarrow[H;k]{a\langle\widetilde{b}\rangle} A'$ *and* $P \gtrsim k \rhd (H,K) \mid [\![A']\!]K$.

   (b) *If* $[\![A]\!]K \xrightarrow{(\nu\,\widetilde{b'}k)\overline{a}\langle\widetilde{bk}\rangle} P$ *then there exist* A', H *s.t.* $A \xrightarrow[H;k]{\nu\,\widetilde{b'}\,\overline{a}\langle\widetilde{b}\rangle} A'$ *and* $P \gtrsim k \rhd (H,K) \mid [\![A']\!]K$.

PROOF: The proof exploits Lemma 5.6.1. We only deal with case (1.a), since the others are similar or easier. We proceed by induction on the length of the derivation of $A \xrightarrow[H;k]{a\langle\widetilde{b}\rangle} A'$.

Case 1: `C-Inp` rule.

(This is the base case.) Then we have $A = a(\widetilde{c}).\,P \xrightarrow[\emptyset\,;\,k]{a\langle\widetilde{b}\rangle} k :: P\{\widetilde{b}/\widetilde{c}\}$. Therefore

$$\begin{array}{rclcl} [\![A]\!]K & = & a(\widetilde{c}k).\,(\,k \rhd K \mid [\![P]\!]\{k\}) & \xrightarrow{a\langle\widetilde{bk}\rangle} & k \rhd K \mid [\![P\{\widetilde{b}/\widetilde{c}\}]\!]\{k\} \\ & \gtrsim & k \rhd K \mid [\![P\{\widetilde{b}/\widetilde{c}\}]\!](k,K) & = & k \rhd K \mid [\![k :: P\{\widetilde{b}/\widetilde{c}\}]\!]K \end{array}$$

where the use of $\gtrsim$ is due to Lemma 5.6.1.

Case 2: `C-Par` rule.

Then $A = A_1 \mid A_2$ and, supposing that the move originates from A_1, the last rule applied is of the form

$$\texttt{C-Par} \;:\; \frac{A_1 \xrightarrow[H;k]{a\langle\widetilde{b}\rangle} A_1'}{A_1 \mid A_2 \xrightarrow[H;k]{a\langle\widetilde{b}\rangle} A_1' \mid A_2}\,.$$

From the inductive hypothesis, we know that for all K there is P s.t.

$$[\![A_1]\!]K \xrightarrow{a\langle\widetilde{bk}\rangle} P \gtrsim k \rhd (H,K) \mid [\![A_1']\!]K\,.$$

From this, using rule `Par` , we get

$$[\![A_1 \mid A_2]\!]K \xrightarrow{a\langle\widetilde{bk}\rangle} P \mid [\![A_2]\!]K \gtrsim k \rhd (H, K) \mid [\![A_1' \mid A_2]\!]K$$

which is the thesis.

Case 3: `C-Cau` rule.

Then $A = K_0 :: A'$. The last rule applied in deriving the transition from A is of the form

$$\texttt{Cau} : \frac{A' \xrightarrow[H';\,k]{a\langle\widetilde{b}\rangle} A''}{A = K_0 :: A' \xrightarrow[H;\,k]{a\langle\widetilde{b}\rangle} K_0 :: A''}$$

where $H = K_0 \cup H'$. From the inductive hypothesis, for each K' there exists some P such that

$$[\![A']\!]K' \xrightarrow{a\langle\widetilde{bk}\rangle} P \gtrsim k \rhd (H', K') \mid [\![A'']\!]K'.$$

Recalling that $[\![A]\!]K = [\![A']\!](K_0, K)$ and letting $K' = K_0 \cup K$ in the above transition, the thesis follows.

Case 4: `C-Out` , `C-Sum` , `C-Res` , `C-Open` or `C-Repl` .

These cases easily follow from the inductive assumption.

□

Proposition 5.6.8 (Correspondence on strong silent transitions)

1. *If $A \xrightarrow{\tau} A'$, then $[\![A]\!]K \xrightarrow{\tau} \gtrsim [\![A']\!]K$.*

2. *If $[\![A]\!]K \xrightarrow{\tau} P$, then there exists A' such that $A \xrightarrow{\tau} A'$ and $P \gtrsim [\![A']\!]K$.*

Proof: We only deal with (1), since (2) is similar. The proof is an induction on the derivation of $A \xrightarrow{\tau} A'$ and exploits the previous proposition (5.6.7), Lemma 5.5.2 and Corollary 5.6.6. The most difficult case is when `C-Com` is the last rule applied:

$$\texttt{C-Com} : \frac{A_1 \xrightarrow[H_1;\,k]{a\langle\widetilde{b}\rangle} A_1', \qquad A_2 \xrightarrow[H_2;\,k]{(\nu\,\widetilde{c})\overline{a}\langle\widetilde{b}\rangle} A_2'}{A = A_1 \mid A_2 \xrightarrow{\tau} \nu\,\widetilde{c}\,(A_1'[k \rightsquigarrow H_2] \mid A_2'[k \rightsquigarrow H_1]) \stackrel{def}{=} A'} \tag{5.4}$$

where $\widetilde{c} \subseteq \widetilde{b}$ and $k \notin \mathcal{K}(A)$. By applying Lemma 5.3.5 to rename k, if needed, we can also suppose $k \notin K$. From Proposition 5.6.7(1), applied to the premises of (5.4) we get

$$\begin{array}{lcl} [\![A_1]\!]K & \xrightarrow{a\langle\widetilde{bk}\rangle} \gtrsim & k \rhd (H_1, K) \mid [\![A_1']\!]K \stackrel{def}{=} R_1 \\ [\![A_2]\!]K & \xrightarrow{(\nu\,ck)\overline{a}\langle\widetilde{bk}\rangle} \gtrsim & k \rhd (H_2, K) \mid [\![A_2']\!]K \stackrel{def}{=} R_2\,. \end{array}$$

From the above two transitions and rule `Com` we infer

$$\begin{array}{rcl} [\![A]\!]K = [\![A_1]\!]K \mid [\![A_2]\!]K \xrightarrow{\tau} & \gtrsim & (\nu\,\widetilde{c}k)(R_1 \mid R_2) \\ & \sim^c & (\nu\,\widetilde{c}k)(\ k \rhd (H_1, K)\ \mid\ k \rhd (H_2, K)\ \mid \\ & & [\![A'_1]\!]K \mid [\![A'_2]\!]K) \\ & \stackrel{def}{=} & P\,. \end{array}$$

Now, k can occur free in $[\![A'_i]\!]K$ $(i \in \{1,2\})$ only in output subject position; thus we can apply Lemma 5.5.2 (1) to distribute the restricted replications $k \rhd (H_1, K) \mid k \rhd (H_2, K)$ over the parallel composition $[\![A'_1]\!]K \mid [\![A'_2]\!]K$ in P, obtaining:

$$\begin{array}{rcl} P & \sim & \nu\,\widetilde{c}\,\Pi_{i=1,2}\big(\nu\, k\ (\ k \rhd (H_1, K)\ \mid\ k \rhd (H_2, K)\ \mid [\![A'_i]\!]K)\big) \\ & \stackrel{def}{=} & \nu\,\widetilde{c}\,(S_1 \mid S_2)\,. \end{array}$$

Now, since $A_i \xrightarrow[H_i;k]{\mu} A'_i$ and $k \notin K \cup H_i \cup \mathcal{K}(A_i)$, $(i \in \{1,2\})$, we can apply Corollary 5.6.6 to S_i, so to replace k with H_j inside $[\![A'_i]\!]$ and eliminate the wires, thus:

$$S_i \gtrsim [\![A'_i[k \leadsto H_j]]\!]K$$

where if $i = 1$ then $j = 2$, and if $i = 2$ then $j = 1$. Since parallel composition and restriction preserve $\gtrsim$, $\nu\,\widetilde{c}\,(S_1 \mid S_2) \gtrsim \nu\,\widetilde{c}\,([\![A'_1[k \leadsto H_2]]\!]K \mid [\![A'_2[k \leadsto H_1]]\!]K) = [\![A']\!]K$ holds. Summarizing, we have obtained that $[\![A]\!]K \xrightarrow{\tau}\ \gtrsim P \gtrsim [\![A']\!]K$. This concludes the case. $\square$

5.6.2 Operational correspondences on weak transitions

In this subsection we study the operational correspondence between encoded and encoding processes on weak transitions.

Proposition 5.6.9 (Correspondence on weak silent transitions)

1. *If* $A \Longrightarrow A'$, *then* $[\![A]\!] \Longrightarrow\ \gtrsim [\![A']\!]$.
2. *If* $[\![A]\!] \Longrightarrow P$, *then there exists* A' *such that* $A \Longrightarrow A'$ *and* $P \gtrsim [\![A']\!]$.

PROOF:

1. It must be $A \xrightarrow{\tau^n} A'$, for some $n \geq 0$. We proceed by induction on n. For $n = 0$ the statement holds trivially. Suppose $n > 0$; then, for some A'', we have $A \xrightarrow{\tau^{n-1}} A'' \xrightarrow{\tau} A'$. From the inductive hypothesis, for some Q,

$$[\![A]\!] \Longrightarrow Q \gtrsim [\![A'']\!]\,.$$

Now, from Proposition 5.6.8(1) and $A'' \xrightarrow{\tau} A'$ we have that, for some P,

$$[\![A'']\!] \xrightarrow{\tau} P \gtrsim [\![A']\!]\,.$$

Since $Q \gtrsim [\![A'']\!]$, from the above transition we get, for some Q',

$$Q \stackrel{\tau}{\Longrightarrow} Q' \gtrsim P\,.$$

From this and $P \gtrsim [\![A']\!]$, we get $Q' \gtrsim [\![A']\!]$. To sum up, we have obtained:

$$[\![A]\!] \Longrightarrow Q \stackrel{\tau}{\Longrightarrow} Q' \gtrsim [\![A']\!]$$

which proves the thesis.

2. For some $n \geq 0$ it holds that $[\![A]\!] \stackrel{\tau^n}{\longrightarrow} P$. Again, we proceed by induction on n. The case $n = 0$ is trivial. If $n > 0$, the for some Q, we have $[\![A]\!] \stackrel{\tau^{n-1}}{\longrightarrow} Q \stackrel{\tau}{\longrightarrow} P$. By the inductive hypothesis, we have, for some A'',

$$A \Longrightarrow A'' \quad \text{with } Q \gtrsim [\![A'']\!]\,.$$

From this and $Q \stackrel{\tau}{\longrightarrow} P$, we deduce that, for some R,

$$[\![A'']\!] \stackrel{\widehat{\tau}}{\longrightarrow} R \quad \text{with } P \gtrsim R\,.$$

Now, by Proposition 5.6.8(2), there is A' s.t.

$$A'' \stackrel{\widehat{\tau}}{\longrightarrow} A' \quad \text{with } R \gtrsim [\![A']\!].$$

Thus, we have found A' s.t. $A \Longrightarrow A'$ and $P \gtrsim [\![A']\!]$, and proved the thesis.

□

Remark 5.6.10 In the proof of item (2) of the above proposition the use of the expansion relation turns out to be necessary to close up the induction of the proof. Had we used $\approx$ in place of $\gtrsim$ in the above proof, from $Q \approx [\![A'']\!]$ and $Q \stackrel{\tau}{\longrightarrow} P$, we could have only inferred $[\![A'']\!] \Longrightarrow R$ (in place of the stronger $[\![A'']\!] \stackrel{\widehat{\tau}}{\longrightarrow} R$); as a consequence, we could not have applied Proposition 5.6.8 to close up the induction. □

Proposition 5.6.11 (Correspondence on weak visible transition)

1. *(a) If* $A \underset{K;k}{\stackrel{a\langle\widetilde{b}\rangle}{\Longrightarrow}} A'$, *then* $[\![A]\!] \stackrel{a\langle\widetilde{b}k\rangle}{\Longrightarrow} \gtrsim k \rhd K \mid [\![A']\!]$.

   (b) If $k \notin \mathcal{K}(A)$ *and* $A \underset{K;k}{\stackrel{(\nu\,\widetilde{b'})\overline{a}\langle\widetilde{b}\rangle}{\Longrightarrow}} A'$, *then* $[\![A]\!] \stackrel{(\nu\,\widetilde{b'}k)\overline{a}\langle\widetilde{b}k\rangle}{\Longrightarrow} \gtrsim k \rhd K \mid [\![A']\!]$.

2. *The converse of (1), on the actions from* $[\![A]\!]$*:*

   (a) If $[\![A]\!] \stackrel{a\langle\widetilde{b}k\rangle}{\Longrightarrow} P$, *then there exist* A' *and* K *s.t.* $A \underset{K;k}{\stackrel{a\langle\widetilde{b}\rangle}{\Longrightarrow}} A'$ *and* $P \gtrsim k \rhd K \mid [\![A']\!]$.

(b) *If* $[\![A]\!] \stackrel{(\nu\,\widetilde{b'}k)\overline{a}\langle\widetilde{b}k\rangle}{\Longrightarrow} P$*, then there exist* A' *and* K *s.t.* $A \underset{K;k}{\stackrel{\nu\,\widetilde{b'}\,\overline{a}\langle\widetilde{b}\rangle}{\Longrightarrow}} A'$ *and* $P \gtrsim k \rhd K \mid [\![A']\!]$.

Proof: We only consider case (1.a); the remaining ones are similar. For some A'' and A''' we have

$$A \Longrightarrow A'' \underset{K;k}{\stackrel{a\langle\widetilde{b}\rangle}{\longrightarrow}} A''' \Longrightarrow A'\,.$$

From Proposition 5.6.9, we have, for some P'',

$$[\![A]\!] \Longrightarrow P'' \gtrsim [\![A'']\!]\,.$$

Furthermore, from Proposition 5.6.7, for some Q,

$$[\![A'']\!] \stackrel{a\langle\widetilde{b}k\rangle}{\longrightarrow} Q \gtrsim k \rhd K \mid [\![A''']\!]\,.$$

Therefore, since $P'' \gtrsim [\![A'']\!]$, there exists P''' such that:

$$P'' \stackrel{a\langle\widetilde{b}k\rangle}{\Longrightarrow} P''' \gtrsim Q\,.$$

Since $A''' \Longrightarrow A'$, by Proposition 5.6.9(1), $[\![A''']\!] \Longrightarrow\ \gtrsim [\![A']\!]$. Then, using rule `Par` , we get, for some Q',

$$k \rhd K \mid [\![A''']\!] \Longrightarrow Q' \gtrsim k \rhd K \mid [\![A']\!] \tag{5.5}$$

We have proved that $P''' \gtrsim Q \gtrsim k \rhd K \mid [\![A''']\!]$. From this and (5.5) we deduce that there is P' s.t.

$$P''' \Longrightarrow P' \gtrsim Q' \gtrsim k \rhd K \mid [\![A']\!]\,.$$

To sum up, we have

$$[\![A]\!] \Longrightarrow P'' \stackrel{a\langle\widetilde{b}k\rangle}{\Longrightarrow} P''' \Longrightarrow P' \gtrsim k \rhd K \mid [\![A']\!]$$

which concludes the case.

□

Remark 5.6.12

1. Suppose $[\![A]\!] \Longrightarrow P$; then P cannot perform actions along a cause name k. This is a consequence of Proposition 5.6.9 (2) and of the fact that the processes returned by the encoding cannot immediately perform actions along a cause name (the latter fact holds by the definition of the encoding).

2. Suppose $[\![A]\!] \stackrel{\mu}{\Longrightarrow} P$; then P cannot perform *output* actions along a cause name k. This is a consequence of Proposition 5.6.11(2) and of the item (1) of this remark. □

5.7 Full Abstraction

In the first part of the section we prove two results about wires processes, the *Cancelation Lemma* and the *Saturation Lemma*, that will be crucial in the second part to prove the full abstraction theorem.

5.7.1 The Cancelation and the Saturation lemmata

The main result of this section is the Cancelation Lemma. It is very useful when verifying observation equivalence between processes returned by the encoding, to cancel wires which appear at the outermost level. The Cancelation Lemma will be used in combination with the *Saturation Lemma* in the proof of soundness for the encoding.

To state the two lemmata, we need two auxiliary notions: *Cause-Sets* of a process and *Saturation*. The cause-sets of a process A contains all sets of causes which can appear in transitions from A; that is, if $A \xrightarrow[K;k]{\mu} A'$, then K is in the cause-sets of A (this property is formalized in Lemma B.1.8 (1), in Appendix B).

Definition 5.7.1 (Cause-Sets of a process) *For a causal process A,* the cause-sets of *A, written $\mathcal{CS}(A)$, is the set of sets of causes inductively defined as follows:*

$$\begin{aligned}
\mathcal{CS}(P) &\stackrel{def}{=} \{\emptyset\} \\
\mathcal{CS}(K :: A) &\stackrel{def}{=} \{K \cup H \,|\, H \in \mathcal{CS}(A)\} \\
\mathcal{CS}(\nu\, a\, A) &\stackrel{def}{=} \mathcal{CS}(A) \\
\mathcal{CS}(A_1 \mid A_2) &\stackrel{def}{=} \mathcal{CS}(A_1) \cup \mathcal{CS}(A_2)\,.
\end{aligned}$$

Note that in general not all sets in $\mathcal{CS}(A)$ need to appear in causal transition from A; e.g., if A is $\nu\, a\,(K :: a.\mathbf{0})$, then $K \in \mathcal{CS}(A)$, but A cannot perform any transition.

Now, the definition of saturation. Intuitively, a causal process A is saturated for a product of wires W if each set of causes H which appear in A is "closed" w.r.t. the causality relation encoded by W; i.e., if a wire in W has its entrance-point in H, then all end-points of the wire are in H.

Definition 5.7.2 (Saturation) *Let A be a causal process and let $W \stackrel{def}{=} \Pi_{i\in I}\, k_i \rhd K_i$. We say that A is* saturated *for W if for each $H \in \mathcal{CS}(A)$ and $i \in I$, $k_i \in H$ implies $K_i \subseteq H$.*

We can now state the Saturation and the Cancelation lemmata; their proofs are reported in Appendix B.

Lemma 5.7.3 (Saturation lemma) *Let A be a causal process saturated for the product of wires $W \stackrel{def}{=} \Pi_{i\in I}\, k_i \rhd K_i$.*

1. *If* $A \xrightarrow[K;k]{\mu} A'$ *with* k *fresh, then* A' *is saturated for* $k \triangleright K \mid W$*.*
2. *If* $A \xrightarrow{\tau} A'$*, then* A' *is saturated for* W*.*
3. *If* $A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$ *with* k *fresh, then* A' *is saturated for* $k \triangleright K \mid W$*.* □

Lemma 5.7.4 (Cancelation lemma) *Let* A *be saturated for* $k \triangleright K$ *and* B *be saturated for* $k \triangleright K'$*. Then*

$$k \triangleright K \mid [\![A]\!] \approx k \triangleright K' \mid [\![B]\!] \text{ implies } K = K' \text{ and } [\![A]\!] \approx [\![B]\!]\,.$$ □

5.7.2 Soundness and completeness of the encoding

Theorem 5.7.5 (Soundness) $[\![A]\!] \approx [\![B]\!]$ *implies* $A \approx_{\mathrm{c}} B$*.*

Proof: We prove that the relation

$$\mathcal{R} = \{(A, B) \ : \ [\![A]\!] \approx [\![B]\!]\}$$

is a $\approx_{\mathrm{c}}$-bisimulation. Let $(A, B) \in \mathcal{R}$. We only show how the moves of A are matched by B, since the relation is symmetrical. We check clauses (1) and (2) of the definition of $\approx_{\mathrm{c}}$.

Clause (1)

$$\begin{array}{rcll} A & \Longrightarrow & A' & \text{implies, from Proposition 5.6.9(1)} \\ [\![A]\!] & \Longrightarrow & P\,, & \text{for some } P \text{ with } P \approx [\![A']\!]\text{. Since } [\![A]\!] \approx [\![B]\!] \\ [\![B]\!] & \Longrightarrow & Q\,, & \text{for some } Q \text{ with } P \approx Q. \\ & & & \text{From Proposition 5.6.9(2) we get} \\ B & \Longrightarrow & B'\,, & \text{for some } B' \text{ with } Q \approx [\![B']\!]\,. \end{array}$$

To sum up, we have $[\![A']\!] \approx P \approx Q \approx [\![B']\!]$. Hence $(A', B') \in \mathcal{R}$.

Clause (2) We only look at the input case; the output case is similar. We suppose that $k \notin \mathcal{K}(A, B)$.

$$\begin{array}{ll} A \overset{a\langle\widetilde{b}\rangle}{\underset{K;k}{\Longrightarrow}} A' & \text{implies, from Proposition 5.6.11(1),} \\ [\![A]\!] \overset{a\langle\widetilde{bk}\rangle}{\Longrightarrow} P\,, & \text{for some } P \text{ with } P \approx k \triangleright K \mid [\![A']\!]\text{. Since } [\![A]\!] \approx [\![B]\!]\text{, we get} \\ [\![B]\!] \overset{a\langle\widetilde{bk}\rangle}{\Longrightarrow} Q\,, & \text{for some } Q \text{ with } Q \approx P. \\ & \text{From Proposition 5.6.11(2), we get} \\ B \overset{a\langle\widetilde{b}\rangle}{\underset{K';k}{\Longrightarrow}} B'\,, & \text{for some } Q \text{ with } Q \approx k \triangleright K' \mid [\![B']\!]. \end{array}$$

We have obtained that $k \triangleright K \mid [\![A']\!] \approx k \triangleright K' \mid [\![B']\!]$. From the Saturation Lemma applied to above two weak causal transitions, it follows that A' is saturated for $k \triangleright K$ and that B' is saturated for $k \triangleright K'$. From the Cancelation Lemma, it follows that $K = K'$ and $[\![A']\!] \approx [\![B']\!]$. Hence $(A', B') \in \mathcal{R}$, and we are done. □

Theorem 5.7.6 (Completeness) *$A \approx_{\mathrm{c}} B$ implies $[\![A]\!] \approx [\![B]\!]$.*

PROOF: We exploit the characterisation of causal bisimulation in terms of loose causal bisimulation ($\approx'_{\mathrm{c}}$) in Proposition 5.3.12 (we need $\approx'_{\mathrm{c}}$ for the treatment of clause (2) below). We prove that the relation

$$\mathcal{R} = \{([\![A]\!], [\![B]\!]) \; : \; A \approx'_{\mathrm{c}} B\}$$

is a weak bisimulation up to $\gtrsim$ and up to context. Let $([\![A]\!], [\![B]\!]) \in \mathcal{R}$. We look at clause (1) and (2) of the definition of observation equivalence, on the moves by $[\![A]\!]$.

Clause (1)

$[\![A]\!] \Longrightarrow P$	implies, from Proposition 5.6.11(2)
$A \Longrightarrow A'$,	for some A' with $P \gtrsim [\![A']\!]$. Since $A \approx'_{\mathrm{c}} B$
$B \Longrightarrow B'$,	for some B' with $A' \approx'_{\mathrm{c}} B'$.
	From Proposition 5.6.11(1) we get
$[\![B]\!] \Longrightarrow Q \gtrsim [\![B']\!]$,	for some Q.

To sum up, we have $[\![A]\!] \Longrightarrow P \gtrsim [\![A']\!]$, $[\![B]\!] \Longrightarrow Q \gtrsim [\![B']\!]$, and $A' \approx'_{\mathrm{c}} B'$. Hence $([\![A']\!], [\![B']\!]) \in \mathcal{R}$, and we are done.

Clause (2) We check only the input case (the output case is similar, only note that in that case, one can suppose w.l.o.g. $k \notin \mathcal{K}(A, B)$):

$[\![A]\!] \overset{a\langle \widetilde{bk}\rangle}{\Longrightarrow} P$	implies, from Proposition 5.6.11(2),
$A \underset{K;k}{\overset{a\langle \tilde{b}\rangle}{\Longrightarrow}} A'$,	for some A' with $P \gtrsim k \rhd K \mid [\![A']\!]$. Since $A \approx'_{\mathrm{c}} B$
$B \underset{K;k}{\overset{a\langle \tilde{b}\rangle}{\Longrightarrow}} B'$,	for some B' with $A' \approx'_{\mathrm{c}} B'$. From Proposition 5.6.11(1)
$[\![B]\!] \overset{a\langle \widetilde{bk}\rangle}{\Longrightarrow} Q$	for some Q' with $Q \gtrsim k \rhd K \mid [\![B']\!]$.

Processes $k \rhd K \mid [\![A']\!]$ and $k \rhd K \mid [\![B']\!]$ allow us to close the bisimulation $\mathcal{R}$, up to $\gtrsim$ and up to the static context $k \rhd K \mid [\,.\,]$.

$\square$

Theorems 5.7.5 and 5.7.6 prove the full abstraction of the encoding. That is, (using also Proposition 5.4.2) causal bisimulation on causal terms in $\mathcal{P}_{\mathrm{c}}(\mathcal{N}, \mathcal{K})$ and which respect a sorting Ob can be reconducted to the ordinary observation equivalence among standard terms in $\mathcal{P}_3(\mathcal{N}^+)$ and which respect a sorting $[\![Ob]\!]$, where $\mathcal{N}^+$ is an appropriate extension of the class of names $\mathcal{N}$.

In particular, this result holds for the standard processes in $\mathcal{P}_{\mathrm{c}}(\mathcal{N}, \mathcal{K})$, i.e., those terms in $\mathcal{P}_{\mathrm{c}}(\mathcal{N}, \mathcal{K})$ which do not contain the causal prefix $K :: A$. Formally, these are the processes in the class $\mathcal{P}_3(\mathcal{N})$. The applicability of the encoding to standard processes was the result we were most interested in.

Corollary 5.7.7 (Full abstraction of the encoding on standard processes) *For all P and $Q \in \mathcal{P}_3(\mathcal{N})$, we have $[\![P]\!], [\![Q]\!] \in \mathcal{P}_3(\mathcal{N}^+)$ and $P \approx_c Q$ iff $[\![P]\!] \approx [\![Q]\!]$.* □

5.8 Examples of Applications

In this section we give some examples of how the encoding and the full abstraction theorems can be used to reason about causal bisimulation.

Some simple laws for causal bisimulation follow directly from the definition of the encoding; for example:

$$\begin{array}{rcl} K :: (A \mid B) & \approx_c & (K :: A) \mid (K :: B)\,, \\ K :: H :: A & \approx_c & K \cup H :: A\,, \\ \emptyset :: A & \approx_c & A\,. \end{array}$$

As an easy corollary of the definition of our encoding and its full abstraction, we can derive the congruence of $\approx_c$ w.r.t. the causal prefix $K :: -$ and all operators which preserve $\approx$. For instance, the following sequence of implications, that holds for any causal processes A, B and C, provides the argument for parallel composition:

$$\begin{array}{lcl} A \approx_c B & \text{implies} & [\![A]\!] \approx [\![B]\!] \\ & \text{which implies} & [\![A \mid C]\!] = [\![A]\!] \mid [\![C]\!] \approx [\![B]\!] \mid [\![C]\!] = [\![B \mid C]\!]\,, \\ & \text{which implies} & A \mid C \approx_c B \mid C\,. \end{array}$$

Similarly, we can use the encoding and the property that $\approx$ is not preserved by the sum operator to check that also $\approx_c$ is not preserved by sum. We have

$$[\![\tau.a]\!] = \tau.[\![a]\!] \approx [\![a]\!]$$

and

$$[\![\tau.a + b]\!] = \tau.[\![a]\!] + [\![b]\!] \not\approx [\![a]\!] + [\![b]\!] = [\![a + b]\!]\,.$$

Hence $\tau.a \approx_c a$, but $\tau.a + b \not\approx_c a + b$, which gives us a counterexample for the congruence of $\approx_c$ w.r.t. sum. Thus, the only congruence property for $\approx_c$ remained is the one for the input prefix operator $a(\widetilde{b}).\,-$. The known counter-examples showing that $a(\widetilde{b}).\,-$ does not preserve $\approx$ involve either the *match* operator $[a = b]$ — not considered here — or the expansion law. For example, $a \mid b \approx a.\overline{b} + \overline{b}.a$, but $c(a).\,(a \mid \overline{b}) \not\approx c(a).\,(a.\overline{b} + \overline{b}.a)$, since the process $c(a).\,(a|\overline{b})$, after receiving b, can perform a communication, whereas $c(a).\,(a.\overline{b}+\overline{b}.a)$ cannot. The same kind of counter-example does not apply to $\approx_c$, as the expansion law $a \mid b \approx a.\overline{b} + \overline{b}.a$ does not hold for this equivalence. Indeed, the problem of input prefix for $\approx_c$ appears non-trivial. A discussion on this topic is postponed to the concluding section.

More complex laws can be proved by taking advantage of the well-understood algebraic theory of $\approx$. Below, we show an example of this. Consider $P \stackrel{def}{=} \nu\, b\, (a.b.c \mid \overline{b}.d)$ and $Q \stackrel{def}{=} \nu\, b\, (a.b.d \mid \overline{b}.c)$. Processes P and Q differ only because c and d are placed at different locations. However, in both cases c and d have the same cause, namely a. Indeed, it holds that $P \approx_c Q$. We can prove this equality via algebraic manipulations, by showing that $[\![P]\!] \approx [\![Q]\!]$. If

$$R \stackrel{def}{=} h \rhd \emptyset \;\mid [\![d]\!]h \tag{5.6}$$

then we have

$$[\![P]\!] = \nu\, b \left(a(k).\, (\, k \rhd \emptyset \;\mid [\![b.c]\!]k) \mid \nu\, h\, \overline{b}\langle h\rangle.\, R\right). \tag{5.7}$$

From (5.7), applying the expansion law and simple laws for restriction (Table 2.2) we obtain

$$[\![P]\!] \;\sim\; a(k).\left(\, k \rhd \emptyset \;\mid \nu\, b\, ([\![b.c]\!]k \mid \nu\, h\, \overline{b}\langle h\rangle.\, R)\right). \tag{5.8}$$

Since, by definition,

$$[\![b.c]\!]k = b(h).\, (\, h \rhd k \;\mid [\![c]\!]h) \tag{5.9}$$

applying expansion and restriction laws we obtain

$$\begin{array}{rcl} \nu\, b\, ([\![b.c]\!]k \mid \nu\, h\, \overline{b}\langle h\rangle.\, R) & \sim & \\ \tau.\nu\, h\left(\, h \rhd k \;\mid [\![c]\!]h \mid R\right) & = & \\ \tau.\nu\, h\left(\, h \rhd k \;\mid [\![c]\!]h \mid\; h \rhd \emptyset \;\mid [\![d]\!]h\right) & \stackrel{def}{=} & T\,. \end{array} \tag{5.10}$$

Rearranging subterms, we can write

$$T \;\sim\; \tau.\nu\, h\left(\, h \rhd k \;\mid\; h \rhd \emptyset \;\mid [\![c \mid d]\!]h\right). \tag{5.11}$$

Applying Lemma 5.6.2 to the right-hand side of (5.11):

$$T \gtrsim \tau.[\![c \mid d]\!]k\,. \tag{5.12}$$

From (5.10) and (5.12), and since $\gtrsim$ is preserved by parallel composition,

$$k \rhd \emptyset \;\mid \nu\, b\, ([\![b.c]\!]k \mid \nu\, h\, \overline{b}\langle h\rangle.\, R) \gtrsim\; k \rhd \emptyset \;\mid \tau.[\![c \mid d]\!]k\,. \tag{5.13}$$

Relation (5.13) holds or any cause k. From this, due to the congruence property of $\gtrsim$, and from (5.8), we get:

$$[\![P]\!] \gtrsim a(k).\left(\, k \rhd \emptyset \;\mid \tau.[\![c \mid d]\!]k\right). \tag{5.14}$$

In a symmetric way — exchanging the roles of c and d — one proves that

$$[\![Q]\!] \gtrsim a(k).\left(\, k \rhd \emptyset \;\mid \tau.[\![d \mid c]\!]k\right). \tag{5.15}$$

But by the definition of encoding and by the commutativity law for parallel composition operator, we have

$$[\![c \mid d]\!]k \;\sim\; [\![d \mid c]\!]k\,,$$

for any k. Hence,

$$a(k).\left(\ k \rhd \emptyset \ \ | \, \tau.[\![c \mid d]\!]k \right) \sim a(k).\left(\ k \rhd \emptyset \ \ | \, \tau.[\![d \mid c]\!]k \right). \tag{5.16}$$

Finally, we deduce the original claim $[\![P]\!] \approx [\![Q]\!]$ from (5.14), (5.15) and (5.16).

The above sequence of manipulations consists of applications of familiar algebraic laws (expansion and restriction laws), plus an application of the absorption Lemma 5.6.2, to manipulate wire processes. In particular, Lemma 5.6.2 avoided us having to unfold the definition of wire processes and hence having explicitly to deal with the replication operator. A proof of $P \approx_c Q$ is also possible utilising a proof system for $\approx_c$ on CCS, like Kiehn's [Kie93]. However, the laws for the interleaving $\approx$ are simpler than those for the non-interleaving $\approx_c$, and do not require the introduction of extra operators. Kiehn's system, besides the causal prefix $K :: -$, uses the *left merge* and the *communication merge* operators, to express an appropriate form of expansion law for causal bisimulation. Moreover, Kiehn's system is not purely equational — it includes non-trivial inference rules.

5.9 A Comparison with Location Bisimulation

We have presented an encoding of the non-interleaving causal bisimulation into the interleaving observation equivalence in the π-calculus. In a previous paper [San94a], the same issue has been tackled for *location bisimulation.* Causal bisimulation and location bisimulation aim, respectively, to capture the causal and the spatial dependencies on processes. Location bisimulation can be described using a syntax and an operational semantics similar to those for causal bisimulation in Definition 5.3.9 and Table 5.1. What distinguishes location bisimulation is that an action μ causes a later action μ' only if μ' emanates from a position (called *location*) syntactically underneath μ's position. In this framework, a communication between two subterms of a process has no causal effect whatsoever because the spatial disposition of the process components is left unchanged.

The processes

$$P_1 \stackrel{def}{=} a \mid \overline{c} \qquad \text{and} \qquad P_2 \stackrel{def}{=} \nu\, b\, (a.\, b|\overline{b}.\, c) + \nu\, b\, (c.\, b|\overline{b}.\, a)$$

are location bisimilar: actions a and c are unrelated both in P_1 and P_2, because they emanate from unrelated locations. However, P_1 and P_2 are not causally bisimilar: only in P_2 the firing of either a or c is necessary to liberate the other action. On the other hand, and for opposite reasons, processes

$$Q_1 \stackrel{def}{=} \nu\, b\, (a.b.c|\overline{b}.d) \qquad \text{and} \qquad Q_2 \stackrel{def}{=} \nu\, b\, (a.b.d|\overline{b}.c)$$

are causally bisimilar (see Section 5.8) but not location bisimilar. The contrasts between the two equivalences are only imputable to the treatment of communications between processes. Indeed, Aceto's work [Ace92b] shows that they coincide in a calculus with no such communications. Affinities and differences are evidenced in their encodings into observation equivalence presented here and in [San94a]. If we limit ourselves to standard processes, then the two encodings act as a homomorphism on all operators but input and output prefixes. there is a basic schema in the two encodings of the prefixes:

$$[\![a(\widetilde{b}).\,P]\!]k \stackrel{def}{=} a(\widetilde{b}x).\,\big(\boxed{\triangleright x, h}\;\; h \rhd k \;\mid [\![P]\!]h\big) \tag{5.17}$$

$$[\![\overline{a}\langle\widetilde{b}\rangle.\,P]\!]k \stackrel{def}{=} (\nu\, x)\overline{a}\langle\widetilde{b}x\rangle.\,\big(\boxed{\triangleleft\, x, h}\;\; h \rhd k \;\mid [\![P]\!]h\big) \tag{5.18}$$

where h is a fresh name. Formally, $\boxed{\triangleright x, h}$ and $\boxed{\triangleleft\, x, h}$ stand for unary operators on the argument $h \rhd k \mid [\![P]\!]h$. But it is more enlightening — and this explains the way in which they appear in (5.17) and (5.18) — to think of them as 'missing parts', still to be filled, of the expressions (5.17) and (5.18). Indeed, $\boxed{\triangleright x, h}$ and $\boxed{\triangleleft\, x, h}$ play the role of the *access key* to the wire $h \rhd k$, which holds the causality information. In the case of location bisimulation, the access key is a restricted particle $\overline{x}\langle h\rangle$ in parallel with the wire, that is:

$$\begin{array}{lll} \text{“}\boxed{\triangleright x, h}\text{”} & \text{is} & \text{“}\nu\, h\,\overline{x}\langle h\rangle \mid\text{”} \quad \text{and} \\ \text{“}\boxed{\triangleleft\, x, h}\text{”} & \text{is} & \text{“}\nu\, h\,\overline{x}\langle h\rangle \mid\text{”}\,. \end{array} \tag{5.19}$$

If the prefix $a(\widetilde{b}x)$ or $\overline{a}\langle\widetilde{b}x\rangle$ in (5.17-5.18) is consumed in a visible action, the external observer can use the key $\overline{x}\langle h\rangle$ to access name h and, with it, the wire $h \rhd k$. By contrast, if the two complementary prefixes $a(\widetilde{b}x)$ and $\overline{a}\langle\widetilde{b}x\rangle$ communicate with each other, the respective keys remain deadlocked, because of the same output polarity. Consequently, the related wires remain disconnected. This reflects the fact that in location bisimulation a communication between processes has no effect on causes. A few simplifications on this schema lead to the final encoding, reported in Table 5.9. (Note in particular the elimination of the innermost replication of the wire: This is possible because, in the case of location bisimulation, only 'linear' causal chains may be generated, hence at most one wire may get connected to the end-point of another wire.)

In the encoding of location bisimulation interactions between a process and the external observer and between processes are different — the key $\overline{x}\langle h\rangle$ is to be used only by the external observer. The encoding of causal bisimulation is obtained by eliminating this asymmetry. A key takes the same polarity as the related prefix. Writing “$\overline{x}(h).$” as abbreviation for the bound output “$\nu\, h\,\overline{x}\langle h\rangle.$” to emphasise the symmetry, then

$$\begin{array}{lll} \text{“}\boxed{\triangleright x, h}\text{”} & \text{is} & \text{“}x(h).\text{”} \quad \text{and} \\ \text{“}\boxed{\triangleleft\, x, h}\text{”} & \text{is} & \text{“}\overline{x}(h).\text{”}\,. \end{array} \tag{5.20}$$

Thus, if the prefix $a(\widetilde{b}x)$ or $\overline{a}\langle\widetilde{b}x\rangle$ in (5.17-5.18) produces a visible action, the key $x(h)$ or $\overline{x}(h)$ can be used by the external observer, in the same way as for location bisimulation, to access the wire $h \rhd k$. But now, if the prefixes $a(\widetilde{b}x)$ and $\overline{a}\langle\widetilde{b}x\rangle$ communicate with

Encoding of Location Bisimulation; process $h \diamond k$ is $!\, h.\, \overline{k}$.

$$[\![a(\widetilde{b}).\, A]\!] k \stackrel{def}{=} a(\widetilde{b}x).\, \nu\, h\, (\overline{x}\langle h\rangle \mid h \diamond k \mid [\![A]\!] h)$$
$$[\![\overline{a}\langle\widetilde{b}\rangle.\, A]\!] k \stackrel{def}{=} \nu\, h\, \overline{a}\langle\widetilde{b}x\rangle.\, \nu\, h\, (\overline{x}\langle h\rangle \mid h \diamond k \mid [\![A]\!] h)$$

. . . .

Encoding of Causal Bisimulation.

$$[\![a(\widetilde{b}).\, A]\!] k \stackrel{def}{=} a(\widetilde{b}h).\, (\, h \rhd k \mid [\![A]\!] h)$$
$$[\![\overline{a}\langle\widetilde{b}\rangle.\, A]\!] k \stackrel{def}{=} \nu\, h\, \overline{a}\langle\widetilde{b}h\rangle.\, (\, h \rhd k \mid [\![A]\!] h)$$

Table 5.9

each other, the respective keys coalesce, because of opposite polarities. Consequently, the related wires get connected, which reflects the fact that in causal bisimulation a communication involves a merge of cause sets (the fact that a key is not blocking also justifies the sequentialisation between a key and the associated wire in (5.20), in place of the parallel composition as in (5.19)). In this schema of encoding, a key $x(h)$ or $\overline{x}\langle h\rangle$ only delays the access to the wire $h \rhd k$ one step. The encoding used in this paper is obtained by removing this level of indirection. To facilitate the comparison with the encoding location bisimulation, the clauses for input and output prefixes have been rewritten in Table 5.9.

Chapter 6

Conclusions and Related Work

6.1 Concluding Remarks

We summarize below the main achievements of the thesis.

- We have investigated a theory of testing equivalence for the π-calculus, showing that the approach smoothly applies to a name-passing setting.

- Two variants of the testing theory for the π-calculus have been been explored. In the first variant, omitting mismatch from the language leads to testing equivalences that do not correspond to any operational intuition. As a result, the importance of mismatch from an observational point of view is assessed. In the second variant, the the theory of testing over an early transitional semantics is considered and it is shown it leads to the same testing relations as the late semantics.

- A complete proof system for testing has been provided; differently from the equivalences based on bisimulation, it turns out that the axiomatization of CCS applies (up to some mild modifications) to the π-calculus as well.

- A fully abstract (w.r.t to one of the testing preorders) denotational model has been discussed. We have showed that the framework of natural interpretations is suitable for giving denotational semantics to the π-calculus. The model takes advantage of reduction of processes to certain normal forms.

- We have provided the π-calculus with a symbolic semantics; a symbolic characterization of known bisimulation-based equivalences has then been given. As a consequence, we have laid the basis for more efficient checking of these equivalences. The symbolic characterization also sheds light on the logical differences among the various forms of bisimulations.

- Complete proof systems for (late, early) symbolic bisimulations have also been provided. These improve previous systems by avoiding explicit case-analysis of the input-prefix congruence rule.

- We have investigated the meaning of causality in mobile systems, singling out two forms of causal dependencies: subject dependencies (given by the structure of processes) and object dependencies (given by the binding mechanism on names). A causal equivalence that distinguish processes on the basis of both kinds of dependencies has then been defined.

- We have shown that causal equivalence can be expressed, by means of a compositional encoding, within the interleaving observation equivalence. This result sheds further light on the expressive power of the π-calculus and permits using the simpler theory of the interleaving equivalence to reason about causality in mobile systems.

6.2 Related Work

Besides [MPW92], our work on testing equivalence relates mainly to [Hen91] and, in some respects, to [PS93]. In [Hen91], independently, a testing-based axiomatization and a fully abstract denotational model are worked out for a language slightly different from ours: internal moves are replaced by a binary internal choice operator and the match- mismatch pair is replaced by an *if-then-else* construct. The actual completeness proof relies on the existence of head normal forms similar to ours. The different selection of operators does not permit any assessment of the widely studied original π-calculus with respect to the testing approach. It also prevents a precise comparison between the work described in this chapter and Hennessy's. It can however be said that the main differences between the two axiomatizations are essentially the same as those between the axiomatization of testing for CCS [DH84] and that for TCCS [DH87]. The comparison becomes more significant when considering the two denotational models: in [Hen91] no notion of normal form is introduced and the compact elements of the domain are defined as (the equivalence classes of) those finite terms which have a "finite behaviour" on input actions. As a consequence a term-model is obtained.

In [PS93], four strong bisimulation-based equivalences over the π-calculus are axiomatized, namely early and late ground bisimulations and the congruences they induce. The axiomatizations of the ground equivalences contain an input prefix inference rule similar to our rule C1 and to that used in Hennessy's work. In the proof systems for congruences, the input prefix rule is replaced by a simpler one that exploits the fact that congruences are fully preserved by substitutions. All axiomatizations of [PS93], but that of late ground equivalence, heavily rely on the presence of the mismatch operator. The authors pose the interesting problem of formally establishing the necessity of this operator for axiomatizing bisimulation-based equivalences. We have seen that mismatch has a fundamental role in the definition of our testing preorders; its omission would lead to a theory that does not correspond to any operational intuition.

Let us now discuss some work related with our framework for symbolic equivalence. The papers mainly related are [HL92],[HL93] and, again [PS93]. The original idea of symbolic bisimulation has been presented in [HL92]. There, a verification algorithm is proposed for a class of symbolic transition graphs and a theorem relating symbolic bisimulations to concrete bisimulations over a version of CCS with value-passing is presented. In [HL93], the same language has then been equipped with a sound and complete proof system. The results obtained by Hennessy and Lin are the direct inspiration of our work but they cannot be directly extended to the π-calculus for two main reasons:

1. the blurring of the distinction between variables, values and channels proper of the π-calculus;

2. the absence of a specific language for boolean constraints in the work by Hennessy and Lin.

It is easier to deal with a static value-passing process algebra, because channel names are neatly separated from the exchanged values and thus channels do not appear in the constraints. Of course, this is no longer true in a name-passing calculus, where a subtle interplay between name-scoping and boolean constraining is present. An example of such interplay is offered by the symbolic SOS rules for the restriction operator:

$$\texttt{Res} \;:\; \frac{P \stackrel{\phi,\,\alpha}{\longrightarrow} P'}{(y)P \stackrel{(y)\phi,\,\alpha}{\longrightarrow} (y)P'} \; y \notin n(\alpha).$$

The symbolic framework of [HL92] and [HL93] is parameterized on the language of boolean constraints, in other words they do not have a specific language for them. In order to establish the relationship between symbolic and concrete bisimulation, they just assume existence of an extremely expressive language, capable of describing any given collection of environments (associations of variables with values). This is admittedly [HL92] a very strong requirement. It is at least not obvious, in presence of non-trivial value types, that such a language exists. Here, we had to consider a *specific* language (BF) and had to deal with name substitutions rather than with environments. Indeed, it must be said that our solution heavily depends on the specific features of the π-calculus: only finitely many substitutions are important when dealing with a fixed set of constraints and processes. This property does not hold for languages that, besides names, permit exchanging other kinds of values (e.g. integers) and make use of predicates (e.g. $\leq$) over them.

We now come to [PS93]. If we confine ourselves to one specific form of bisimulation (be it early or late), the main differences between our proof system and theirs can be summarized as follows. In [PS93], the ground equivalence and the congruence are considered separately; in our framework, all equivalences obtainable as substitution-closures of the ground one (including, as particular cases, the ground equivalence itself and the congruence) are considered at once. As a consequence, it is possible to reason about each such equivalence, just by selecting the appropriate ϕ. Furthermore, in many cases, we gain in efficiency: within our framework, if it has to be proven that $x(y).P$ is ground bisimilar to $x(y).Q$, it just suffices to derive $\phi \rhd P = Q$, for some ϕ not containing y and not stronger than the constraint given by our symbolic characterization of $\sim$. Within the framework of [PS93], it is instead needed to apply the input-prefix rule:

$$\mathtt{IP} \ : \ \frac{\forall z \in fn(P,Q,y).\ P\{z/y\} = Q\{z/y\}}{x(y).P = x(y).Q}$$

whose premise *always* requires as many sub-proofs as the cardinality of $fn(P,Q,y)$. This example shows that making reasoning assumptions explicit can often avoid a number of useless checks. An accurate comparison between the two approaches w.r.t. efficient deduction strategies would be interesting.

Our work is also somewhat related to [Dam93] and to [FMQ94], where other symbolic transitional semantics for the π-calculus have been presented. In [Dam93], a symbolic semantics is used as a basis for developing a model checker; first-order (rather than boolean) formulae are used; in the operational rules for the restriction operator, the "hiding" of a name y in a formula is modeled using an existential quantifier $\exists y$. The aim of [FMQ94] is to define a general framework, within which the different kinds of instantiation strategies (such as early, late, open) can be described just by instantiating certain parameters. The problem of efficiently representing the considered equivalences is not tackled.

Between the publication of the extended abstract of the paper on which this chapter is based [BD94b] and the revision of the final version, we learnt of a related work by Hui Min Lin [Lin94]. There, a symbolic framework similar to ours is proposed. The considered calculus does not contain mismatch, which is however used in the language for boolean constraints; furthermore the latter does not permit disjunctions. These facts prevent dealing with negation in processes and capturing specific identifications (such as $[x=y]P + [x=z]P \simeq^{\phi} P$, with $\phi = [x=y] \vee [x=z]$). The loss of generality allows for a simpler treatment of restriction; also in [Lin94] the formulation of symbolic bisimulation is slight simpler than ours, in that a particular "canonical" decomposition is always forced. However, due to this very fact, it is not clear to us how this formulation could lead to efficient verification.

6.3 Future Work

It must be said that the construction of our denotational model is syntax-dependent and thus that it is not completely satisfactory. We would have preferred a more abstract presentation of the cpo, for example in terms of a domain equation; an attempt has been made of using a variant of *generalized acceptance trees* as in [HI91], but we could not directly describe the restriction operator as a binder in such a model. To find an abstract description of a denotational model for the calculus is certainly a challenging task.

We have not tackled the problem of the input prefix congruence rule for testing. The presence of this rule in a proof system leads in general to inefficient derivations, in that multiple checks are required whenever a (sub-)proof of the form $a(x).P \sqsubseteq a(x).Q$ is needed. A possible solution to this problem would consist in devising a proof system with guarded statements of the form $\phi \vdash a(x).P \sqsubseteq a(x).Q$, where ϕ is a boolean condition over names, as we did in Chapter 4 for bisimulation. It would be interesting to extend the approach to the theory of testing.

The symbolic characterization of the bisimulations sheds new light on the logical difference between π-calculus bisimulations based on different instantiation strategies, such as early, late and *open* [San93]. Indeed, it is not difficult to see that different instantiation strategies correspond to different degrees of generality in the case analysis. Indeed, early bisimulation is the most general (and natural) equivalence, since no restriction is imposed on the case-decomposition D. Late bisimulation is obtained by requiring that the formal parameter of the input action be not in D, i.e. by forbidding case-analysis on the actual value of the formal parameter. Open bisimulation is only defined over the language without negation (hence essentially without mismatch $[x \neq y]$). We strongly conjecture that, for this sublanguage, open bisimulation can be obtained from the early symbolic one by completely omitting case analysis; this amounts essentially to requiring that D be a singleton. The conjecture is mainly suggested by the "efficient characterization" of open bisimulation given in [San93].

We believe that the full abstraction of our encoding of causal equivalence into observation equivalence extends to the congruence relations; i.e., if $\approx^{\mathrm{c}}$ and $\approx^{\mathrm{c}}_{\mathrm{c}}$ are the congruences induced by observation equivalence and causal bisimulation, respectively, then for all causal terms A and B, it holds that

$$A \approx^{\mathrm{c}}_{\mathrm{c}} B \quad \text{iff} \quad [\![A]\!] \approx^{\mathrm{c}} [\![B]\!] .$$

It should be possible to prove this exploiting the full abstraction on the bisimilarity relations. The result could then be used to examine whether a proof system for $\approx^{\mathrm{c}}_{\mathrm{c}}$ can be obtained from one for $\approx^{\mathrm{c}}$ plus, possibly, Lemmata 5.5.4, 5.5.6, 5.6.1 and 5.6.2 to manipulate wires.

An interesting issue is the study of the congruence properties for the input-prefix operator in causal-sensitive behavioural equivalences. Intuitively, a relation over π-calculus processes is preserved by input prefix if it is preserved by name substitutions. This property usually fails if the matching operator is present, as for the calculus in [MPW92]. If a and b are different names, one might want to regard $[a = b]a.\mathbf{0}$ and $\mathbf{0}$ as equivalent since both are deadlocked, but $([a = b]a.\mathbf{0})\{a/b\}$ and $\mathbf{0}\{a/b\}$ as distinct since only the latter is deadlocked. Indeed, with the matching operator observation equivalence, location bisimulation and causal bisimulation are all not preserved by substitutions. However, in many situations matching can be avoided or simulated. Hence one might wish to consider a matching-free calculus, as we did in the present chapter. In such a language, the congruence for substitutions still fails for observation equivalence and location bisimulation [San94a], but we strongly conjecture that it holds for causal bisimulation. The failure for the former equivalences is essentially due to the fact that they may relate processes with different degrees of parallelism (consider, for instance, the processes P_1 and P_2 in Section 5.9); as a consequence, when a substitution applied to such two processes identifies previously distinct channels, new communication possibilities may be created in one process, but not in the other. However, it can be expected that the situation for $\approx_c$ is better, since $\approx_c$ does respect parallelism. Preliminary results indicate that in the matching-free calculus $\approx_c$ *is* indeed preserved by substitutions and hence by input prefix. If confirmed, this would have significant consequences, both practical — proving causal bisimulation equalities would be easier — and more theoretical — the proliferation of different forms of bisimulation, like the *early, late* and *open* of interleaving and location bisimilarities [PS93, San93, San94a], would be avoided in this causality framework.

We have individuated two forms of causal dependency between π-calculus actions, which we called subject and object dependencies. The former originate from prefix-nesting and are propagated through interactions within processes, the latter originate from the name-binding mechanisms. Our formulation of causal bisimulation distinguishes between processes which differ for the subject or for the object dependencies. Thus we discriminate between $P \stackrel{def}{=} \nu\, b\,\big(\overline{a}b.\,\mathbf{0} \,|\, \overline{b}y.\,\mathbf{0}\big)$ and $Q \stackrel{def}{=} \nu\, b\,(\overline{a}b.\,\overline{b}y.\,\mathbf{0})$, because in P there is both a subject and an object dependency between the actions at a and at b, whereas in Q there is only an object dependency. We would like to compare this notion of causal bisimulation with one in which subject and object dependencies are still taken into account but not separated, so to equate P and Q. One possibility is to carry out this analysis using our encoding or variations of it.

Bibliography

[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[Ace92a] L. Aceto. History preserving, causal and mixed-ordering equivalence over stable event structure (Note). *Fundamentae Informaticae*, 17(4):319–331, 1992.

[Ace92b] L. Aceto. Relating distributed, temporal and causal observations of simple processes. *Fundamentae Informaticae*, 17(4):369–397, 1992.

[AKH92] Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[BCHK93] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing localities. *Theoretical Computer Science*, 114:31–61, 1993. Extended abstract in *Proceedings MFCS '91, LNCS 520*, 1991.

[BCHK94] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. *Formal Aspects of Computing*, 6:165–200, 1994. Extended abstract in *Proceedings Concur '92, LNCS 630*, 1992.

[BD92a] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. Technical Report SI- 92 - 04, Dipartimento di Scienze dell'Informazione Università "La Sapienza", Roma, 1992. To appear in *Information and Computation.*

[BD92b] M. Boreale and R. De Nicola. Testing equivalence for mobile processes - Extended abstract. In R. Cleaveland, editor, *Proceedings of CONCUR '92, LNCS 630*, pages 2–17. Springer-Verlag, Berlin, 1992.

[BD94a] M. Boreale and R. De Nicola. A symbolic semantics for the π-calculus. Technical Report SI-94-04, Dipartimento di Scienze dell'Informazione Università "La Sapienza", Roma, 1994.

[BD94b] M. Boreale and R. De Nicola. A symbolic semantics for the π-calculus- Extended abstract. In B. Jonsson and J. Parrow, editors, *Proceedings of Concur '94, LNCS 836*, pages 299–314. Springer-Verlag, Berlin, 1994.

[BK89] J.A. Bergstra and J.-W. Klop. Process theory based on bisimulation semantics. In *Linear Time, Branching Time and Partial Orders in Logic and Models for Concurrency, LNCS, 354*, pages 50–122. Springer-Verlag, Berlin, 1989.

[BS94] M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the π-calculus. Technical Report ECS-LFCS-94-297, Department of Computer Science, University of Edinburgh, 1994.

[BS94b] M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the π-calculus - Ext. abstr. In E. Mayr and C. Puech, editors, *Proceedings of STACS'95, LCNS 900*, pages 243-254, Springer-Verlag, Berlin, 1995.

[BW90] J. Baeten and W. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Uuniversity Press, 1990.

[CD93] F. Corradini and R. De Nicola. Locality and causality in distributed process algebras. Technical Report SI-93/04, Dipartimento di Scienze dell'Informazione, Università degli studi di Roma "La Sapienza", 1993. Revised version in *Proc. ICALP'94, LNCS 820, pp. 154-165.*

[CH89] I. Castellani and M. Hennessy. Ditributed bisimulations. *Journal of ACM*, 10(4):887–911, 1989.

[Dam93] M. Dam. Model checking mobile processes. In E. Best, editor, *Proceedings of CONCUR '93, LNCS 715*. Springer-Verlag, Berlin, 1993.

[DD89] P. Degano and P. Darondeau. Causal trees. In *Proc. 15th ICALP, LNCS 372, Springer-Verlag*, pages 234–248, 1989.

[DDM88] P. Degano, R. De Nicola, and U Montanari. Partial ordering descriptions and observations of concurrent processes. In *Linear Time, Branching Time and Partial Order in Logics and Models of Concurrency, LNCS 354, Springer-Verlag*, pages 438–466, 1988.

[DDM93] P. Degano, R. De Nicola, and U. Montanari. Universal axioms for bisimulation. *Theoretical Computer Science*, 114:63–91, 1993.

[DeN87] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, J24:211–237, 1987.

[DH84] R. De Nicola and M. Hennessy. Testing equivalence for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[DH87] R. De Nicola and M. Hennessy. CCS without τ's. In *Proceedings of Tapsoft '87, LNCS 249*, pages 138–152, 1987.

[DIN90] R. De Nicola, P. Inverardi, and M. Nesi. Using axiomatic presentation of behavioural equivalences for manipulating CCS specifications. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Machines, LNCS 407*, pages 54–67. Springer-Verlag, 1990.

[DP92] P. Degano and C. Priami. Proved trees. In *Proc. ICALP 92, LNCS 623*. Springer-Verlag, 1992.

[EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical Report DAIMI-PB 208, Computer Science Dept., Aarhus University, 1986.

[FMQ94] G. Ferrari, U. Montanari, and P. Quaglia. π-calculus with explicit substitutions. Technical report, Università di Pisa, 1994. Extended abstract appeared in *Proc. of MFCS'94*.

[Gue81] I. Guessarian. *Algebraic Semantics*. LNCS, 99. Springer-Verlag, Berlin, 1981.

[Hen88] M. Hennessy. *An Algebraic Theory of Processes*. MIT Press, Cambridge, 1988.

[Hen91] M. Hennessy. A model for the π-calculus. Technical report, University of Sussex, 1991.

[HI91] M. Hennessy and A. Ingolfsdottir. A theory of testing equivalence with value-passing. *Information and Computation*, 1991. (to appear).

[HL92] M. Hennessy and H. Lin. Symbolic bisimulations. Technical report, University of Sussex, 1992. Revised version in *Theoretical Computer Science,* 88: 353-389, 1995.

[HL93] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. In E. Best, editor, *Proceedings of CONCUR '93, LNCS 715*. Springer-Verlag, Berlin, 1993.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall Int., London, 1985.

[HP80] M. Hennessy and G. Plotkin. A term model for CCS. *LNCS*, 88, 1980.

[Ing93] A. Ingolfsdottir. Late and early semantics coincide for testing. Technical Report IR93-2008, Departement of Mathematics and Computer Science, University of Aalborg, 1993.

[Kie91] A. Kiehn. Local and global causes. Technical Report 342/23/91, Technische Universität München, 1991. Revised version in *Acta Informatica*, 31:697-718, 1994 with the title: Comparing locality and causality based equivalences.

[Kie93] A. Kiehn. Proof systems for cause based equivalences. In Springer Verlag, editor, *Proc. MFCS'93, LNCS 711*, 1993.

[KS90] P.C. Kannellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[Lin94] H. Lin. Symbolic bisimulations and proof systems for the π-calculus. Technical report, Laboratory for Computer Science, Institute of Software, Chinese Academy of Sciences, 1994.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. LNCS, 92. Springer-Verlag, Berlin, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Mil91] R. Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, LFCS, Dept. of Computer Science, Edinburgh Univ., 1991.

[Mil92] R. Milner. Functions as processes. *Mathematical Structure in Computer Science*, 2(2):119–141, 1992.

[MPW91] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. In *Proc. Concur '91, LNCS 527*. Springer-Verlag, 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I and II. *Information and Computation*, 100:1 –41 and 42–78, 1992. Appeared in 1989 as a technical report of the Comp. Sci. Dept. , Unv. of Edinburg.

[MY92] M. Montanari and D. Yankelevich. A parametric approach to localities. In *Proc. ICALP 92, LNCS 623*, pages 617–628. Springer-Verlag, 1992.

[Par81] D.M. Park. Concurrency on automata and infinite sequences. In P. Deussen, editor, *Conf. on Theoretical Computer Science, LNCS 104*. Springer-Verlag, 1981.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[PS93] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. Technical Report ECS–LFCS–93–262, University of Edinburgh, Department of Computer Science, 1993. To appear in *Information and Computation.*

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

[San93] D. Sangiorgi. A theory of bisimulation for the π-calculus. In E. Best, editor, *Proceedings of CONCUR '93, LNCS 715*. Springer-Verlag, Berlin, 1993. To appear in *Acta Informatica*.

[San94a] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. Technical Report ECS–LFCS–94–282, LFCS, Dept. of Computer Science, Edinburgh University, 1994. An extract appeared in *Proc. TACS '94, LNCS*, Springer-Verlag.

[San94b] D. Sangiorgi. On the bisimulation proof method. Technical Report ECS-LFCS-94-299, Department of Computer Science, University of Edinburgh, 1994. Extended abstract to appear in the *Proc. MFCS '95*.

[SM92] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up-to". In W.R. Cleveland, editor, *Proceedings of CONCUR '92*, volume 630, pages 32–46. Springer Verlag, 1992.

[Tho89] B Thomsen. A calculus of higher order processes. In *Proc. of POPL 89, ACM*, 1989.

[Wal94] D. Walker. Objects in the π-calculus. *Information and Computation*, 1994. To appear.

Appendix A

Proofs of Lemmata 5.6.1 and 5.6.2.

To facilitate the proof of a few lemmata in this appendix and in the next one, and to better capture the strength of their assertions we use the *structural congruence* relation. Roughly, two processes are structurally congruent if they may only differ syntactically with each other on the way in which their respective subcomponents are assembled. Structural congruence [Mil91] is written $\equiv_S$ and is defined as the smallest congruence over processes which satisfies the rules in Table A.1. These are self-evident rules which must be valid in any reasonable behavioural equivalence.

Lemma A.1.1 *The relation $\equiv_S$ is a strong bisimulation and is preserved by name substitution.* $\square$

The following lemma allows us to rename names in the object part of input transitions.

Lemma A.1.2 (Renaming Lemma) *Suppose $P \stackrel{a\langle\widetilde{b}\rangle}{\longrightarrow} P'$. Then there are $\widetilde{x}$ and P'' s.t. $P' = P''\{\widetilde{b}/\widetilde{x}\}$ and for all $\widetilde{c}$, $P \stackrel{a\langle\widetilde{c}\rangle}{\longrightarrow} P''\{\widetilde{c}/\widetilde{x}\}$.*

Proof: Simple transition induction. $\square$

Corollary A.1.3

1. *If $P \stackrel{a\langle\widetilde{b}h\rangle}{\longrightarrow} P'$, and $k \cap fn(P) = \emptyset$, then also $P \stackrel{a\langle\widetilde{b}k\rangle}{\longrightarrow} P''$, for some P'' with $P''\{h/k\} = P'$.*

2. *If $P \stackrel{a\langle\widetilde{b}k\rangle}{\longrightarrow} P''$ and $k \cap fn(P) = \emptyset$, then, for all h, also $P \stackrel{a\langle\widetilde{b}h\rangle}{\longrightarrow} P' = P''\{h/k\}$.* $\square$

The proofs of Lemmata 5.6.1 and 5.6.2 use Lemmata 5.5.4 and 5.5.6. We also rely on Lemma A.1.5 below. This lemma, basically, says that the actions a process $[\![A]\!]K$ can perform do not depend from the parameter K. That is, two processes $[\![A]\!]K$ and $[\![A]\!]K'$ can

Abelian monoid laws for $+$:	$P_1 + P_2$	$\equiv_S$	$P_2 + P_1$
	$P_1 + (P_2 + P_3)$	$\equiv_S$	$(P_1 + P_2) + P_3$
	$P + \mathbf{0}$	$\equiv_S$	P
Abelian monoid laws for $\mid$:	$P_1 \mid P_2$	$\equiv_S$	$P_2 \mid P_1$
	$P_1 \mid (P_2 \mid P_3)$	$\equiv_S$	$(P_1 \mid P_2) \mid P_3$
	$P \mid \mathbf{0}$	$\equiv_S$	P
Restriction:	$\nu\, a\, \mathbf{0}$	$\equiv_S$	$\mathbf{0}$
	$\nu\, a\, \nu\, b\, P$	$\equiv_S$	$\nu\, b\, \nu\, a\, P$
if $a \notin fn(P_2)$, then	$(\nu\, a\, P_1) \mid P_2$	$\equiv_S$	$\nu\, a\, (P_1 \mid P_2)$
Replication:	$!\,P$	$\equiv_S$	$P \mid\, !\,P$

Table A.1: The rules of structural congruence ($\equiv_S$)

perform the same actions and, in doing so, they evolve to derivatives which are the same up to the choice of the parameter K or K'. To express this fact, we have to accurately study how the parameter K is used in a transition $[\![A]\!]K \xrightarrow{\mu} P$. In the proof of Lemma A.1.5 we use the following fact:

Lemma A.1.4 *If $!Q \xrightarrow{\mu} Q'$ with $\mu \neq \tau$, then there is Q'' s.t. $Q' \equiv_S Q'' \mid\, !Q$ and $Q \xrightarrow{\mu} Q''$.* □

Lemma A.1.5 *Let A be a causal term, $K, K' \subseteq_{\mathrm{fin}} \mathcal{K}$ and suppose $[\![A]\!]K \xrightarrow{\mu} P$. Then*

1. *If $\mu = a\langle\widetilde{b}h\rangle$ or $\mu = (\nu\, \tilde{c}h)\overline{a}\langle\widetilde{b}h\rangle$, for $h \notin \mathcal{K}(A) \cup K \cup K'$, then there exist $K_0 \subseteq \mathcal{K}(A)$, $\widetilde{d}$, A' and R with $h \notin \mathcal{K}(A')$, s.t.*

$$P \equiv_S \; h \rhd (K, K_0) \;\mid (\nu\, \widetilde{d})([\![A']\!]K \mid [\![h :: R]\!])$$

   and moreover

$$[\![A]\!]K' \xrightarrow{\mu} \equiv_S \; h \rhd (K', K_0) \;\mid (\nu\, \widetilde{d})([\![A']\!]K' \mid [\![h :: R]\!])\,.$$

2. *If $\mu = \tau$, then either*
   - *there exists A' s.t. $P \equiv_S [\![A']\!]K$ and also $[\![A]\!]K' \xrightarrow{\tau} Q \equiv_S [\![A']\!]K'$, or*
   - *there are $K_i \subseteq_{\mathrm{fin}} \mathcal{K}(A)$ $i \in \{1,2\}$,$\widetilde{d}$, A', R, and some $h \notin (\mathcal{K}(A') \cup K \cup K')$ s.t.*

$$P \equiv_S (\nu\, h)\big(\; h \rhd (K_1, K) \;\mid\; h \rhd (K_2, K) \;\mid (\nu\, \widetilde{d})([\![A']\!]K \mid [\![h :: R]\!])\big)$$

   and also

$$[\![A]\!]K' \xrightarrow{\tau} \equiv_S \quad (\nu\, h)\big(\; h \rhd (K_1, K') \;\mid\; h \rhd (K_2, K') \;\mid (\nu\, \widetilde{d})([\![A']\!]K' \mid [\![h :: R]\!])\big) \quad .$$

PROOF: The proof of (1) is by induction on the structure of A. We consider the most significant cases in detail, namely the cases in which the outermost operator of A is a causal prefix or a replication.

Cause prefix $A = H :: B$.

Thus $[\![A]\!]K = [\![B]\!](K, H)$ and $[\![A]\!]K' = [\![B]\!](K', H)$. We have $[\![B]\!](K, H) \xrightarrow{\mu} P$. Hence, by the inductive assumption, there are $K_0' \subseteq \mathcal{K}(B), \widetilde{d}, A''$ and R s.t.

$$\begin{array}{rcl} P & \equiv_S & h \rhd (K, H, K_0') \mid (\nu\, \widetilde{d})([\![A'']\!](K, H) \mid [\![h :: R]\!]) \\ & = & h \rhd (K, H, K_0') \mid (\nu\, \widetilde{d})([\![H :: A'']\!]K \mid [\![h :: R]\!]) \end{array}$$

and

$$\begin{array}{rcl} [\![B]\!](K', H) & \xrightarrow{\mu} \equiv_S & h \rhd (K', H, K_0') \mid (\nu\, \widetilde{d})([\![A'']\!](K', H) \mid [\![h :: R]\!]) \\ & = & h \rhd (K', H, K_0') \mid (\nu\, \widetilde{d})([\![H :: A'']\!]K' \mid [\![h :: R]\!]) . \end{array}$$

For $K_0 \stackrel{def}{=} H \cup K_0'$ and $A' \stackrel{def}{=} H :: A''$, this concludes the case.

Replication $A = !\,Q$.

Thus $[\![A]\!]K = !\,[\![Q]\!]K \xrightarrow{\mu} P$. By Lemma A.1.4, we have $P \equiv_S P_1 \mid !\,[\![Q]\!]K$, for some P_1 s.t. $[\![Q]\!]K \xrightarrow{\mu} P_1$. Using the inductive assumption on $[\![Q]\!]K \xrightarrow{\mu} P_1$ we get that for some processes R and S and name tuple $\widetilde{d}$,

$$P_1 \equiv_S h \rhd K \mid (\nu\, \widetilde{d})([\![S]\!]K \mid [\![h :: R]\!])$$

and also $[\![Q]\!]K' \xrightarrow{\mu} P_2$ with

$$P_2 \equiv_S h \rhd K' \mid (\nu\, \widetilde{d})([\![S]\!]K' \mid [\![h :: R]\!]) .$$

Now, from $[\![Q]\!]K' \xrightarrow{\mu} P_2$ and rule `Par` , we infer

$$[\![Q]\!]K' \mid !\,[\![Q]\!]K' \xrightarrow{\mu} P_2 \mid !\,[\![Q]\!]K' \stackrel{def}{=} P'$$

and hence, applying `S-rep`:

$$!\,[\![Q]\!]K' = [\![\,!\,Q]\!]K' \xrightarrow{\mu} P' .$$

To sum up, extruding the scope of $(\nu\, d)$, we have:

$$\begin{array}{rl} P \equiv_S & P_1 \mid !\,[\![Q]\!]K \equiv_S h \rhd K \mid (\nu\, \widetilde{d})([\![S]\!]K \mid [\![h :: R]\!]) \mid [\![\,!\,Q]\!]K \\ & \equiv_S h \rhd K \mid (\nu\, \widetilde{d})([\![S \mid !\,Q]\!]K \mid [\![h :: R]\!]) \end{array}$$

and similarly

$$[\![\,!\,Q]\!]K' \equiv_S h \rhd K' \mid (\nu\, \widetilde{d})([\![S \mid !\,Q]\!]K' \mid [\![h :: R]\!]) .$$

For $K_0 \stackrel{def}{=} \emptyset$ and $A' \stackrel{def}{=} S \mid !\,Q$, this proves our claim.

Now, the proof of assertion (2) of the lemma. The proof is again an induction on the structure of A. The non-trivial cases are when the outermost operator in A is a parallel composition or a replication. We only consider the former case. Thus suppose $A = A_1 \mid A_2$; the last rule applied in deriving $[\![A]\!]K \xrightarrow{\tau} P$ must be either a `Par` or a `Com` ; we analyse the latter case; the former is easier. We have:

$$\texttt{Com} \ : \frac{[\![A_1]\!]K \xrightarrow{a\langle\widetilde{bh}\rangle} P_1 \qquad [\![A_2]\!]K \xrightarrow{(\nu\,\widetilde{ch})\overline{a}\langle\widetilde{bh}\rangle} P_2}{[\![A]\!]K = [\![A_1]\!]K \mid [\![A_2]\!]K \xrightarrow{\tau} (\nu\,\widetilde{c}h)(P_1 \mid P_2) \stackrel{def}{=} P}.$$

For simplicity we suppose that $h \notin K \cup K' \cup \mathcal{K}(A)$; the case $h \in K \cup K' \cup \mathcal{K}(A)$ can be accommodated using Corollary A.1.3(1) to first rename h to a fresh name.

From part (1) of this lemma, we know that there are $A'_i, K_i \subseteq \mathcal{K}(A_i), \widetilde{d}_i$ and R_i with $h \notin A'_i$, for $i = 1, 2$, such that

$$P_i \equiv_S \ h \rhd (K_i, K) \ \mid (\nu\,\widetilde{d}_i)([\![A'_i]\!]K \mid [\![h :: R_i]\!])\,.$$

and also

$$[\![A_1]\!]K' \xrightarrow{a\langle\widetilde{bh}\rangle} Q_1 \quad \text{and} \quad [\![A_2]\!]K' \xrightarrow{(\nu\,\widetilde{ch})\overline{a}\langle\widetilde{bh}\rangle} Q_2 \tag{A.1}$$

with

$$Q_i \equiv_S \ h \rhd (K_i, K') \ \mid (\nu\,\widetilde{d}_i)([\![A'_i]\!]K' \mid [\![h :: R_i]\!]), \text{ for } i \in \{1,2\}.$$

From the two transitions in (A.1) and rule `Com` , we infer

$$[\![A]\!]K' = [\![A_1]\!]K' \mid [\![A_2]\!]K' \xrightarrow{\tau} (\nu\,\widetilde{c}h)(Q_1 \mid Q_2) \stackrel{def}{=} Q\,.$$

Finally, for $A' \stackrel{def}{=} A'_1 \mid A'_2$, $\widetilde{d} \stackrel{def}{=} \widetilde{c}\widetilde{d_1}\widetilde{d_2}$ and $R \stackrel{def}{=} R_1 \mid R_2$, using the equality $[\![h :: R_1]\!] \mid [\![h :: R_2]\!] = [\![h :: (R_1 \mid R_2)]\!]$, we can write:

$$P \equiv_S (\nu\,h)\big(\ h \rhd (K_1, K) \ \mid \ h \rhd (K_2, K) \ \mid (\nu\,\widetilde{d})([\![A']\!]K \mid [\![h :: R]\!])\big)$$

and, similarly,

$$Q \equiv_S (\nu\,h)\big(\ h \rhd (K_1, K') \ \mid \ h \rhd (K_2, K') \ \mid (\nu\,\widetilde{d})([\![A']\!]K' \mid [\![h :: R]\!])\big)\,.$$

This concludes the case. □

We are now ready to prove Lemma 5.6.1. We first recall its assertion:

$$h \rhd K \ \mid [\![A]\!](h, K_0, K) \lesssim \ h \rhd K \ \mid [\![A]\!](h, K_0)\,.$$

Proof of Lemma 5.6.1. Let

$$H \stackrel{def}{=} \{h\} \cup K_0 \cup K \qquad \text{and} \qquad H' \stackrel{def}{=} \{h\} \cup K_0\,. \tag{A.2}$$

We show that the relation

$$\mathcal{R} = \{(\ h \rhd K \ \mid [\![A]\!]H,\ \ h \rhd K \ \mid [\![A]\!]H') \ : \ A \in \mathcal{P}_{\mathrm{c}}(\mathcal{N}, \mathcal{K})\}$$

is an expansion up to context and up to $\gtrsim$. Let $(Q_1, Q_2) \in \mathcal{R}$, for $Q_1 \stackrel{def}{=} \ h \rhd K \ \mid [\![A]\!]H$ and $Q_2 \stackrel{def}{=} \ h \rhd K \ \mid [\![A]\!]H'$. We show that the moves of Q_1 can be matched by Q_2. The converse, on the actions from Q_2, is proved by reversing the steps. If Q_1 moves, then the move originates either from $h \rhd K$, or from $[\![A]\!]H$ (an interaction among these two processes is impossible, see Remark 5.6.12). We only show the details of the second case, since the first case is easier. Thus, we suppose that $Q_1 \stackrel{\mu}{\longrightarrow} \ h \rhd K \ \mid P_1 \stackrel{def}{=} Q_1'$, for some P_1 s.t. $[\![A]\!]H \stackrel{\mu}{\longrightarrow} P_1$. We have to find a static context $C[\cdot]$, and processes Q_2', T_1, and T_2 s.t.

$$Q_1' \sim C[T_1], \quad Q_2 \stackrel{\mu}{\Longrightarrow} Q_2' \gtrsim C[T_2] \quad \text{and} \quad (T_1, T_2) \in \mathcal{R}\ . \tag{A.3}$$

We distinguish the cases in which μ is an input, an output, or a silent action.

Case 1: μ is an input, say $\mu = a\langle \widetilde{b} h' \rangle$.

Let h_0 be a fresh name. From $[\![A]\!]H \stackrel{a\langle \widetilde{b}h'\rangle}{\longrightarrow} P_1$ and Corollary A.1.3(1), we have

$$[\![A]\!]H \stackrel{a\langle \widetilde{b}h_0\rangle}{\longrightarrow} P_1' \text{ with } P_1'\{^{h'}\!/h_0\} \equiv_S P_1\,.$$

From Lemma A.1.5(1), we know that there are $H_0 \subseteq \mathcal{K}(A)$, $\widetilde{d}$ and A' with $h_0 \notin \mathcal{K}(A')$, s.t.

$$P_1' \equiv_S \ h_0 \rhd (H_0, H) \ \mid (\nu\, \widetilde{d})([\![A']\!]H \mid [\![h_0 :: R]\!])$$

and moreover

$$[\![A]\!]H' \stackrel{a\langle \widetilde{b}h_0\rangle}{\longrightarrow} P_2' \equiv_S \ h_0 \rhd (H_0, H') \ \mid (\nu\, \widetilde{d})([\![A']\!]H' \mid [\![h_0 :: R]\!])\,.$$

From the above transition and Corollary A.1.3(2), we obtain

$$[\![A]\!]H' \stackrel{a\langle \widetilde{b}h'\rangle}{\longrightarrow} P_2 \quad \text{with } P_2 \equiv_S P_2'\{^{h'}\!/h_0\}.$$

Now, from this, using rule `Par` , we get a transition for Q_2:

$$Q_2 = \ h \rhd K \ \mid [\![A]\!]H' \stackrel{a\langle \widetilde{b}h'\rangle}{\longrightarrow} \ h \rhd K \ \mid P_2 \stackrel{def}{=} Q_2'\,.$$

Thus, we have found the process Q_2' to use in (A.3); we have now to find $C[\cdot]$, T_1 and T_2, and prove the requirements in (A.3). From the definitions of Q_1', P_1 and P_1', we get

$$Q_1' \equiv_S \ h \rhd K \ \mid P_1'\{^{h'}\!/h_0\} \equiv_S \ h \rhd K \ \mid h' \rhd (H_0, H) \ \mid (\nu\, \widetilde{d})([\![A']\!]H \mid [\![h' :: R]\!]) \tag{A.4}$$

and, similarly, from the definitions of Q'_2, P_2 and P'_2, we get

$$Q'_2 \equiv_S h \rhd K \mid P'_2\{^{h'}/_{h_0}\} \equiv_S h \rhd K \mid h' \rhd (H_0, H') \mid (\nu\, \widetilde{d})([\![A']\!]H' \mid [\![h' :: R]\!]) \,. \quad \text{(A.5)}$$

From (A.4) and (A.5), extruding the scope of $(\nu\, \widetilde{d})$, and commuting the order of some components, we get

$$\begin{aligned} Q'_1 &\equiv_S (\nu\, \widetilde{d})([\![h' :: R]\!] \mid h' \rhd (H_0, H) \mid h \rhd K \mid [\![A']\!]H) & \text{(A.6)}\\ Q'_2 &\equiv_S (\nu\, \widetilde{d})([\![h' :: R]\!] \mid h' \rhd (H_0, H') \mid h \rhd K \mid [\![A']\!]H') \,. & \text{(A.7)} \end{aligned}$$

Recall by (A.2) that $H = H' \cup K$. Therefore, by Lemma 5.5.4, we deduce that

$$h' \rhd (H_0, H') \mid h \rhd K \gtrsim h' \rhd (H_0, H) \mid h \rhd K \,.$$

From this and (A.7) we get

$$Q'_2 \gtrsim (\nu\, \widetilde{d})([\![h' :: R]\!] \mid h' \rhd (H_0, H) \mid h \rhd K \mid [\![A']\!]H'). \quad \text{(A.8)}$$

Finally, define $C[\cdot] \stackrel{def}{=} (\nu\, \widetilde{d})([\![h' :: R]\!] \mid h' \rhd (H_0, H) \mid [\,.\,])$, $T_1 \stackrel{def}{=} h \rhd K \mid [\![A']\!]H$ and $T_2 \stackrel{def}{=} h \rhd K \mid [\![A']\!]H'$. The results in (A.6) and (A.8) show that $Q'_1 \equiv_S C[T_1]$ and that $Q'_2 \gtrsim C[T_2]$; this suffices to prove (A.3), since $\equiv_S$ implies $\sim$.

Case 2: μ is an output.

This case is similar to the previous one.

Case 3: $\mu = \tau$.

If $[\![A]\!]H \xrightarrow{\tau} P_1$, then, by Lemma A.1.5(2), one of the following holds:

(a) there is A' s.t. $P_1 \equiv_S [\![A']\!]H$ and also $[\![A]\!]H' \xrightarrow{\tau} P_2 \equiv_S [\![A']\!]H'$, or

(b) there are $\widetilde{d}$, $A', K_{0i} \subseteq \mathcal{K}(A)$ $(i \in \{1,2\})$, R and h_0 fresh s.t.

$$P_1 \equiv_S (\nu\, h_0)\big(\, h_0 \rhd (K_{01}, H) \mid h_0 \rhd (K_{02}, H) \mid (\nu\, \widetilde{d})([\![A']\!]H \mid [\![h_0 :: R]\!])\big)$$

and also

$$\begin{aligned} [\![A]\!]H' &\xrightarrow{\tau} P_2 \\ &\equiv_S (\nu\, h_0)(\, h_0 \rhd (K_{01}, H') \mid h_0 \rhd (K_{02}, H') \mid \\ &\qquad (\nu\, \widetilde{d})([\![A']\!]H' \mid [\![h_0 :: R]\!])\,) \,. \end{aligned}$$

In either cases, using rule `Par` , we get

$$Q_2 = h \rhd K \mid [\![A]\!]H' \xrightarrow{\tau} h \rhd K \mid P_2 \stackrel{def}{=} Q'_2 \,.$$

If (a) holds, then (A.3) is immediately validated, since the pair (Q_1', Q_2') belongs to $\mathcal{R}$, up to $\equiv_S$. Let us consider case (b). By definitions of Q_1', Q_2', P_1 and P_2, we have $Q_1' = h \rhd K \mid P_1$, where the latter is in turn equal to:

$$h \rhd K \mid (\nu\, h_0)\big(\, h_0 \rhd (K_{01}, H) \mid h_0 \rhd (K_{02}, H) \mid (\nu\, \widetilde{d})([\![A']\!]H \mid [\![h_0 :: R]\!])\big)$$

and similarly $Q_2' = h \rhd K \mid P_2$, where the latter is in turn equal to:

$$h \rhd K \mid (\nu\, h_0)\big(\, h_0 \rhd (K_{01}, H') \mid h_0 \rhd (K_{02}, H') \mid (\nu\, \widetilde{d})([\![A']\!]H' \mid [\![h_0 :: R]\!])\big)\,.$$

Extruding the scope of $\nu\, \widetilde{d}$ and $(\nu\, h_0)$, and commuting the order of some components, gives that Q_1' is structurally congruent to:

$$(\nu\, h_0\widetilde{d})\big([\![h_0 :: R]\!] \mid h_0 \rhd (K_{01}, H) \mid h_0 \rhd (K_{02}, H) \mid h \rhd K \mid [\![A']\!]H\big) \tag{A.9}$$

and, similarly, that Q_2' is structurally congruent to:

$$(\nu\, h_0\widetilde{d})\big([\![h_0 :: R]\!] \mid h_0 \rhd (K_{01}, H') \mid h_0 \rhd (K_{02}, H') \mid h \rhd K \mid [\![A']\!]H'\big)\,. \tag{A.10}$$

Since $H = H' \cup K$, by Lemma 5.5.4 (applied twice), we derive

$$\begin{array}{l} h_0 \rhd (K_{01}, H') \mid h_0 \rhd (K_{02}, H') \mid h \rhd K \quad \gtrsim \\ h_0 \rhd (K_{01}, H) \mid h_0 \rhd (K_{02}, H) \mid h \rhd K\,. \end{array}$$

From this and (A.10) we get

$$Q_2' \gtrsim (\nu\, h_0\widetilde{d})\big([\![h_0 :: R]\!] \mid h_0 \rhd (K_{01}, H) \mid h_0 \rhd (K_{02}, H) \mid h \rhd K \mid [\![A']\!]H'\big)\,. \tag{A.11}$$

Finally, define

$$\begin{array}{rcl} C[\cdot] & \stackrel{def}{=} & (\nu\, h_0\widetilde{d})\big([\![h_0 :: R]\!] \mid h_0 \rhd (K_{01}, H) \mid h_0 \rhd (K_{02}, H) \mid [\,.\,]\big) \\ T_1 & \stackrel{def}{=} & h \rhd K \mid [\![A']\!]H \\ T_2 & \stackrel{def}{=} & h \rhd K \mid [\![A']\!]H'\,. \end{array}$$

The results in (A.9) and (A.11) show that $Q_1' \equiv_S C[T_1]$ and that $Q_2' \gtrsim C[T_2]$, thus proving (A.3).

$\square$

Now Lemma 5.6.2, whose assertion is:
For any K_0, K_i, $i \in I$, $h \notin K_0 \cup (\cup_{i\in I} K_i)$, A:

$$B \stackrel{def}{=} (\nu\, h)(\Pi_{i\in I}\, h \rhd K_i \mid [\![A]\!](K_0, h)) \gtrsim [\![A]\!](K_0, \cup_i K_i) \stackrel{def}{=} A\,.$$

Proof of Lemma 5.6.2. The proof of this lemma only differ from that of Lemma 5.6.1 because it uses Lemma 5.5.6 in place of of Lemma 5.5.4. $\square$

Appendix B

Saturation and Cancelation

To ease the reading of this appendix, we recall the definitions of Cause-Sets and Saturation.

Definition B.1.6 (Process cause-sets, Definition 5.7.1) *If A is a causal process, then $\mathcal{CS}(A)$ is the set of sets of causes inductively defined as follows:*

$$\begin{array}{rcl}\mathcal{CS}(P) & \stackrel{def}{=} & \{\emptyset\} \\ \mathcal{CS}(K :: A) & \stackrel{def}{=} & \{K \cup H \,|\, H \in \mathcal{CS}(A)\} \\ \mathcal{CS}(\nu\, a\, A) & \stackrel{def}{=} & \mathcal{CS}(A) \\ \mathcal{CS}(A_1 \mid A_2) & \stackrel{def}{=} & \mathcal{CS}(A_1) \cup \mathcal{CS}(A_2)\,.\end{array}$$

Definition B.1.7 (Saturation, Definition 5.7.2) *A causal process A is* saturated *for a product of wires $W \stackrel{def}{=} \Pi_{i \in I}\, k_i \rhd K_i$ if for each $H \in \mathcal{CS}(A)$ and $i \in I$, $k_i \in H$ implies $K_i \subseteq H$.*

To prove the Saturation Lemma, we use Lemma B.1.8 below on cause-sets of processes. Item (1) says that all sets of causes that appear in causal transitions of a process are also in the cause-sets of that process; item (2) and (3) relate the cause-sets of a process to those of its derivatives; finally, item (4) represents a generalization of item (1) to weak transitions.

Lemma B.1.8 *Let A be a causal term.*

1. *If for some μ, k, A', $A \xrightarrow[K;k]{\mu} A'$ then $K \in \mathcal{CS}(A)$.*
2. *If $A \xrightarrow[K;k]{\mu} A'$ with $k \notin \mathcal{K}(A)$, and $H \in \mathcal{CS}(A')$, then:*
   - *(a) if $k \notin H$, then $H \in \mathcal{CS}(A)$;*
   - *(b) if $k \in H$, then $H = K \cup \{k\}$.*

3. *Suppose* $A \xrightarrow{\tau} A'$ *and* $H \in \mathcal{CS}(A')$. *Then either* $H \in \mathcal{CS}(A)$ *or* $H = H_1 \cup H_2$, *with* $H_1, H_2 \in \mathcal{CS}(A)$.

4. *If for some* μ, k, A', $A \overset{\mu}{\underset{K;k}{\Longrightarrow}} A'$ *then* $K = H_1 \cup \cdots \cup H_n$, *for some* $n > 0$ *and* $H_1, \ldots, H_n \in \mathcal{CS}(A)$.

PROOF:

1. A simple transition induction on $A \overset{\mu}{\underset{K;k}{\longrightarrow}} A'$.

2. By transition induction on $A \overset{\mu}{\underset{K;k}{\longrightarrow}} A'$; we only consider the non-trivial cases, namely `C-Par` and `C-Cau` .

   Case a: `C-Par` rule.

   The last rule applied is of the form

$$\texttt{C-Par} \ : \frac{A_1 \ \overset{\mu}{\underset{K;k}{\longrightarrow}} \ A_1'}{A = A_1 \mid A_2 \ \overset{\mu}{\underset{K;k}{\longrightarrow}} \ A_1' \mid A_2 \overset{def}{=} A'} .$$

   If $H \in \mathcal{CS}(A') = \mathcal{CS}(A_1') \cup \mathcal{CS}(A_2')$ then, by definition, either $H \in \mathcal{CS}(A_1')$ or $H \in \mathcal{CS}(A_2)$. If $H \in \mathcal{CS}(A_2)$ then $k \notin H$ and, since A_2 is a component of A, also $H \in \mathcal{CS}(A)$. Suppose $H \in \mathcal{CS}(A_1')$. If $k \notin H$ then, by inductive hypothesis, $H \in A_1$. Hence $H \in \mathcal{CS}(A)$ since A_1 is a component of A. Suppose now $k \in H$: by inductive hypothesis, $H = K \cup \{k\}$, that proves the claim.

   Case b: `C-Cau` rule.

   The last rule applied is of the form

$$\texttt{C-Cau} \ : \frac{A_0 \ \overset{\mu}{\underset{K_0;k}{\longrightarrow}} \ A_0'}{A = K' :: A_0 \ \overset{\mu}{\underset{K;k}{\longrightarrow}} \ K' :: A_0' \overset{def}{=} A'}$$

   where $K = K' \cup K_0$. If $H \in \mathcal{CS}(A')$ then, by definition, $H = K' \cup H'$, for some $H' \in \mathcal{CS}(A_0')$. If $k \notin H$, then, by inductive hypothesis, $H' \in \mathcal{CS}(A_0)$ and hence also $H \in \mathcal{CS}(A)$. If $k \in H$, then it must be $k \in H'$. By inductive hypothesis, $H' = K_0 \cup \{k\}$. To sum up, we have: $H = K' \cup H' = K' \cup K_0 \cup \{k\} = K \cup \{k\}$, which is our claim.

3. The proof goes by transition induction. All cases are trivial, except the case when `C-Com` is the last rule applied, where the last step of the derivation is of the form

$$\texttt{C-Com} \ : \frac{A_1 \ \overset{\mu_1}{\underset{K_1;k}{\longrightarrow}} \ A_1', \ A_2 \ \overset{\mu_2}{\underset{K_2;k}{\longrightarrow}} \ A_2'}{A = A_1 \mid A_2 \ \xrightarrow{\tau} \ (\nu\, \tilde{c})(A_1'[k \leadsto K_2] \mid A_2'[k \leadsto K_1]) \overset{def}{=} A'}$$

with $k \notin \mathcal{K}(A_1, A_2)$. Let $H \in \mathcal{CS}(A'_1[k \rightsquigarrow K_2])$ (the case when $H \in \mathcal{CS}(A'_2[k \rightsquigarrow K_1])$ is perfectly symmetrical). It must be $H = H_0[k \rightsquigarrow K_2]$, for some $H_0 \in \mathcal{CS}(A'_1)$. Now, there are two cases, namely $k \notin H_0$ and $k \in H_0$. If $k \notin H_0$, then, from item (2) of this lemma, $H_0 \in \mathcal{CS}(A_1)$ and furthermore $H = H_0$, which proves our claim. If $k \in H_0$, then, from item (2) of this lemma, $H_0 = K_1 \cup \{k\}$; since $k \notin K_1$, this implies $H = K_1 \cup K_2$; but, from item (1) of this lemma, $K_1, K_2 \in \mathcal{CS}(A)$; this proves the claim.

4. Using item (3) of this lemma, one can prove, by induction on m, that

$$\begin{gathered}\text{if } A \xrightarrow{\tau^m} A',\ m \geq 0, \text{ and } H \in \mathcal{CS}(A'), \text{ then}\\ H = H_1 \cup \cdots \cup H_n \text{ for some } H_1, \ldots, H_n \in \mathcal{CS}(A).\end{gathered} \tag{B.1}$$

Now, suppose $A \underset{K;k}{\overset{\mu}{\Longrightarrow}} A'$; this means that $A \xrightarrow{\tau^m} A'' \underset{K;k}{\overset{\mu}{\longrightarrow}} A''' \Longrightarrow A'$, for some m, A'' and A'''. Now, the thesis follows using item (1) of this lemma and (B.1).

□

We now come to the proof of the Saturation Lemma; we first recall its assertion: Let A be a causal process saturated for the product of wires $\Pi_{i \in I}\, k_i \rhd K_i$. Then:

1. If $A \underset{K;k}{\overset{\mu}{\longrightarrow}} A'$ with k fresh, then A' is saturated for $k \rhd K \mid \Pi_{i \in I}\, k_i \rhd K_i$.
2. If $A \xrightarrow{\tau} A'$, then A' is saturated for $\Pi_{i \in I}\, k_i \rhd K_i$.
3. If $A \underset{K;k}{\overset{\mu}{\Longrightarrow}} A'$ with k fresh, then A' is saturated for $k \rhd K \mid \Pi_{i \in I}\, k_i \rhd K_i$.

Proof of the Saturation Lemma.

1. Let $H \in \mathcal{CS}(A')$ and $h \in H \cap (\{k_i\ :\ i \in I\} \cup \{k\})$. We have to show that $K \subseteq H$ in case $h = k$, and that $K_i \subseteq H$ in case $h = k_i$. This fact follows from Lemma B.1.8(1) and (2) and from the fact that A is saturated for $\Pi_{i \in I}\, k_i \rhd K_i$.
2. Follows from Lemma B.1.8(3) and from the definition of saturation.
3. It holds that $A \Longrightarrow A'' \underset{K;k}{\overset{\mu}{\longrightarrow}} A''' \Longrightarrow A'$, for some A'' and A'''; the thesis is obtained by (repeatedly) using item (2) of this lemma on the transition $A \Longrightarrow A''$, then item (1) on the transition $A'' \underset{K;k}{\overset{\mu}{\longrightarrow}} A'''$, then item (2) again (repeatedly) on the transition $A''' \Longrightarrow A'$.

□

We are left with the proof of the Cancelation lemma. For this, we use the following auxiliary technical result:

Lemma B.1.9 *Let A, B be saturated for $W \stackrel{def}{=} \Pi_{i \in I}\, k_i \rhd K_i$. Suppose that for some μ*

$$A \underset{H';\,k_0}{\stackrel{\mu}{\Longrightarrow}} A' \text{ and } B \underset{H'';\,k_0}{\stackrel{\mu}{\Longrightarrow}} B'$$

with k_0 fresh and, furthermore, suppose that

$$W \mid\ k_0 \rhd H' \ \mid [\![A']\!] \approx W \mid\ k_0 \rhd H'' \ \mid [\![B']\!]\,.$$

Then $H' = H''$.

PROOF: We prove that $H' \subseteq H''$ (symmetrically, it will be $H'' \subseteq H'$). Given $k \in H'$, we show that $k \in H''$ as well. Let

$$R \stackrel{def}{=} W \mid\ k_0 \rhd H' \ \mid [\![A']\!]$$

and

$$S \stackrel{def}{=} W \mid\ k_0 \rhd H'' \ \mid [\![B']\!].$$

Since $k \in H'$, for all v there is R' s.t.

$$R \stackrel{k_0\langle v\rangle}{\longrightarrow} \stackrel{\overline{k}\langle v\rangle}{\longrightarrow} R'\,.$$

Since $R \approx S$, there is S' s.t.

$$S \stackrel{k_0\langle v\rangle}{\Longrightarrow} \stackrel{\overline{k}\langle v\rangle}{\Longrightarrow} S'\,.$$

We now analyse the above sequence of transitions. For some S_1,S_2,S_3,S_4 and S_5 we have

$$S \Longrightarrow S_1 \stackrel{k_0\langle v\rangle}{\longrightarrow} S_2 \Longrightarrow S_3 \stackrel{\overline{k}\langle v\rangle}{\longrightarrow} S_4 \Longrightarrow S_5 = S'\,.$$

We now argue on the form of S_1, S_2 and S_3. From Remark 5.6.12, it follows that $W \mid\ k_0 \rhd H''$ does not contribute to the weak transition $S \Longrightarrow S_1$; in other words, there exists P s.t. $[\![B']\!] \Longrightarrow P$ and

$$S_1 = W \mid\ k_0 \rhd H'' \ \mid P\,.$$

From Remark 5.6.12 and from the fact that $k_0 \notin \{k_i \,:\, i \in I\}$, we get that the transition $S_1 \stackrel{k_0\langle v\rangle}{\longrightarrow} S_2$ originates from $k_0 \rhd H''$; thus, letting $Z \stackrel{def}{=} W \mid\ k_0 \rhd H'' \ \mid \Pi_{k'' \in H''}\, !\, \overline{k''}\langle v\rangle$, it holds that

$$S_2 = Z \mid P\,.$$

Again by Remark 5.6.12 we know that processes Z and P, as well as any τ-derivative of them, cannot interact with one another. Hence the transition $S_2 \Longrightarrow S_3$ can be decomposed into two independent sequences of transitions from Z and P; formally, there exist V and P' s.t. $Z \Longrightarrow V$, $P \Longrightarrow P'$ and

$$S_3 = V \mid P'\,.$$

Since $[\![B']\!] \Longrightarrow P'$, from Remark 5.6.12 it follows that the transition $S_3 \stackrel{\overline{k}\langle v\rangle}{\longrightarrow} S_4$ originates from V (i.e. $V \stackrel{\overline{k}\langle v\rangle}{\longrightarrow} V'$, for some V'). This and the fact that V is a τ-derivative of Z imply that $Z \stackrel{\overline{k}\langle v\rangle}{\Longrightarrow} V'$, and therefore, by definition of Z we get that: Either $k \in H''$, or there is $i \in I$ s.t. $k_i \in H''$ and $k \in K_i$ (so that the transition $\stackrel{\overline{k}\langle v\rangle}{\longrightarrow}$ can occur after an interaction of $!\,\overline{k_i}\langle v\rangle$ with $k_i \triangleright K_i$). Now, if $k \in H''$ we have finished our proof. Assume therefore that, for some i, $k_i \in H''_\mu$ and $k \in K_i$; we show that also $k \in H''$ holds. From Lemma B.1.8(4) and from $B \underset{H'';k_0}{\Longrightarrow} B'$, it follows that $H'' = H_1 \cup \cdots \cup H_n$, for some n, with the $H_j \in \mathcal{CS}(B)$, $1 \leq j \leq n$; hence for some $j \in \{1, \ldots, n\}$, we have $k_i \in H_j$; but, since by hypothesis B is saturated for W, it holds that $K_i \subseteq H_j$; since $k \in K_i$ and $H_j \subseteq H''$, we conclude that $k \in H''$, which proves our claim. $\square$

We are now ready to prove the Cancelation Lemma. We first recall its assertion: Let A be saturated for $k \triangleright K$ and B be saturated for $k \triangleright K'$. Then

$$k \triangleright K \mid [\![A]\!] \approx k \triangleright K' \mid [\![B]\!] \text{ implies } K = K' \text{ and } [\![A]\!] \approx [\![B]\!].$$

Proof of Cancelation Lemma 5.7.4. If $k \triangleright K \mid [\![A]\!] \approx k \triangleright K' \mid [\![B]\!]$, then it must be $K = K'$, otherwise the two processes would be distinguishable through interactions at cause names (by Remark 5.6.12 processes $[\![A]\!]$, $[\![B]\!]$, and τ-derivatives of them, cannot perform actions at a cause name).

The difficult part is to prove $[\![A]\!] \approx [\![B]\!]$. For this, we show that

$$\begin{array}{rcll} \mathcal{R} & = & \{([\![A\rho]\!], [\![B\rho]\!]) : & \rho \text{ is a finite cause substitution} \\ & & & \text{and for some } I \text{ and } k_i, K_i \ \ (i \in I), \\ & & & A \text{ and } B \text{ are saturated for } \Pi_{i\in I}\, k_i \triangleright K_i \\ & & & \text{and } \Pi_{i\in I}\, k_i \triangleright K_i \mid [\![A]\!] \approx \Pi_{i\in I}\, k_i \triangleright K_i \mid [\![B]\!]\,\} \end{array}$$

is a weak bisimulation up to $\gtrsim$ and up to context. Thus the thesis will follow by letting ρ equal to the empty cause substitution and letting $\Pi_{i\in I}\, k_i \triangleright K_i$ equal to $k \triangleright K$ in the definition of $\mathcal{R}$. In the rest of the proof, we abbreviate $\Pi_{i\in I}$ as Π_i. We only show how the moves of $[\![A\rho]\!]$ are matched by $[\![B\rho]\!]$, since $\mathcal{R}$ is symmetrical. We consider the case of weak input transitions; the cases of output and invisible transitions are similar or easier. Thus suppose:

$$[\![A\rho]\!] \stackrel{a\langle\widetilde{b}k\rangle}{\Longrightarrow} P\,. \tag{B.2}$$

We have to find processes Q, P' and Q' and a static context $C[\,.\,]$ s.t.

$$P \gtrsim C[P']\,, \quad [\![B\rho]\!] \stackrel{a\langle\widetilde{b}k\rangle}{\Longrightarrow} Q \gtrsim C[Q'] \text{ and } (P', Q') \in \mathcal{R}\,. \tag{B.3}$$

From Proposition 5.6.11(2), we know that, for some H and A',

$$P \gtrsim k \triangleright H \mid [\![A']\!] \tag{B.4}$$

and

$$A\rho \overset{a\langle\widetilde{b}\rangle}{\underset{H;k}{\Longrightarrow}} A'. \tag{B.5}$$

Take a fresh $k_0 \in \mathcal{K}$; since k_0 is fresh, it holds that $k_0 \notin \mathcal{K}(A, B) \cup (\cup_{i\in I} K_i) \cup \{k_i : i \in I\}$ and, moreover, $k'\rho = k_0$ iff $k' = k_0$ (note that such a fresh k_0 exists because the nominated sets of causes are finite and ρ is a finite cause substitution).

By applying first Lemma 5.3.8 and then Lemma 5.3.7 to (B.5), we infer that

$$A \overset{a\langle\widetilde{b}\rangle}{\underset{H';k_0}{\Longrightarrow}} A'' \tag{B.6}$$

with

$$H'\rho = H \text{ and } A''\rho[k_0 \rightsquigarrow k] = A'. \tag{B.7}$$

Applying again Proposition 5.6.11 to (B.6), we get

$$[\![A]\!] \overset{a\langle\widetilde{b}k_0\rangle}{\Longrightarrow} P_1 \gtrsim k_0 \rhd H' \mid [\![A'']\!] \tag{B.8}$$

from which, using rule `Par` , we infer

$$\Pi_i\, k_i \rhd K_i \mid [\![A]\!] \overset{a\langle\widetilde{b}k_0\rangle}{\Longrightarrow} \Pi_i\, k_i \rhd K_i \mid P_1 \overset{def}{=} R\,. \tag{B.9}$$

Since, by hypothesis, $\Pi_i\, k_i \rhd K_i \mid [\![A]\!] \approx \Pi_i\, k_i \rhd K_i \mid [\![B]\!]$, there is S s.t.

$$\Pi_i\, k_i \rhd K_i \mid [\![B]\!] \overset{a\langle\widetilde{b}k_0\rangle}{\Longrightarrow} S \approx R\,. \tag{B.10}$$

Since $\Pi_i\, k_i \rhd K_i$ cannot perform an action at a and, moreover, $\Pi_i\, k_i \rhd K_i$ cannot interact with any T such that $[\![B]\!] \Longrightarrow T$ or $[\![B]\!] \overset{a\langle\widetilde{b}k_0\rangle}{\Longrightarrow} T$ (by Remark 5.6.12), the term $\Pi_i\, k_i \rhd K_i$ does not contribute to the above weak transition. That is, there exists Q_1 such that

$$[\![B]\!] \overset{a\langle\widetilde{b}k_0\rangle}{\Longrightarrow} Q_1 \tag{B.11}$$

and

$$S = \Pi_i\, k_i \rhd K_i \mid Q_1\,. \tag{B.12}$$

From (B.11) and Proposition 5.6.11(2), we deduce that, for some B'' and H'',

$$B \overset{a\langle\widetilde{b}\rangle}{\underset{H'';k_0}{\Longrightarrow}} B'' \tag{B.13}$$

with

$$Q_1 \gtrsim k_0 \rhd H'' \mid [\![B'']\!]. \tag{B.14}$$

Now, since $\gtrsim$ implies $\approx$ and parallel composition preserves $\approx$, we have (next to each step we put the equation number that justifies it):

$$\begin{array}{rcll} \Pi_i\ k_i \rhd K_i\ \mid\ k_0 \rhd H'\ \mid [\![A'']\!] & \approx & \Pi_i\ k_i \rhd K_i\ \mid P_1 & \text{(B.8)} \\ & = & R & \text{(B.9)} \\ & \approx & S & \text{(B.10)} \\ & = & \Pi_i\ k_i \rhd K_i\ \mid Q_1 & \text{(B.12)} \\ & \approx & \Pi_i\ k_i \rhd K_i\ \mid\ k_0 \rhd H''\ \mid [\![B'']\!] & \text{(B.14)}\,. \end{array} \tag{B.15}$$

Thus, from (B.6), (B.13) and (B.15), we have found that $A \overset{a\langle\widetilde{b}\rangle}{\underset{H';\,k_0}{\Longrightarrow}} A''$, that $B \overset{a\langle\widetilde{b}\rangle}{\underset{H'';\,k_0}{\Longrightarrow}} B''$ and that

$$\Pi_i\ k_i \rhd K_i\ \mid\ k_0 \rhd H'\ \mid [\![A'']\!] \approx \Pi_i\ k_i \rhd K_i\ \mid\ k_0 \rhd H''\ \mid [\![B'']\!]\,.$$

Moreover, by definition of $\mathcal{R}$, we know that A and B are saturated for $\Pi_i\ k_i \rhd K_i$. From these facts, using Lemma B.1.9 we deduce that $H' = H''$.

Having proved that $H' = H''$, we can now conclude the proof of this lemma. Applying Lemma 5.3.8 and Lemma 5.3.6 to the transition in (B.13) we get

$$B\rho \overset{a\langle\widetilde{b}\rangle}{\underset{H''\rho;\,k}{\Longrightarrow}} B'$$

with $B' = B''\rho[k_0 \rightsquigarrow k]$. Since $H' = H''$ and $H'\rho = H$ (by B.7), the above transition can be rewritten as

$$B\rho \overset{a\langle\widetilde{b}\rangle}{\underset{H;\,k}{\Longrightarrow}} B'\,.$$

Moreover, by Proposition 5.6.11(1) we also have

$$[\![B\rho]\!] \overset{a\langle\widetilde{bk}\rangle}{\Longrightarrow} Q \gtrsim\ k \rhd H\ \mid [\![B']\!]\,. \tag{B.16}$$

We can now show that transition (B.16) matches the one by $[\![A\rho]\!]$ in (B.2). For this, we have to exhibit two processes P', Q' and a static context $C[\,.\,]$ s.t. (B.3) is satisfied. We set $P' \overset{def}{=} [\![A']\!]$, $Q' \overset{def}{=} [\![B']\!]$ and $C[\,.\,] \overset{def}{=}\ k \rhd H\ \mid [\,.\,]$. We first show that $(P', Q') \in \mathcal{R}$. We recall that $A' = A''\rho[k_0 \rightsquigarrow k]$, that $B' = B''\rho[k_0 \rightsquigarrow k]$ and that $\rho[k_0 \rightsquigarrow k]$ is a finite cause substitution. From the Saturation Lemma, applied to the transitions in (B.6) and (B.13), and since $H' = H''$, we deduce that A'' and B'' are saturated for $\Pi_i\ k_i \rhd K_i \mid k_0 \rhd H'$. These facts and (B.15) demonstrate that $(P', Q') \in \mathcal{R}$. Finally, (B.4) and (B.16) demonstrate that $P \gtrsim C[P']$ and $Q \gtrsim C[Q']$; hence (B.3) is satisfied, which closes up the bisimulation. $\square$

Ai miei genitori, Maria e Carmelo