Aalborg University Centre
Institute of Electronic Systems
M. Sc. Thesis in Computer Science

Verification of CCS-processes

by

Michael Hillerström

Abstract

The main achievement of this thesis is the construction and (rigorous) verification of an operational based inference system for reasoning about bisimulation equivalence and inequivalence.

The system is implemented in PROLOG and is capable of producing a bisimulation containing a given pair of (CCS-)processes in case they are equivalent. More important, the system is also capable of giving a modal property as *an argument for inequivalence* in case the processes are not equivalent. This makes the system very suitable as an automatic tool aiding verification and debugging of systems. The usefulness is demonstrated through several examples.

A notion of *extended bisimulation* is introduced as a faithful extension of ordinary bisimulation supplying information on how the pairs of a bisimulation actually fit together.

An extended system arguing for extended bisimulation equivalence and inequivalence is constructed and implemented. The extended PROLOG system is demonstrated through some examples.

Aalborg University Centre
Institute of Electronic Systems
M. Sc. Thesis in Computer Science
January 12, 1987

Verification of CCS-processes

Author: Michael Hillerström

Supervisor: D. Ph. Kim G. Larsen

Period: September 1, 1986 — January 12, 1987

Preface

The work presented here, performed during the autumn of 1986, is my M. Sc. Thesis in Computer Science. Througout this period D. Ph. Kim G. Larsen has been my supervisor.

As presented, the work is intended for fellow students and other people interested in construction, verification, and implementation of automatic tools aiding verification of (CCS-)processes.

Figures, theorems, definitions etc. are numberes sequentially within each section (e.g. theorem 3.1.2 is theorem two of section one in chapter three).

Acknowledgements

First of all I would like to thank my supervisor D. Ph. Kim G. Larsen for his invaluable guidance, his constant encouragement, and enthusiasm throughout this period.

I would also like to thank my family and friends for being helpful, supporting, and understanding during my time of studies. It would certainly not have been possible without you.

A special word of gratitude is directed towards Henrik I. Christensen for making my drawings look much better than they deserve.

Aalborg January 12, 1987

Michael Hillerström

Contents

Chapter 1

Introduction

In the last 10–15 years much effort has been made in order to provide ways of formalizing the semantics of systems. A formal semantics provides us with three things. Firstly, a yardstick with which to compare different implementations of a language; secondly, a framework for formal proofs of correctness of individual programs; and thirdly, the basis of a method for automatically generating implementations (e.g. Ph. D. Lawrence Paulsons's system [Paulson]).

Of course, there are several alternative approaches to formal semantics. There are mainly three alternatives; Axiomatic Semantics, Operational Semantics, and Denotational Semantics. The axiomatic approach deals with relating predicates which are true before execution to predicates being true afterwards for each type of construct. This is an elegant basis for program correctness proofs of individual programs in imperative languages. An operational semantics specifies the language by defining an interpreter for the abstract syntax of the language, i.e. this approach gives good pointers for methods of implementation, however, program proving implies a total execution of the program 'by hand.' The denotational semantics defines what each grammatical unit denotes in some well-defines mathematical model. Programs are normally modeled as computable functions from the domain of input values to the domain of output values. Also, the meaning of a particular syntactic construct is defined in terms of the meanings of its immediate sub-components [Gordon].

A major goal of system design and implementation is to achieve semantic theories which support modular design and verification of systems. It should be possible given the specification of components to deduce whether the components in a particular context will implement (satisfy) some overall specification.

Denotational semantics supports the requirement of program development and verification. However, for concurrent systems this semantic theory is inadequate. A concurrent system can have many interesting properties which cannot be described by an input–output function semantics (e.g. deadlock). Also, the purpose of a concurrent system may be entirely different from that of computing a function (e.g. a non-terminating program, such as an operation system,, is normally considered a useful system). Even if we only consider input–output function behaviour of concurrent systems, the requirement of modularity would fail to hold; there is no way of predicting the input–output behaviour of its sub-components [Larsen][Milner]. So, in order to determine the systems overall behaviour, it seems that further information about possible intermediate states of the components is needed.

[Milne,Milner] solved the problem for a Calculus of Communicating Systems by choosing an operational semantics. Based on this operational semantics several equivalences between non-deterministic and concurrent processes have been proposed in order to capture different notions of the properties of a process. The proposed equivalences are useful in the design and verification of concurrent systems as each equivalence provide a notion of correctness of an implementation, *IMP*, with respect to some specification, *SPEC*; providing the correctness of *IMP* simply consists of proving the equivalence, $\mathit{SPEC} \equiv \mathit{IMP}$ (where $\equiv$ is the equivalence under consideration).

In general, though, the implementation, *IMP*, is derived through successive refinements, IMP_k, of the specification, *SPEC* (stepwise refinement). Thus, in order to guarantee $\mathit{SPEC} \equiv \mathit{IMP}$, each refinement, IMP_k (where IMP_{m+1} is increasingly more refined than IMP_m, $\mathit{IMP}_0 = \mathit{SPEC}$ and $\mathit{IMP}_n = \mathit{IMP}$), must be proved to be equivalent to its predecessor, IMP_{k-1} (see figure 1.1). For each IMP_k, the refinement consists of a small refinement of IMP_{k-1} (see figure 1.2), where some small part p is replaced by a more concrete device q, keeping the rest of the system unaffected. In order to complete the refinement step we must prove that $\mathit{IMP}_{k-1} \equiv \mathit{IMP}_k$. The direct approach to this involves proving the equivalence by the definition of the equivalence function. However, this will in general cause us not only to prove the equivalence of p and q but actually prove the equivalence of IMP_{k-1} and IMP_k of which p and q are just (very) small parts. This approach seems to give us more work than we are willing to perform.

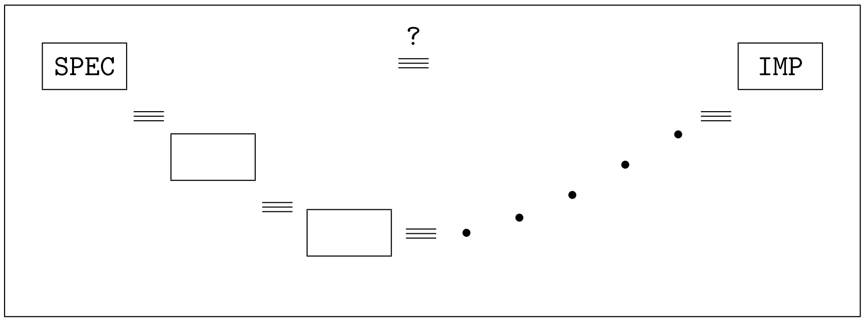


Figure 1.1: Stepwise refinement

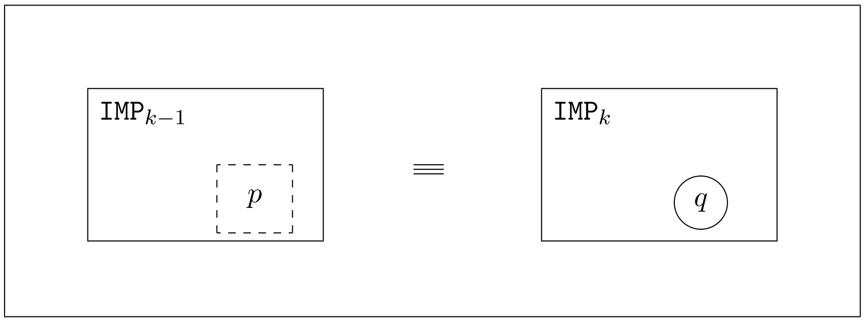


Figure 1.2: Single refinement step

Alternatively, though, by ensuring that the equivalence in question in fact is a congruence with respect to the various process construction operators, it will be sufficient to show $p \equiv q$ in order to establish $IMP_{k-1} \equiv IMP_k$. Unfortunately, experiences with correctness proofs in the framework of Calculus of Communicating Systems have shown that the congruence approach also leads to unnecessarily long and complicated proofs when they succeed

[Prasad].

Through the study of examples it has become clear that computer assistance is essential; not just to ensure the correctness of the proof but even to make the analyses feasible. In January 1986 the Laboratory of Foundations of Computer Science was founded in Edinburgh. The laboratory will be engaged in developing systems for computer assisted formal reasoning in general, including a wide collection of computer based tools for design and verification of concurrent systems [Larsen]. In relation to this project a PROLOG.system for deciding bisimulation equivalence of CCS-processes was developed and implemented as one of many results of [Larsen]. A similar system, developed by K. V. S. Prasad, differs from the one of [Larsen] in that, it will check whether a (by the user) given binary relation on processes constitutes a bisimulation.

Unfortunately, these systems have the drawback of being absolutely uninformative in case the processes under consideration are inequivalent. We want to add such information. Often, when knowing the reason *why* something acts strangely, it is easy to find a 'cure' that will prevent the situations occurring in the future. For example, most programs never do what they are intended to do in the first try. There will always be some situations not catered for. An argument of *why* this situation actually occurred would be of great help in the debugging of the program. Of course, we will not accept all sorts of arguments; the arguments should be of a certain 'quality,' that is:

1. arguments must be correct
2. arguments most be concise in the sense that an argument must not contain redundant or irrelevant information

Clearly, the first requirement is vital. The other requirement is also of great importance if the arguments given should be taken seriously e.g. it is of little use to know that Aalborg is a city of Denmark if you are trying to determine why your coffee machine is malfunctioning. Also, you will not be any happier knowing that the heating element is not heating, that the light on the machine is not lit etc. when you already know that you forgot to insert the plug into the power outlet.

This thesis will mainly be concerned with the construction and verification of a system for finding *bisimulations of CCS-processes*. The system will be based upon the same principles as the PROLOG-system in [Larsen], but in addition to being a theorem prover, our system will also be able to give an explanation when two processes are not equivalent. The explanation given is a *modal property* which only one of the processes enjoys. The constructed system will be implemented in PROLOG and its usefulness will be demonstrated through several examples. Moreover, a notion of *extended bisimulation* will be presented (and implemented) resulting in more informative proofs (bisimulations) being generated by the system and thereby improving the overall usefulness of the system.

In order to reach these goals we will first have to study some of the theory behind CCS and verification of CCS-processes. More specifically, in chapter 2, we will give a short review of the basic theory behind CCS supplemented with results of the related theory of labeled transition systems. In doing so, we will introduce the notions of simulation and bisimulation as means of abstracting the operational behaviour of a process, and as an elegant proof technique for proving the equivalence of processes. Also, a modal characterization of processes identifying the processes with the properties they enjoy will be reviewed.

In chapter 3, the principles of two existing systems will be presented — each taking a different approach towards proving (or disproving) bisimulations equivalence. The first system, a result of a master thesis work [Vestmar,Olesen], tries to find a 'maximal' bisimulation using an application of generalized partitioning problem. The second system [Larsen], in contrast, tries to find a 'minimal' bisimulation using a principle strongly connected to the definition of bisimulation. Our system will be based on the same principles as this system.

Then, in chapter 4, we present our own system for reasoning about the equivalence *and inequivalence* of processes. We will prove soundness and (restricted) completeness of the system.

Furthermore, in chapter 5, we will demonstrate the usefulness of the system through some examples. The extension of the system in order to make it find *weak* bisimulations is briefly discussed and this (much more interesting) system is demonstrated through several examples.

Finally, in chapter 6, the notion of *extended* bisimulations is presented in order to make the theorem proving part of the system more useful. A new extended system for arguing for extended bisimulation equivalence and inequivalence is constructed and implemented. The usefulness of this improved system is demonstrated through some examples.

Chapter 2

Basic theory

2.1 CCS

CSS, which is short for Calculus of Communicating Systems, originate from the work of Robin Milner [Milner]. CCS can be considered as based upon the traditional theory of finite automata. In CCS, however, the external behaviour of a process is considered more important than the internal behaviour. In fact, in CCS one wants to make an abstraction from the internal behaviour.

The calculus is founded on two central ideas, The first being *observation* and the other *synchronized communication*. The aim is to describe a concurrent system fully enough to determine exactly what behaviour will be seen or experienced by an external observer i.e. two systems are indistinguisable if we cannot tell them apart without pulling them apart. (Later on the formal definition of *observational equivalence* and some of its properties will be stated).

This, however, does not prevent us ffrom studying the structure of systems and as every interesting concurrent system is built from independent agents (programs/processes) which communicate, this leads us to synchronized communication. Communication between two agents is regarded as ana indivisable action of the composit system in CCS and the central operation of the algebra of systems is *concurrent composition*, a binary operation which compose two independent agents, allowing them to communicate. Note, when we compose two agents it is synchronization of their mutual communication which determines the composite; we treat their independent actions as occuring in arbitrary order but *not* simultaneously.

These two ideas can in fact be unified as on one hand the only way to observe a system is to communicate with it, which makes the observer and system together a larger system. On the other hand, to place two components in communication (i.e. to compose them) is just to let them obseve each other. If observing and communicating are the same, it follows that one cannot observe a system without its participation.

Since a program or a system in CCS is just a term of the calculus, the structure of the program or system (its *intension*) is reflected in the structure of the term. We will now turn to the syntax and operational semantics of terms in CCS.

2.1.1 Syntax and operational semantics

In this section the syntax and operational semantics of CCS-terms (henceforth called agnts, CCS-processes or processes for short) in terms of a labelled transition system [Plotkin] will be introduced formally. We will restrict this presentation to a subset of CCS as this suffices for our purposes. A full treatment of CCS can be found in [Milner].

The system of CCS-processes is closed under *action prefixing* and *binary summation*. Apart from this, CCS-processes are built from a number of operations, namely the *parallel operator*, the *restriction operator*, and the *renaming operator*.

The *parallel operator*, $|$, represents the parallel composition of two processes, enabeling communication to occur between them. Furthermore, the parallel operator also allows the processes behaviours to interleave freely.

Together with the $|$-operator a structure on the *action set* Act is introduced; it is assumed that Act is the disjoint union of these sets Δ, $\overline{\Delta}$ and $\{\tau\}$. Δ is a (fixed) set of *names* and $\overline{\Delta}$ is a set of *co-names*, disjoint from Δ and in bijection with it, the bijection being

$$\overline{} : \Delta \mapsto \overline{\Delta} \tag{2.1}$$

and $\overline{\alpha}$ is the co-name of α, and the inverse bijection is $\overline{\overline{\alpha}} = \alpha$. $\alpha \in \Delta$ ($\overline{\alpha} \in \overline{\Delta}$) is also called the *complementary action* of $\overline{\alpha} \in \overline{\Delta}$ ($\alpha \in \Delta$).

Communication between two processes in parallel may then take place if they can perform complementary actions. As a result of the communication the combined system will perform a τ-action (often called an 'invisible', 'silent', or 'internal' action). τ-actions are also used to model nondeterministic bahaviour of agents.

The second sort of operator was the *restriction operator*, $\backslash S$ where $S \subseteq \Delta$, which restricts a process' actions not to include actions belonging to the set $S \cup \overline{S}$. The restriction operator is useful for ensuring that certain communications of processes composed by the $|$-operator occur internally.

The third sort op operator was the *renaming operator*, $[N]$, where

$$N : \Delta \cup \overline{\Delta} \mapsto \Delta \cup \overline{\Delta}, \tag{2.2}$$

and N is assumed to be a bijection, to respect complements, and to preserve τ.

By using the above mentioned operators it is possible to define processes with quite complex behaviours, however, the behaviours will in all cases be finite. In order to obtain infinite behaviour, a form of recursion of processes is needed. This is introduced by the notion of behaviour/process identifiers. We presuppose a set, *Ident*, of *process identifiers*. Each process identifier, X, is defined by a (possibly recursive) rule. This is denoted by a declaration, D, being a mapping

$$D : \mathit{Ident} \mapsto CCS_{\mathit{Ident}}$$

We write $X :=: p_X$ to indicate that $p_X = D(X)$, whenever D is implicitly given. We are now able to introduce the syntax of CCS-processes and it is given in figure 2.1.

In order to obtain an *image-finite* process system (see corollary 2.1.4) a syntactic restriction is imposed on $X :=: p_X$, namely that X should be guarded in p_X i.e. every occurence of X in p_X is within some subexpression $\alpha.p'$ of p_X, where $\alpha \in \mathit{Act}$.

We define the process system $\mathbf{P}$ as the (labelled) transition system $\mathbf{P} = (\mathit{Pr}, \mathit{Act}, \longrightarrow)$, where $\longrightarrow$ is the smallest relation on $\mathit{Pr} \times \mathit{Act} \times \mathit{Pr}$ satisfying the rules in figure 2.2.

$$\begin{array}{rrl} p ::= & NIL \mid & \quad inaction \\ & X \mid & \quad recursion \\ & \alpha.p \mid & \quad prefixing \\ & p + p' \mid & \quad summation \\ & p \mid p' \mid & \quad parallel \\ & p \backslash S \mid & \quad restriction \\ & p[N] \mid & \quad renaming \end{array}$$

$$where\ X \in Ident$$

Figure 2.1: Syntax of CCS-processes.

Definition 2.1.1 *Let $\mathcal{R}$ be a binary relation over the set S. Then $\mathcal{R}$ is image-finite iff for each element $s \in S$ the set*

$$\{t \mid s\mathcal{R}t\}$$

is finite. □

Definition 2.1.2 *A labelled transition system, $\mathbf{S} = (S, A, \longrightarrow)$, is image-finite iff for all elements $a \in A$ the set*

$$\xrightarrow{a} = \{(s,t) \in S \times S \mid s \xrightarrow{a} t\}$$

is image-finite. □

We are now able to state the following proposition.

Proposition 2.1.3 *For all CCS process expressions, p, the set*

$$\{(\alpha, p') \mid p \xrightarrow{\alpha} p'\}$$

is finite.

PROOF: *See [Larsen].* □

Corollary 2.1.4 *The process system* **P** *(over CCS-processes) is image-finite, or equivalently*

$$\forall \alpha \in Act : \xrightarrow{\alpha} = \{(p,q) \mid p \xrightarrow{\alpha} q\}$$

is image-finite. □

Definition 2.1.5 *A set $S \subseteq Pr$ is $\longrightarrow$-closed if and only if*

$$\forall \alpha \in Act : p \in S \ \wedge \ p \xrightarrow{\alpha} p' \Rightarrow p' \in S$$

□

Definition 2.1.6 *Let $\mathbf{P} = (Pr, Act, \longrightarrow)$ be a process system and let Q be a $\longrightarrow$-closed subset of Pr then the restricted process system, $\mathbf{P} \downarrow Q$, is defined as*

$$\mathbf{P} \downarrow Q = (Q, Act_Q, \longrightarrow_Q)$$

where

$$\begin{aligned} Act_Q &= \{\alpha \in Act \mid \exists q \in Q : q \xrightarrow{\alpha}\} \quad \textit{and} \\ \longrightarrow_Q &= \longrightarrow \cap (Q \times Act_Q \times Q) \\ &= \longrightarrow \cap (Q \times Act \times Q) \end{aligned}$$

□

Having defined the syntax and the operational semantics of CCS-processes it will be interesting to be able to determine whether or not two processes show the same behaviour—they posesses the same derivations.

2.2 Simulation, bisimulation, and equivalences

In the following we present the general notion of *simulation* and *bisimulation* as means of abstracting the operational behaviour of a process, and we shall state some of their properties. Bisimulation yields an elegant proof technique for proving equivalence of processes which is superior to the traditional techniques. traditionally, equivalence is proved by algebraic laws. However, from the point of view os automating it turns out that the notion

Action $\alpha.p \xrightarrow{\alpha} p'$

Sum $\dfrac{p_1 \xrightarrow{\alpha} p'}{p_1 + p_2 \xrightarrow{\alpha} p'}$ $\dfrac{p_2 \xrightarrow{\alpha} p'}{p_1 + p_2 \xrightarrow{\alpha} p'}$

Parallel $\dfrac{p_1 \xrightarrow{\alpha} p_1'}{p_1 \mid p_2 \xrightarrow{\alpha} p_1' \mid p_2}$ $\dfrac{p_2 \xrightarrow{\alpha} p_2'}{p_1 \mid p_2 \xrightarrow{\alpha} p_1 \mid p_2'}$

$\dfrac{p_1 \xrightarrow{\alpha} p_1' \quad p_2 \xrightarrow{\overline{\alpha}} p_2'}{p_1 \mid p_2 \xrightarrow{\tau} p_1' \mid p_2'}$

Restriction $\dfrac{p \xrightarrow{\alpha} p'}{p\setminus S \xrightarrow{\alpha} p'\setminus S}$; $\alpha \notin S \cup \overline{S}$

Renaming $\dfrac{p \xrightarrow{\alpha} p'}{p[N] \xrightarrow{N\alpha} p'[N]}$

Recursion $\dfrac{p_X \xrightarrow{\alpha} p}{X \xrightarrow{\alpha} p}$; $X :=: p_X$.

Figure 2.2: Operational semantics for CCS.

of bisimulation yields much more efficiency. The following presentation is strongly inspired by [Larsen].

As apparent from the previous section, processes and their operational behaviour are modeled by labelled transition systems. Let $\mathbf{P} = (Pr, Act, \longrightarrow)$ be such a system. We shall alternatively refer to the transition relation, $\longrightarrow$, of $\mathbf{P}$ as the *derivation relation*. Let p and q be two processes of $\mathbf{P}$. We then say that q simulates p or p *is simulated by* q, if every derivation of p can be matched by a derivation of q in such a way that the simulation property is maintained. Formally this is

Definition 2.2.1 ***[Simulation]***
A simulation $\mathcal{R}$ is a binary relation on $\mathbf{P}$ such that whenever $p\mathcal{R}q$ and $\alpha \in Act$ then

$$p \xrightarrow{\alpha} p' \rightarrow \exists q' : q \xrightarrow{\alpha} q' \wedge p'\mathcal{R}q' \qquad (2.3)$$

A process q is said to simulate *a process p if and only if there exists a simulation $\mathcal{R}$ with $p\mathcal{R}q$. In this case we write $p \preceq q$.* □

Now, for $\mathcal{R} \subseteq Pr^2$ we can define $\mathcal{S}(\mathcal{R}) \subseteq Pr^2$ as

$$\mathcal{S}(\mathcal{R}) = \{(p,q) \in Pr^2 \mid \forall \alpha \in Act : p \text{ and } q \text{ satisfies equation 2.3}\} \qquad (2.4)$$

With this definition in mind the following properties can be stated:

Proposition 2.2.2 *$\mathcal{R} \subseteq Pr^2$ is a simulation iff $\mathcal{R} \subseteq \mathcal{S}(\mathcal{R})$.* □

Proposition 2.2.3 *$\mathcal{S}$ is a monotonic endofunction on the complete lattice of binary relations (over Pr) under inclusion.* □

Using the standard fixed-point result, originally due to Tarski, this implies:

Proposition 2.2.4 *$\mathcal{S}$ has a maximal fixed-point given by*

$$\bigcup\{\mathcal{R} \mid \mathcal{R} \subseteq \mathcal{S}(\mathcal{R})\}.$$

Moreover $\preceq$ equals this fixed-point. □

Proposition 2.2.5 *$\preceq$ is a preorder on Pr^2.*

PROOF: *see [Larsen].* $\square$

It should be noted, that definition 2.2.1 of simulation ordering admits an elegant proof technique: to show that $p \preceq q$ it is sufficient and necessary to find a simulation containing (p, q).

Example 2.2.6 *Let* **P** *be given by the diagram below:*

$$\begin{tikzcd}
 & p_0 \arrow[dl, "a"'] \arrow[dr, "a"] & & & & q_0 \arrow[d, "a"] & \\
p_1 \arrow[d, "b"'] & & p_2 \arrow[d, "c"] & & & q_1 \arrow[dl, "b"'] \arrow[dr, "c"] & \\
p_3 & & p_4 & & q_2 & & q_3
\end{tikzcd}$$

Then $\mathcal{R} = \{(p_0, q_0), (p_1, q_1), (p_2, q_1), (p_3, q_2), (p_4, q_3)\}$ is a simulation. Thus, $p_0 \preceq q_0$. However, $q_0 \not\preceq p_0$, because assume that $\mathcal{R}$ is a simulation containing (q_0, p_0), then either (q_1, p_1) or (q_1, p_2) must be in $\mathcal{R}$. But in the former case $q_1 \xrightarrow{c}$ but $p_1 \not\xrightarrow{c}$ so if $\mathcal{R}$ is to be a simulation then $(p_1, p_1) \notin \mathcal{R}$. Similarly $(q_1, p_2) \notin \mathcal{R}$ can be argued. Therefore if $\mathcal{R}$ is a simulation $(q_0, p_0) \notin \mathcal{R}$.

$\square$

Now, two processes p and q could be considered equivalent if they simulate each other, i.e. $p \simeq q$ iff $p \preceq q$ and $q \preceq p$. However, this equivalence (often called *trace equivalence* does not preserve deadlock properties, as illustrated by the following example:

Example 2.2.7 *Let* **P** *be given by the diagram below:*

$$\begin{tikzcd}
& p_1 \arrow[dl, "a"'] \arrow[dr, "a"] & & & q_1 \arrow[d, "a"] \\
p_2 \arrow[d, "b"'] & & p_4 & & q_2 \arrow[d, "b"] \\
p_3 & & & & q_3
\end{tikzcd}$$

Then $\mathcal{R}_1 = \{(q_i, p_i) \mid i = 1, 2, 3\}$ and $\mathcal{R}_2 = \{(p_i, q_i) \mid i = 1, 2, 3\} \bigcup \{(p_4, q_2)\}$ are both simulations. Thus, $p_1 \preceq q_1$ and $q_1 \preceq p_1$. However, p_1 can perform and a-action and reach a state where a b-action is impossible, whereas q cannot. Thus, p and q have different deadlock properties. □

In order to obtain an equivalence that does preserve deadlock properties the notion of bisimulation is introduced. Uner this notion, two processes are considered equivalent if they have the same set of potential first actins and can remain having equal pontentiallity during the course of execution. More formally we have:

Definition 2.2.8 ***[Bisimulation]***
A binary releation $\mathcal{R}$ on Pr is a bisimulation *iff both $\mathcal{R}$ and*

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

are simulations. Two processes, p and q are said to be bisimulation equivalent *iff there exists a bisimulatin $\mathcal{R}$ with $p\mathcal{R}q$. In this case we write $p \sim q$.*
□

For $\mathcal{R} \subseteq Pr^2$ define $\overline{\mathcal{S}}(\mathcal{R})$, $\mathcal{B}(\mathcal{R}) \subseteq Pr^2$ as:

$$\overline{\mathcal{S}}(\mathcal{R}) = (\mathcal{S}(\mathcal{R}^T))^T \text{ and } \mathcal{B}(\mathcal{R}) = \mathcal{S}(\mathcal{R}) \cap \overline{\mathcal{S}}(\mathcal{R}). \tag{2.5}$$

Then we have the following properties:

Proposition 2.2.9 *$\mathcal{R} \subseteq Pr^2$ is a bisimulation iff $\mathcal{R} \subseteq \mathcal{B}(\mathcal{R})$.*

PROOF: *By proposition 2.2.2 and the definition of bisimulation.* □

For future development the following more general notion of *approximate bisimulations* is useful.

Definition 2.2.10 ***[Approximate bisimulation]***
Let $B \subseteq Pr^2$. A binary relation $\mathcal{R}$ on Pr is said to be an approximate bisimulation *with respect to B if*

$$\mathcal{R} \setminus B \subseteq \mathcal{B}(\mathcal{R}).$$

Whenever B is implicitly given we will say that $\mathcal{R}$ is an approximate bisimulation. □

Thus $\mathcal{R}$ will be a bisimulation with respect to B if only $B \subseteq \mathcal{B}(\mathcal{R})$ can be established.

From an implementation point of view the definition of bisimulation leaves much to be desired. However, the following proposition throws some light on these problems.

Proposition 2.2.11 ***[Strong bisimulation]***
$\mathcal{R} \subseteq Pr^2$ is a strong bisimulation iff whenever $p\mathcal{R}q$

1. *if $p \xrightarrow{\alpha} p'$ then $\exists q' : q \xrightarrow{\alpha} q' \;\wedge\; p'\mathcal{R}q'$*
2. *if $q \xrightarrow{\alpha} q'$ then $\exists p' : p \xrightarrow{\alpha} p' \;\wedge\; q'\mathcal{R}p'$*

□

As we can see from this proposition, it is possible to determine whether two processes are bisimulation equivalent or not just by (recursively) trying to match a derivation $p \xrightarrow{\alpha} p'$ of p by a derivation $q \xrightarrow{\alpha} q'$ of q so that the derivations of p' again can be matched by some derivation of q' and so on. q's derivations should be treated similarily.

We still have some important properties to state.

Proposition 2.2.12 *$\mathcal{B}$ is a monotonic endofunction on the complete lattice of binary relations over Pr.*

PROOF: *By proposition 2.2.3 and the fact that $\bigcap$ and $()^T$ (transposition) are monotonic functions.* □

Proposition 2.2.13 *$\mathcal{B}$ has a maximal fixed-point which equals $\sim$.* □

Proposition 2.2.14 *$\sim$ is a equivalence relation.*

PROOF: *$Id_{Pr} = \{(p,p) \mid p \in Pr\}$ is a bisimulation. Bisimulations are closed under composition and $()^T$.* □

'$\sim$' is a congruence relation, i.e. id $p \sim q$ then p and q can be interchanged in any context, as stated in the following theorem.

Theorem 2.2.15 *$\sim$ is a congruence relation. More preciesely $p_1 \sim p_2$ implies*

1. $p_1 + q \ \sim\ p_2 + q$, $q + p_1 \ \sim\ q + p_2$
2. $\alpha.p_1 \ \sim\ \alpha.p_2$, $\tau.p_1 \ \sim\ \tau.p_2$
3. $p_1 \mid q \ \sim\ p_2 \mid q$, $q \mid p_1 \ \sim\ q \mid p_2$
4. $p_1 \setminus S \ \sim\ p_2 \setminus S$, $p_1[N] \ \sim\ p_2[N]$

PROOF: *see [Milner].* □

As for simulation the definition of bisimulation equivalence provides an elegant proof technique due to proposition 2.2.13. To prove that $p \sim q$ it is sufficient and necessary to find a bisimulation containing (p,q).

Example 2.2.16 *Let P be given by the diagram below:*

$$\begin{tikzcd}
 & p_0 \arrow[d, "a"] & & & & & q_0 \arrow[dll, "a"'] \arrow[drr, "a"] & & & \\
 & p_1 \arrow[dl, "b"'] \arrow[dr, "c"] & & & q_1 \arrow[dl, "b"'] \arrow[dr, "c"] & & & & q_2 \arrow[dl, "c"'] \arrow[dr, "b"] & \\
p_2 & & p_3 & q_3 & & q_4 & & q_5 & & q_6
\end{tikzcd}$$

Then $\mathcal{R} = \{(p_0, q_0), (p_1, q_1), (p_1, q_2), (p_2, q_3), (p_2, q_6), (p_3, q_4), (p_3, q_5)\}$ *is a bisimulation with* $p_0 \mathcal{R} q_0$*. Thus* $p \sim q$*. In example 2.2.7,* $\mathcal{R}_1 \neq {\mathcal{R}_2}^T$ *so there is no reason to conclude* $p_1 \sim q_1$*. In fact it can be shown that the two processes* p_1 *and* q_1 *of example 2.2.7 are not bisimulation equivalent.* □

The above example gives some indication od the relationship between the simulation ordering $\preceq$ and the bisimulation equivalence $\sim$.

Proposition 2.2.17 *if* $p \sim q$ *then* $p \simeq q$*.*

PROOF: *Follows straight forward from the definition of* $\mathcal{B}$ *(equation 2.5).* □

Restricted process systems have an interesting property concerning bisimulation equivalence, which is stated in the following theorem.

Theorem 2.2.18 *Let* $\mathbf{P} = (pr, Act, \longrightarrow)$ *be a process system and* Q *be* $\longrightarrow$*-closed subset of* Pr*, then*

$$\forall p, q \in Q : p \sim q \text{ in } \mathbf{P} \;\Leftrightarrow\; p \sim q \text{ in } \mathbf{P} \downarrow Q$$

PROOF:
$\underline{\Rightarrow}$: $p \sim q$ in $\mathbf{P} \downarrow Q$ *follows directly as* $Q \subseteq Pr \;\wedge\; Q$ *is* $\longrightarrow$*-closed.*
$\underline{\Leftarrow}$: *Follows from the fact that* Q *is* $\longrightarrow$*-closed.* □

This theorem states that if two processes p and q are bisimulation equivalent in a restricted process system, $\mathbf{P}\!\downarrow Q$, then they will be bisimulation equivalent in $\mathbf{P}$ (notice that no restraints on $\mathbf{P}$ are given).

Though '$\sim$' has these properties we are still not quite satisfied, as in general we are not interested in *how* the process got to a given point (performed by a specific action) but rather *if* it gets there. Intuitively, we do not care about how many (if any) silent actions (τ) the process has to make in order to reach the goal i.e. we want unobservable actions (τ) to be absorbed by '$\longrightarrow$'. For this purpose a new notation is introduced.

Notation 2.2.19 *A general derivation of the form*

$$p \xrightarrow{\alpha_1} p_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} p_n$$

will often be written as

$$p \overset{\alpha_1\alpha_2\dots\alpha_n}{\longrightarrow} p_n.$$

We can also abbreviate

$$p \xrightarrow{\tau^n} p' \text{ by } p \overset{\epsilon}{\Longrightarrow} p',\ n \geq 0$$

and

$$p \overset{\tau^n\alpha\tau^m}{\longrightarrow} p' \text{ by } p \overset{\alpha}{\Longrightarrow} p',\ n, m \geq 0$$

and finally for each $s = \alpha_1\alpha_2\dots\alpha_n \in (Act \setminus \{\tau\})^*$

$$p \overset{\alpha_1\alpha_2\dots\alpha_n}{\Longrightarrow} p' \text{ equals } p \overset{s}{\Longrightarrow} p',\ n, m \geq 0$$

□

With this definition of '$\Longrightarrow$' the following result follows as an easy corollary of proposition 2.1.3.

Corollary 2.2.20 *For all CCS-expressions, p, the set*

$$\{(\alpha, p') \mid p \overset{\alpha}{\Longrightarrow} p'\}$$

is finite. □

In the following we will be considering a special bisimulation, namely *weak bisimulation*, defined as follows

Definition 2.2.21 ***[Weak bisimulation]***
$\mathcal{R} \subseteq Pr^2$ is a weak bisimulation *iff whenever $p\mathcal{R}q$ and $s \in (\Delta \cup \overline{\Delta})^* \cup \{\epsilon\}$*

1. *if $p \overset{s}{\Longrightarrow} p'$ then $\exists q' : q \overset{s}{\Longrightarrow} q' \ \wedge \ p'\mathcal{R}q'$*
2. *if $q \overset{s}{\Longrightarrow} q'$ then $\exists p' : p \overset{s}{\Longrightarrow} p' \ \wedge \ p'\mathcal{R}q'$*

If we can find a weak bisimulation $\mathcal{R}$ with $p\mathcal{R}q$, then we write $p \approx q$ and say that p and q are observational equivalent*.* □

(This definition has a special property; the two processes $infinite$:=: $\tau.infinite$ and NIL are in fact observational equivalent, $infinite \approx NIL$! Thus, whenever we have proved $p \approx q$—e.g. p may be a program and q its specification—we cannot deduce that p nas no infinite unseen actions, even if q has none).

Unlike the definition of (strong) bisimulation the definition of weak bisimulation is not directly suitable for implementation. However, by putting som restrictions on s we obtain the following useful proposition allowing us only to consider observational sequences of lenght at most 1.

Proposition 2.2.22 *$\mathcal{R} \subseteq Pr^2$ is a weak bisimulation if whenever $p\mathcal{R}q$, $s \in (\Delta \cup \overline{\Delta})^* \cup \{\epsilon\}$ and $\|s\| \leq 1$*

1. *if $p \overset{s}{\Longrightarrow} p'$ then $\exists q' : q \overset{s}{\Longrightarrow} q' \ \wedge \ p'\mathcal{R}q'$*
2. *if $q \overset{s}{\Longrightarrow} q'$ then $\exists p' : p \overset{s}{\Longrightarrow} p' \ \wedge \ p'\mathcal{R}q'$*

Proof: *Follows directly from the definition of weak bisimulation.* □

However, from an implementation point of view it would be better if we could get rid of the extra nondeterminism introduced by '$\Longrightarrow$'. therefore, yet another (the last) version of weak bisimulation is needed.

Proposition 2.2.23 *$\mathcal{R} \subseteq Pr^2$ is a weak bisimulation if whenever $p\mathcal{R}q$ and $\alpha \in Act$*

1. *if* $p \xrightarrow{\alpha} p'$ *then* $\exists q' : q \overset{\hat{\alpha}}{\Longrightarrow} q' \ \wedge \ p'\mathcal{R}q'$

2. *if* $q \xrightarrow{\alpha} q'$ *then* $\exists p' : p \overset{\hat{\alpha}}{\Longrightarrow} p' \ \wedge \ p'\mathcal{R}q'$

where $\hat{\alpha} = \alpha$ *when* $\alpha \neq \tau$ *and* $\hat{\tau} = \epsilon$*,*

Proof: *Follows immediately from proposition* 2.2.22 *and the fact that*

$$p \xrightarrow{s} p' \ \Rightarrow \ p \overset{\hat{s}}{\Longrightarrow} p', \ \|s\| = 1$$

□

This proposition is very suitable for implementation purposes as it is easy to determine a process' immediate derivatives.

Naturally, we cannot expect to make abstractions without paying something in return. We can get a hint of the price to pay by the following propositon.

Proposition 2.2.24 $p \approx \tau.p$

Proof: *see [Milner].* □

Using this result it can be argued that '$\approx$' cannot be a congruence relation. In particulr $p_1 \approx p_2$ does not imply $p_1 + p \ \approx \ p_2 + p$ (e.g. take $p_1 :=: NIL$, $p_2 :=: \tau.NIL$ and $p :=: \alpha.NIL$, then $p_1 \approx p_2$ according to proposition 2.2.24, but $p_1 + p \ \not\approx \ p_2 + p$. You see this by observing RHS $\overset{\epsilon}{\Longrightarrow} NIL$ but the only ϵ-experiment possible on LHS yields a result which is $\not\approx NIL$).

Theorem 2.2.25 $p \sim q$ *implies* $p \approx q$*.*

Proof: *Follows directly from the respective propositions, as*

$$p \xrightarrow{\alpha} p' \ \Rightarrow \ p \overset{\alpha}{\Longrightarrow} p', \alpha \in Act \setminus \{\tau\}$$

and

$$p \xrightarrow{\tau} p' \ \Rightarrow \ p \overset{\epsilon}{\Longrightarrow} p'$$

□

This theorem stats that all algebraic lwas that hold for '$\sim$' also hold for '$\approx$'.

Theorem 2.2.26 *Observational equivalence is a congruence for all operations except '+'. More precisely*

$$p \approx p' \implies \begin{cases} \alpha.p \approx \alpha.p',\ \tau.p \approx \tau.p' \\ p \mid q \approx p' \mid q \\ p \setminus S \approx p' \setminus S,\ p[N] \approx p'[N] \end{cases}$$

PROOF: *see [Milner].* □

Proposition 2.2.24 and theorem 2.2.25 and 2.2.26 together state that we can use all our laws and cancel τ's, in proving observational equivalence—as long as nothing is inferred about the result of substituting p for q under '+' when we only know $p \approx q$.

2.3 Modal characterizations

Matthew Hennessy and Robin Milner showed in [Hennessy,Milner] that both $\preceq$ and $\sim$ can alternatively be characterized by identifying a process with the properties it enjoys. That is, two processes are equivalent if and only if they enjoys exactly the same properties. The negative form of this statement brings further clarity to the usefulness of properties: two processes are inequivalent if one enjoys a property the other does not enjoy. It follows, as such, that we may in case of inequivalence use such a property (enjoyed by one and not by the other) as an *explanation* for the inequivalence. The remaining task then, is to find such a distinguishing property automatically.

For image-finite process systems the relevant properties are formulas from the following modal languages:

Let the language M (of formulas) be the least set such that:

1. $\mathtt{tt} \in M$
2. $F \wedge G \in M$ whenever $F, G \in M$

3. $\neg F \in M$ whenever $F \in M$

4. $\langle\alpha\rangle F \in M$ whenever $\alpha \in Act \ \wedge \ F \in M$

In [Hennessy,Milner] the authors define a satisfaction relation, $\models \subseteq Pr \times M$, as the least relation such that:

1. $p \models tt$ for $p \in Pr$

2. $p \models F \wedge G$ iff $p \models F \ \wedge \ p \models G$

3. $p \models \neg F$ iff $p \not\models F$

4. $p \models \langle\alpha\rangle F$ iff $\exists p' : p \xrightarrow{\alpha} p' \ \wedge \ p' \models F$

Notation 2.3.1 *In the later discussions we will adopt the following convenient notations:*

$$\begin{array}{rl} ff & \textit{stands for } \neg tt \\ A \vee B & \textit{stands for } \neg(\neg A \wedge \neg B) \\ [\alpha]F & \textit{stands for } \neg\langle\alpha\rangle\neg F \end{array}$$

□

We say p is α-deadlocked if there are no α-experiments on p. From the definition of the satisfaction relation, $\models$, we can now interpret many simple sentences as assertions about the possibility of deadlock as illustrated in the following example.

Example 2.3.2 *Some interpretations of simple modal properties:*

1. *$p \models \langle\alpha\rangle tt$; it is possible to carry out an α-experiment on p*

2. *$p \models [\alpha]ff$; p is α deadlocked*

3. *$p \models \langle\alpha\rangle([\beta]ff \vee [\gamma]ff)$; via an α-experiment it is possible to get into a state that is either β-deadlocked or γ-deadlocked*

4. *$p \models [\alpha](\langle\beta\rangle[\gamma]ff)$; at the end of any α-experiment a β-experiment that will leave the program in a state that is γ-deadlocked is possible*

□

Let L be the sublanguage of M consisting of the formulas not containing $\neg$. Now, define for $p \in Pr$ the following two sets:

$$M(p) = \{F \in M \mid p \models F\} \text{ and } L(p) = \{F \in L \mid p \models F\}$$

Then $\preceq$ and $\sim$ have the following characterizations:

Theorem 2.3.3 *If* $\mathbf{P}$ *is image-finite then:*

1. $p \sim q$ *iff* $M(p) = M(q)$
2. $p \preceq q$ *iff* $L(p) \subseteq L(p)$

PROOF: *see [Hennessy,Milner].* □

Using proposition 2.2.23 it is easy to obtain similar characterization of $\approx$ using the following manguage M_O. Let the language M_O of formulas be the least set such that:

1. $\mathtt{tt} \in M_O$
2. $F \wedge G \in M_O$ whenever $F, G \in M_O$
3. $\neg F \in M_O$ whenever $F \in M_O$
4. $\langle\langle \alpha \rangle\rangle F \in M_O$ whenever $\alpha \in \{Act \setminus \{\tau\}\} \ \wedge \ F \in M_O$
5. $\langle\langle \epsilon \rangle\rangle F \in M_O$ whenever $F \in M_O$

Furthermore, let $\models_O \subseteq Pr \times M_O$ be the least relation such that:

1. $p \models_O \mathtt{tt}$ for $p \in Pr$
2. $p \models_O F \wedge G$ iff $p \models_O F \ \wedge \ p \models_O G$
3. $p \models_O \neg F$ iff $p \not\models_O F$
4. $p \models_O \langle\langle \alpha \rangle\rangle F$ iff $\exists p' : p \overset{\alpha}{\Longrightarrow} p' \ \wedge \ p' \models_O F$

5. $p \models_O \langle\langle\epsilon\rangle\rangle F$ iff $\exists p' : p \overset{\epsilon}{\Longrightarrow} p' \ \wedge \ p' \models_O F$

Notation 2.3.4 *We will adopt the following convenient notations:*

$$\begin{array}{cll} \mathit{ff} & \textit{stands for} & \neg \mathit{tt} \\ A \vee B & \textit{stands for} & \neg(\neg A \wedge \neg B) \\ [\![\alpha]\!]F & \textit{stands for} & \neg\langle\langle\alpha\rangle\rangle\neg F \\ [\![\epsilon]\!]F & \textit{stands for} & \neg\langle\langle\epsilon\rangle\rangle\neg F \end{array}$$

We will use $\models$ instead of $\models_O$ whenever there is no possibility for confusion.
□

Notice, that M_O gives us the possibility to abstract from τ's in modal formulas as opposed to M. In anology to M, M_O has the following (important) property

Theorem 2.3.5 *If* $\mathbf{P}$ *is image-finite under* $\Longrightarrow$ *then:*

$$p \approx q \Leftrightarrow M_O(p) = M_O(q)$$

PROOF: *see [Hennessy,Milner].* □

Chapter 3

Existing Systems

—arguing for bisimulation equivalence and inequivalence.

In this section two existing systems will be presented—each taking a different approach towards proving (or disproving) bisimulation equivalence between two CCS-processes—emphasizing the principles of operation. The first system tries to find a 'maximal' bisimulation whereas the second system tries to find a 'minimal' bisimulation including a given pair. In contrast to the 'minimal' system, the 'maximal' system also gives information in the case of inequivalence. Following the modal characterization result of theorem 2.3.3 the information given is a modal property enjoyed by one of the processes.

Finally, a way of extending the 'minimal' system in order to enable it to give modal property as an argument of inequaility is presented.

3.1 The 'maximal' system

The first system to be presented is the system trying to find a 'maximal' bisimulation of two processes. This system is a result of a master thesis work done by Karsten Vestmar and Jørgen Olesen [Vestmar,Olesen], inspired by [Larsen]. Actually, 'maximal' is a bit misleading, as this system constructs '$\sim$' (the strong equivalence relation) restricted to the two processes under consideration and their derivatives.

The authors approach is an application of the *generalized partitioning problem* [Larsen]. A *partitioning* of a set S consists of disjoint, nonempty subsets of S called *blocks*, whose union is S.

Definition 3.1.1 ***[Generalized partitioning problem]***
As input is given a finite set S, an initial partitioning of S, $\Gamma_0^{\mathbf{P}} = \{B_1, \ldots, B_p\}$ and k functions with $f_l : S \mapsto 2^S, 1 \leq l \leq k$. The problem is to find the coarsest partitioning $\Gamma_f^{\mathbf{P}} = \{E_1, \ldots, E_q\}$ of S such that

1. *$\Gamma_f^{\mathbf{P}}$ is a refinement of $\Gamma_0^{\mathbf{P}}$*
   (i.e. each block E_i is a subset of some B_j)

2. *For all blocks E_i, all $a, b \in E_i$, any function f_l and any block E_j*

$$f_l(a) \cap E_j \neq \emptyset \;\Leftrightarrow\; f_l(b) \cap E_j \neq \emptyset$$

$\square$

Uniqueness and existence of $\Gamma_f^{\mathbf{P}}$ is covered by [Larsen]. Now, the general partitioning problem is applied in order to solve the (strong) bisimulation equivalence problem as follows: first, for any finite process system $\mathbf{P} = (Pr, Act, \longrightarrow)$, let $\Lambda_{\mathbf{P}}$ be the generalized partitioning system consisting of the set Pr, the initial partitioning $\Gamma_0^{\mathbf{P}} = \{Pr\}$ and $\|Act\|$ functions $f_a : Pr \mapsto 2^{Pr}$ for $a \in Act$, with $f_a(p) = \{p' \mid p \xrightarrow{a} p'\}$. Let $\Gamma_f^{\mathbf{P}}$ be the solution to $\Lambda_{\mathbf{P}}$. Then the following holds:

Theorem 3.1.2 *For all processes p and q of $\mathbf{P}$, $p \sim q$ if and only if p and q belong to the same block of $\Gamma_f^{\mathbf{P}}$.*

Proof:
$\underline{\Leftarrow:}$ *Show that the relation $\mathcal{R} \subseteq Pr \times Pr$ defined by:*

$$p\mathcal{R}q \;\Leftrightarrow\; \exists i : p, q \in E_i, \quad E_i \text{ is a block in } \Gamma_f^{\mathbf{P}}$$

is a strong bisimulation and thus $\mathcal{R} \subseteq \sim$. Let $p\mathcal{R}q$ and $p \xrightarrow{a} p'$. Now, assume $p' \in E_j$, where E_j is some block of $\Gamma_f^{\mathbf{P}}$. $\{p'\} \subseteq f_a(p) \cap E_j \neq \emptyset$ by definition 3.1.1 and thus by the same definition it follows that $f_a(q) \cap E_j \neq \emptyset$, and hence $\exists q' \in E_j : q \xrightarrow{a} q'$.

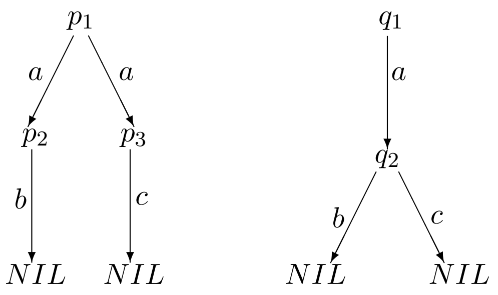
$\underline{\Rightarrow:}$ Let $Pr_\sim$ be the set of equivalence classes of Pr under $\sim$. $Pr_\sim$ is by definition a partitioning of Pr and it is obvious that $Pr_\sim$ satisfies (1) and (2) of definition 3.1.1. Thus, by definition, $\Gamma^{\mathbf{P}}_f$ is coarser than $Pr_\sim$, i.e. "$\Rightarrow$" follows immediately. $\square$

Intuitively, the algorithm applied by [Vestmar,Olesen] works as follows. Let $\Gamma^{\mathbf{P}}_0 = \{Pr\}$. Our goal is to find the coarsest partitioning $\Gamma^{\mathbf{P}}_f = \{E_1, \ldots, E_n\}$ (the solution) of $\Lambda_{\mathbf{P}}$ such that whenever $p, q \in E_i$ then $p \sim q$, Let $\Gamma^{\mathbf{P}}_n = \{B1, \ldots, B_k\}$ be the result of n refinements (partitions of $\Gamma^{\mathbf{P}}_0$) then $\Gamma^{\mathbf{P}}_{n+1}$ is constructed as follows: For each block $B_i \in \Gamma^{\mathbf{P}}_n$ choose any $p \in B_i$. We will now use p and its immediate derivatives (determined by $\longrightarrow$) as the criterion for whether or not to split B_i into two, $B_i = B'_i \cup B''_i$. B'_i is defined to contain all processes 'equivalent' to p (relative to $\Gamma^{\mathbf{P}}_n$), and B''_i contains the rest, i.e.

$$\begin{aligned} &B'_i = \{q \in B_i \mid \forall a \forall j \exists p' \in B_j : p \xrightarrow{a} p' \;\Leftrightarrow\; \exists q' \in B_j : q \xrightarrow{a} q'\} \\ &B''_i = B_i \setminus B'_i \end{aligned}$$

The split is only performed if both B'_i and B''_i are non-empty. Thus, at each step any block is at most split into two.

Example 3.1.3 *Let* $\mathbf{P}$ *be given by the diagram below:*

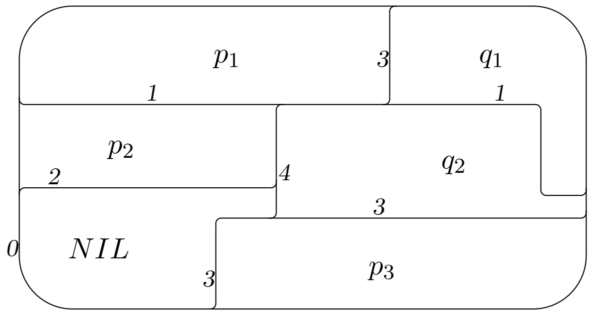


then the following sequence of partitionings could be the result of using the *above mentioned algorithm:*

$$\begin{array}{lcl}\Gamma_0^{\mathbf{P}} & = & \{\{p_1, p_2, p_3, q_1, q_2, NIL\}\},\\ \Gamma_1^{\mathbf{P}} & = & \{\{p_1, q_1\}, \{p_2, p_3, q_2, NIL\}\},\\ \Gamma_2^{\mathbf{P}} & = & \{\{p_1, q_1\}, \{p_2\}, \{p_3, q_2, NIL\}\},\\ \Gamma_3^{\mathbf{P}} & = & \{\{p_1\}, \{q_1\}, \{p_2\}, \{p_3\}, \{q_2, NIL\}\},\\ \Gamma_4^{\mathbf{P}} & = & \{\{p_1\}, \{q_1\}, \{p_2\}, \{p_3\}, \{q_2\}, \{NIL\}\},\\ \Gamma_f^{\mathbf{P}} & = & \{\{p_1\}, \{q_1\}, \{p_2\}, \{p_3\}, \{q_2\}, \{NIL\}\}.\end{array}$$

Notice that p_1 and q_1 are in separate blocks in $\Gamma_f^{\mathbf{P}}$ indicating $p \not\sim q$. Also, the partitioning could have been ended already in $\Gamma_3^{\mathbf{P}}$ if we had detected that p_1 and q_1 were separated at that point. However, then we would not have established $\sim$, which $\Gamma_f^{\mathbf{P}}$ gives us.

This partitioning can alternatively be represented as in the following drawing where the numbers corresponds to the partitionings.



□

Apparently, what we are doing is to split Pr into equivalence classes. When the solution $\Gamma_f^{\mathbf{P}}$ is reached we simply have to look through the equivalence classes of $\Gamma_f^{\mathbf{P}}$ to determine whether or not p and q belong to the same block E_i which will verify $p \sim q$. (We know when $\Gamma_f^{\mathbf{P}}$ is reached, as $\Gamma_f^{\mathbf{P}}$ is the fixed-point of the finite sequence $\Gamma_i^{\mathbf{P}}$ i.e. when for a partitioning $\Gamma_n^{\mathbf{P}}$ none of the blocks $B_i \subseteq \Gamma_n^{\mathbf{P}}$ are split, then $\Gamma_f^{\mathbf{P}} = \Gamma_n^{\mathbf{P}}$).

From the modal characterization of strong equivalence (theorem 2.3.3) we know that in case $p \not\sim q$ then there exists a model expression F such that $p \models F$ and $q \not\models F$. F can be regarded as *the reason why* p and q

are inequivalent. This fact is used by this system so that it returns a modal expression F with $p \models F$ and $q \not\models F$ when $p \not\sim q$. This is implemented by assigning properties to each block B_i in the following manner; each block B_i of the current partitioning is associated with a modal expression F_i such that

$$\forall p \in B_i : p \models F_i \;\wedge\; \forall q \in B_j, i \neq j : q \not\models F_i \tag{3.1}$$

Thus, the property F_i is characteristic for the block B_i. The following strategy ensures the invariance of the above property: The single block of the initial partitioning is associated with the modal expression $\mathtt{tt}$; this will clearly satisfy the property in equation 3.1. Let $B_i'' = \{q_1, \ldots, q_n\}$ and consider any q_j. Then either

$$\exists k \exists p' \in B_k : p \xrightarrow{a} p' \;\wedge\; \forall q' : q_j \xrightarrow{a} q' \Rightarrow q' \notin B_k$$

or

$$\exists k \exists q' \in B_k : q \xrightarrow{a} q' \;\wedge\; \forall p' : p \xrightarrow{a} p' \Rightarrow p' \notin B_k$$

will hold. The property F_{ij} $(p \models F_{ij} \;\wedge\; q_j \not\models F_{ij})$ can then be defined as

$$F_{ij} = \langle a \rangle F_k$$

and

$$F_{ij} = \neg \langle a \rangle F_k$$

respectively. Thus we can define F_i' and F_i'' in terms of the F_{ij}'s as

$$\begin{aligned} F_i' &= F_i \wedge (F_{i1} \wedge \ldots \wedge F_{in}) \\ F_i'' &= F_i \wedge (\neg F_{i1} \vee \ldots \vee \neg F_{in}) \end{aligned}$$

Then clearly $p \models F_i'$ and $q_j \not\models F_i'$ as $q_j \not\models F_{ij}$. Similarly, $p \not\models F_i''$ as $p \not\models \neg F_{ij}$ and $q_j \models F_i''$ and $q_j \not\models F_{ij}$. Thus, this principle secures the desired property of the modal expression assigned to each block B_i. Example 3.1.4 shows a possible result of applying this algorithm to example 3.1.3.

Example 3.1.4 *Let each block B_j in a partitioning $\Gamma_i^{\mathbf{P}}$ be denoted by B_{ij} and let $\models \subseteq 2^{Pr} \times M$ so that $B_{ij} \models F \;\Leftrightarrow\; \forall p \in B_{ij} : p \models F, F \in M$. Then the following modal expressions can be constructed using the above mentioned principle on example 3.1.3 (deleting double negations when* *necessary).*

$$
\begin{array}{lcl}
B_{01} & \models & \mathit{tt} \\
 & & \\
B_{11} & \models & \mathit{tt} \wedge \langle a\rangle \mathit{tt} \\
B_{12} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \\
 & & \\
B_{21} & \models & \mathit{tt} \wedge \langle a\rangle \mathit{tt} \\
B_{22} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\wedge \\
 & & \neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})) \\
B_{23} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})) \\
 & & \\
B_{31} & \models & \mathit{tt} \wedge \langle a\rangle \mathit{tt}\wedge \\
 & & \langle a\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\wedge \\
 & & \neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))) \\
 & & \\
B_{32} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}\wedge \\
 & & \langle a\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\wedge \\
 & & \neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))) \\
 & & \\
B_{33} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}\wedge \\
 & & \neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})) \\
B_{34} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\wedge \\
 & & (\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))\wedge \\
 & & \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))) \\
B_{35} & \models & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\wedge \\
 & & (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\vee \\
 & & \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
 & & \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))) \\
 & & \\
B_{41} & \models & \mathit{tt} \wedge \langle a\rangle \mathit{tt}\wedge \\
 & & \langle a\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\wedge \\
 & & (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))
\end{array}
$$

$$\begin{aligned}
B_{42} \models\ & \mathit{tt} \wedge \langle a\rangle \mathit{tt} \wedge \\
& \neg\langle a\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\wedge \\
& (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))) \\
B_{43} \models\ & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\wedge \\
& (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \wedge (\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})) \\
B_{44} \models\ & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\wedge \\
& ((\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\wedge \\
& \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))) \\
B_{45} \models\ & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\wedge \\
& (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))\vee \\
& \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))))\wedge \\
& (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))\vee \\
& \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))))) \\
B_{46} \models\ & \mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))\wedge \\
& (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))\vee \\
& \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}))))\wedge \\
& (\neg\langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))\vee \\
& \langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt} \wedge (\neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt}) \vee \neg\langle b\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})))))
\end{aligned}$$

As $p_1 \in B_{41}$ and $q_1 \in B_{42}$ we have $p_1 \not\sim q_1$. By using the adopted notational conventions (see notation 2.3.4, p_1's and q_1's properties can be rewritten as:

$$\begin{aligned}
p_1 \models\ & \mathit{tt} \wedge \langle a\rangle \mathit{tt} \wedge \langle a\rangle(\mathit{tt} \wedge [a]\mathit{ff} \wedge \langle b\rangle(\mathit{tt} \wedge [a]\mathit{ff})\wedge \\
& [c](\mathit{ff} \vee \langle a\rangle \mathit{tt}) \wedge \langle b\rangle(\mathit{tt} \wedge [a]\mathit{ff})) \\
q_1 \models\ & \mathit{tt} \wedge \langle a\rangle \mathit{tt} \wedge [a](\mathit{ff} \vee \langle a\rangle \mathit{tt} \vee [b](\mathit{ff} \vee \langle a\rangle \mathit{tt}) \vee \langle c\rangle(\mathit{tt} \wedge \neg\langle a\rangle \mathit{tt})\vee \\
& [b](\mathit{ff} \vee [a]\mathit{ff}))
\end{aligned}$$

by removing the negations. It is easy to see that two alternative and simpler—though not equivalent—properties distinguishing p_1 and q_1 are:

$$\begin{array}{rcl} p_1 & \models & \langle a\rangle[c]\mathit{ff} \\ q_1 & \models & [a]\langle c\rangle\mathit{tt} \end{array}$$

□

As can be seen from this example the above method will for even simple processes produce unnecessarily complicated modal expressions. This (major) deficiency, however, can be remedied by applying a formula reduction algorithm [Vestmar,Olesen]. Unfortunately, the implementation of [Vestmar,Olesen] still remains to be extended in order to cater for recursively defined processes. However, as can be seen from above, there is no theoretical reason why this extension should not be possible.

3.2 The 'minimal' system

The second system to be presented is the system trying to find a 'minimal' bisimulation containing two given processes. This system is the one presented in Kim G. Larsen's Ph.D. Thesis [Larsen]. The system is implemented in PROLOG [Clocksin,Mellish] and the algorithm is proved sound and (restricted) complete. It can be considered to be a *theorem prover* in the following sense; given two processes p and q it will try to construct a bisimulation containing the pair (p, q) (which equals a proof) if and only if p is equivalent to q. If p and q are inequivalent this system will terminate with failure. Moreover, the program will only terminate if the processes p and q both have finite state-transition diagrams—this is the sense in which the algorithm is only restricted complete. One of the drawbacks of this system is thus the lack of information of *why* processes are inequivalent. Later we will show how to remedy this inconvenience.

The approach taken is quite straight forward and follows closely the recursive definition of bisimulation. Let

$$\begin{aligned}
&\mathit{bisim} && \subseteq \; Pr \times Pr \times 2^{Pr\times Pr} \\
&\left.\begin{array}{l}\mathit{closure}\\ \mathit{matchl}\\ \mathit{matchr}\end{array}\right\} && \subseteq \; Pr \times Pr \times 2^{Pr\times Pr} \times 2^{Pr\times Pr} \\
&\left.\begin{array}{l}\mathit{matchl}^{+}\\ \mathit{matchr}^{+}\end{array}\right\} && \subseteq \; Pr \times Pr \times 2^{Act\times Pr} \times 2^{Pr\times Pr} \times 2^{Pr\times Pr}
\end{aligned}$$

be the smallest relations closed under the rules given in figure 3.1. The intended meaning of each of the six relations is informally:

bisim: Given two processes p and q it will try to build a bisimulation containing (p, q).

closure: Given two processes p and q and an approximate bisimulation (see definition 2.2.10) B containing (p, q), *closure* will try to extend B to a genuine bisimulation C.

matchl: Given two processes p and q and an approximate bisimulation B, *matchl* will try to extend B to an approximate bisimulation C closed under $\mathcal{L}(= \mathcal{S})$.

matchr: Given two processes p and q and an approximate bisimulation B, *matchl* will try to extend B to an approximate bisimulation C closed under $\mathcal{R}(= \overline{\mathcal{S}})$.

matchl^{+}: Given two processes p and q, an approximate bisimulation B containing (p, q), and a subset M of p's derivations such that q only remains to match those of p's derivations which still are in M. Then *matchl*$^{+}$ will try to extend B to an approximate bisimulation D closed under $\mathcal{L}$.

matchr^{+}: Given two processes p and q, an approximate bisimulation B containing (p, q), and a subset N of q's derivations such that p only remains to match those of q's derivations which still are in N. Then *matchr*$^{+}$ will try to extend B to an approximate bisimulation D closed under $\mathcal{R}$.

From the definition of $\mathcal{L}$ it follows that the approximate bisimulation D constructed by $matchl^+$ must be such that for each derivation $p \xrightarrow{a} p'$ of p, q has a match within D, i.e. $q \xrightarrow{a} q'$ for some q' and $(p', q') \in D$ (see figure 3.2 where $A = \{p'\} \times \{q' \mid q \xrightarrow{a} q'\}$, $A_1 = A \bigcap B$ and $A_2 = A \setminus A_1$). If p' can be matched by q' such that $(p', q') \in B$ (i.e. $A_1 \neq \emptyset$ in figure 3.3) then all is fine and we do not need to search for another matching q derivation. If q cannot match the derivation (a, p') in B (i.e. $A_1 = \emptyset$ in figure 3.3), B will be extended with a pair (p', q') where $q \xrightarrow{a} q'$ (see figure 3.4 where the hatched area symbolizes the extension to B). Backtracking may be necessary at this point in order to replace the chosen q' with another a-derivation of q, should it later be discovered that the chosen derivation is not a match for p. Obviously, q will then have a match for (a, p') in this extended set. Before dealing with the remaining derivations in M, however, the extended set$B \bigcup \{(p', q')\}$ has to be closed, i.e. (p', q') should be treated similarly etc.

Similarly, all derivations $q \xrightarrow{a} q'$ of q must have a matching p', such that $p \xrightarrow{a} p'$ and $(p', q') \in D$ for the approximate bisimulation D constructed by $matchr^+$ (see figures 3.2, 3.3, and 3.4).

The system is sound in the sense that C is a bisimulation containing the pair (p, q) whenever $bisim(p, q, C)$ holds. It is useful to introduce some verification conditions, Bis, Cl, Ml, Mr, Ml^+, and Mr^+ (see figure 3.5 and 3.6). The induction principle associated with the inference system, then implies we must show that these verification conditions are closed under the rules of the system (see [Aczel]). Then the following inclusions will hold:

$$\begin{array}{llclcl} bisim & Bis & \subseteq & matchr & \subseteq & Mr \\ closure & Cl & \subseteq & matchl^+ & \subseteq & Ml^+ \\ matchl & Ml & \subseteq & matchr^+ & \subseteq & Mr^+ \end{array}$$

due to the leastness of $bisim$, ... The soundness the follows directly (from Bis).

The system is complete in the sense that whenever p and q are equivalent then $bisim(p, q, C)$ holds for some C. (Actually, the system is only restricted complete as there is a restriction on p and q in that they should have finite state-transition diagrams). The completeness is obtained as an easy corollary of the theorem displayed in figure 3.7 proved by induction on n. (Notice that 'ANT()' denotes the antecedent of the corresponding verification predicate and $\sharp_F$ is a size-function measuring the size of the input

$$B \ : \quad \frac{closure(p, q, \{(p, q)\}, C)}{bisim(p, q, C)}$$

$$C \ : \quad \frac{matchl(p, q, B, C) \quad matchr(p, q, C, D)}{closure(p, q, B, D)}$$

$$ML \ : \quad \frac{matchl^{+}(p, q, M, B, C)}{matchl(p, q, B, C)} \quad ; \ M = \{(a, p') \mid p \xrightarrow{a} p'\}$$

$$MR \ : \quad \frac{matchr^{+}(p, q, N, B, C)}{matchr(p, q, B, C)} \quad ; \ N = \{(a, q') \mid q \xrightarrow{a} q'\}$$

$$ML^{+} \ : \quad matchl^{+}(p, q, \emptyset, B, B)$$

$$\frac{matchl^{+}(p, q, M, B, D)}{matchl^{+}(p, q, \{(a, p')\} \cup M, B, D)} \quad ; \ q \xrightarrow{a} q' \wedge (p', q') \in B$$

$$\frac{closure(p', q', \{(p', q')\} \cup B, C) \quad matchl^{+}(p, q, M, C, D)}{matchl^{+}(p, q, \{(a, p')\} \cup M, B, D)} \quad ; \ q \xrightarrow{a} q'$$

$$MR^{+} \ : \quad matchr^{+}(p, q, \emptyset, B, B)$$

$$\frac{matchr^{+}(p, q, N, B, D)}{matchl^{+}(p, q, \{(a, p')\} \cup N, B, D)} \quad ; \ p \xrightarrow{a} p' \wedge (p', q') \in B$$

$$\frac{closure(p', q', \{(p', q')\} \cup B, C) \quad matchr^{+}(p, q, N, C, D)}{matchr^{+}(p, q, \{(a, q')\} \cup N, B, D)} \quad ; \ p \xrightarrow{a} p'$$

Figure 3.1: Inference system for construction of bisimulations.

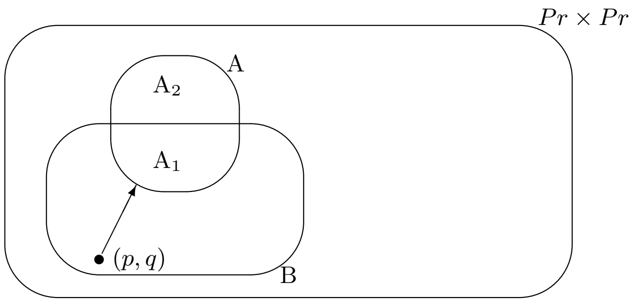


Figure 3.2: Sketch of $matchl^+$ situation.

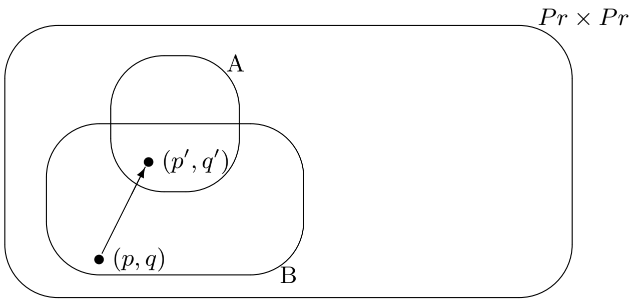


Figure 3.3: $matchl^+$ situation when a match can be found in B.

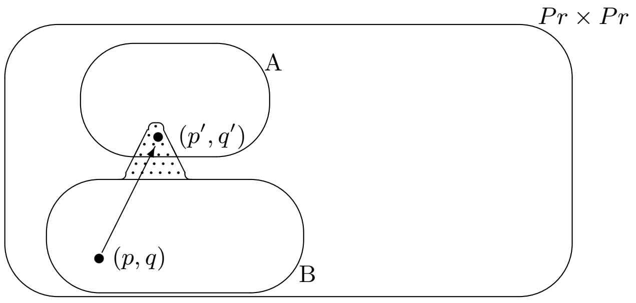


Figure 3.4: $matchl^+$ situation when no match can be found in B.

$$
\begin{aligned}
&Bis(p,q,C) \Leftrightarrow^{\Delta} \\
&\quad \{\} \Rightarrow \\
&\quad \left[\begin{array}{l} (p,q) \in C \;\; \wedge \\ C \subseteq \mathcal{B}(C) \end{array} \right] \\
&Cl(p,q,B,D) \Leftrightarrow^{\Delta} \\
&\quad \{(p,q) \in B\} \Rightarrow \\
&\quad \left[\begin{array}{l} B \subseteq D \;\; \wedge \\ D \setminus (B \setminus \{(p,q)\}) \subseteq \mathcal{B}(D) \end{array} \right] \\
&Ml(p,q,B,C) \Leftrightarrow^{\Delta} \\
&\quad \{(p,q) \in B\} \Rightarrow \\
&\quad \left[\begin{array}{l} B \subseteq C \;\; \wedge \\ C \setminus B \subseteq \mathcal{B}(C) \;\; \wedge \\ (p,q) \in \mathcal{L}(C) \end{array} \right] \\
&Mr(p,q,B,C) \Leftrightarrow^{\Delta} \\
&\quad \{(p,q) \in B\} \Rightarrow \\
&\quad \left[\begin{array}{l} B \subseteq C \;\; \wedge \\ C \setminus B \subseteq \mathcal{B}(C) \;\; \wedge \\ (p,q) \in \mathcal{R}(C) \end{array} \right]
\end{aligned}
$$

Figure 3.5: Verification conditions for minimal system.

$$Ml^+(p,q,M,B,D) \Leftrightarrow^{\Delta}$$
$$\left\{\begin{array}{l} (p,q) \in B \ \wedge \\ M \subseteq \{(a,p') \mid p \xrightarrow{a} p'\} \ \wedge \\ (p_M,q) \in \mathcal{L}(B) \end{array}\right\} \Rightarrow$$
$$\left[\begin{array}{l} B \subseteq D \ \wedge \\ D \setminus B \subseteq \mathcal{B}(D) \ \wedge \\ (p,q) \in \mathcal{L}(D) \end{array}\right]$$

$$Mr^+(p,q,N,B,D) \Leftrightarrow^{\Delta}$$
$$\left\{\begin{array}{l} (p,q) \in B \ \wedge \\ N \subseteq \{(a,q') \mid q \xrightarrow{a} q'\} \ \wedge \\ (p,q_N) \in \mathcal{R}(B) \end{array}\right\} \Rightarrow$$
$$\left[\begin{array}{l} B \subseteq D \ \wedge \\ D \setminus B \subseteq \mathcal{B}(D) \ \wedge \\ (p,q) \in \mathcal{R}(D) \end{array}\right]$$

where $p_M = \sum\{a.p' \mid p \xrightarrow{a} p' \ \wedge \ (a,p') \notin M\}$ (q_N similar)

Figure 3.6: Verification conditions for minimal system (cont.).

$$\begin{bmatrix} Bis(p,q,C) \wedge \\ \sharp_{Bis}(p,q,C) \leq n \end{bmatrix} \Rightarrow \quad \exists C' \subseteq C : bisim(p,q,C')$$

$$\begin{bmatrix} ANT(Cl(p,q,B,C)) \wedge \\ Cl(p,q,B,C) \wedge \\ \sharp_{Cl}(p,q,B,C) \leq n \end{bmatrix} \Rightarrow \quad \exists C' \subseteq C : closure(p,q,B,C')$$

$$\begin{bmatrix} ANT(Ml(p,q,B,C)) \wedge \\ Ml(p,q,B,C) \wedge \\ \sharp_{Ml}(p,q,B,C) \leq n \end{bmatrix} \Rightarrow \quad \exists C' \subseteq C : matchl(p,q,B,C')$$

$$\begin{bmatrix} ANT(Mr(p,q,B,C)) \wedge \\ Mr(p,q,B,C) \wedge \\ \sharp_{Mr}(p,q,B,C) \leq n \end{bmatrix} \Rightarrow \quad \exists C' \subseteq C : matchr(p,q,B,C')$$

$$\begin{bmatrix} ANT(Ml^+(p,q,M,B,C)) \wedge \\ Ml^+(p,q,M,B,C) \wedge \\ \sharp_{Ml^+}(p,q,M,B,C) \leq n \end{bmatrix} \Rightarrow \quad \exists C' \subseteq C : matchl^+(p,q,M,B,C')$$

$$\begin{bmatrix} ANT(Mr^+(p,q,N,B,C)) \wedge \\ Mr^+(p,q,N,B,C) \wedge \\ \sharp_{Mr^+}(p,q,N,B,C) \leq n \end{bmatrix} \Rightarrow \quad \exists C' \subseteq C : matchr^+(p,q,N,B,C')$$

Figure 3.7: Completeness conditions for minimal system.

arguments—in these cases all arguments except for C).

The fact that the system is developed and verified only for finding strong bisimulations is by no means a restriction as the only difference between strong bisimulations and other types of bisimulations (e.g. weak bisimulation) is the derivation relations used. Thus, the system is easily modified in order to cope with other notions of bisimulations; the modification needed is merely a matter of redefining the derivation relation.

Another nice thing about this system is the (almost) one-to-one correspondence between the inference rules and the PROLOG implementation (see appendix A).

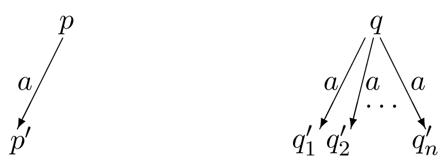


Figure 3.8: Sketch of *matchl*$^+$-situation.

The lack of use of modal expressions for reasoning about inequality of processes will be dealt with in the following section.

3.3 Initial proposal for enhancing the 'minimal' system

We will now sketch a possible way of including modal expressions for use as an argument of inequality in the minimal system.

Intuitively, we will essentially introduce two inference rules in addition to the ones given in figure 3.1. These new rules will act as 'catch all' rules i.e. whenever *matchl*$^+$ or *matchr*$^+$ otherwise would fail the new rules will take over and make a note—construct a modal expression—describing *what* is separating the two proesses under consideration before failing.

An expression arguing for the inequality of p and q is constructed as follows (for *matchl*$^+$); let $p \xrightarrow{a} p'$ and $\{q'_1, \dots, q'_n\} = \{q' : q \xrightarrow{a} q'\}$ (see figure 3.3). *matchl*$^+$ must now try to find a match q'_i for p' of q. If this is possible all is well and *matchl*$^+$ has succeeded. However, if no matching q'_i can be found (i.e. $p \not\sim q$) we know that $\forall i \leq n : p' \not\sim q'_i$ and according to theorem 2.3.3 we have $M(p') \neq M(q'_i)$ i.e. we can find $\forall i \leq n : p' \models F_i \wedge q'_i \not\models F_i$. Now, construct an expression F' from all F_i's as:

$$F' = \bigwedge \{F_1, \dots, F_n\}$$

then $p' \models F'$ as $\forall i \leq n : p' \models F_i$ and $\forall i \leq n : q'_i \not\models F'$ because $\forall i \leq n : q'_i \not\models F_i$. The modal expression, F, explaining the inequality of p and q can then be defined as

$$F = \langle a \rangle F'$$

i.e. $p \models F$ but $q \not\models F$. Similarly, an expression, F, can be constructed in the *matchr*$^+$-case, so that whenever $p \not\sim q$ we have $p \models F$ and $q \not\models F$ where

$$F = [a] \bigvee \{F_1, \ldots, F_n\}$$

In case one of the processes p and q has no a-derivation, then $n = 0$ and we therefore define F_0 to be $\mathtt{tt}$ in the *matchl*$^+$-case and $\mathtt{ff}$ in the *matchr*$^+$-case which clearly will satisfy the above requirement.

In the PROLOG-implementation we can at run-time make notes by extending the database with (Horn-)caluses of the form 'noteq(p, p, F)' saying that p and q are inequivalent because p enjoys the property F which q does not enjoy when *matchl*$^+$ of *matchr*$^+$ fails. However, we have to convince outselves that the principle is sound, i.e. is it correct to make these 'noteq' assumptions? An informal plausability argument for this may be given by looking at the verification conditions used in [Larsen] to prove the soundness and restricted completeness of the minimal system (see figures 3.5, 3.6, and 3.7).

Assume that *matchl* fails. From figure 3.7 it follows that no matter which C we choose the verification predicate Ml must be false as the antecedent of Ml is guaranteed to hold is the 'calling-sequence' implied by the inference rule set is not violated, and $\sharp_{Ml}$ does not depend on C. Then Ml will be false if the conjunction

$$\begin{bmatrix} B \subseteq C \;\wedge \\ C \setminus B \subseteq \mathcal{B}(C) \;\wedge \\ (p, q) \in \mathcal{L}(C) \end{bmatrix}$$

is false for any C (refer to figure 3.5. This can alternatively be expressed as

$$\forall C : B \subseteq C \;\wedge\; C \setminus B \subseteq \mathcal{B}(C) \;\Rightarrow\; (p, q) \notin \mathcal{L}(C) \tag{3.2}$$

Now, for the soundness of this method, we want to argue that whenever equation 3.2 holds, then $p \not\sim q$. Suppose $p \sim q$. We will try to establish a contradiction. Notice that $\sim \subseteq \mathcal{L}(\sim)$ and $(p, q) \in \mathcal{L}(\sim)$. Let $C = \sim \cup B$. Then

$$B \subseteq C$$

and

$$C \setminus B = \sim \setminus B \subseteq \mathcal{B}(\sim) \subseteq \mathcal{B}(\sim \cup B) = \mathcal{B}(C)$$

due to the monotony of $\mathcal{B}$. However,

$$(p,q) \in \mathcal{B}(\sim) \subseteq \mathcal{L}(\sim) \subseteq \mathcal{L}(\sim \cup B) = \mathcal{L}(C)$$

due to the monotony of $\mathcal{L}$. But this is in conflict with equation 3.2 which is supposed to hold for any C. Thus we can conclude that whenever *matchl* fails it is safe and correct to conclude $p \not\sim q$. Similarly, it is safe and correct to conclude that $p \not\sim q$ whenever *matchr* fails.

Thus, we now know that the inclusion of the two catch-all rules in the inference rule set, making 'noteq' assumptions, yields safe and correct assumptions concerning inequivalence of processes. However, this method makes (heavily) use of side-effects (the 'noteq' clauses) which makes it very hard to prove rigorously the soundness and completeness of this extended system (we simply does not know of any suitable proof technique for such selfmodifying PROLOG programs). In the next chapter we will therefore, prove properly and rigogously the soundness and completeness of a similar system which avoids the side-effect pitfall.

Chapter 4

A new system

—arguing for bisimulation equivalence and inequivalence.

In this section we will construct and verify rigorously a pure functional extension of the (minimal) algorithm in [Larsen]. The algorithm extends the one given in [Larsen] by, in addition to being a theorem prover, also being able to give an explanation of *why* two processes are not equivalent. The explanation given is a modal property which only one of the processes enjoy.

4.1 Construction and intuition

We will use the principle suggested in section 3.3, but avoid using side-effects. We will be needing some new concepts for convenience.

Definition 4.1.1 *$\mathcal{F}$ is a set of* argumented inequalities (AI) *if*

$$(p, q, F) \in \mathcal{F} \Rightarrow p \models F \ \wedge \ q \not\models F$$

F is said to argue *$p \not\sim q$. This will often be written as $\mathcal{F} \models p \not\sim q$.* □

It should be noted that $\mathcal{F} \not\models p \not\sim q$ does *not* imply that $p \sim q$, but rather that $\mathcal{F}$ cannot give a valid argument for $p \not\sim q$ i.e. $\forall F : (p, q, F) \notin \mathcal{F}$.

Notation 4.1.2 *In the following sections we will use the convention that an over lined variable in a PROLOG-clause or in an inference rule is intended as an output variable (for that clause/rule).* □

Note that $C = \overline{C}$, but $\overline{C}$ is more informative as it suggests that C is (locally) used as an output variable.

In the same manner as [Larsen] we are going to establish an operational based inference system for constructing bisimulations over **P** based on the derivation relation $\longrightarrow$ of **P**. Moreover, the inference system must also cover the construction of properties, i.e. construction of the set, $\mathcal{F}$, of argumented inequalities, in case a bisimulation cannot be found. We shall prove the soundness and restricted completeness of the new system. Just as the minimal system of [Larsen] this new version can readily be implemented in PROLOG.

Now let

$$bisim \subseteq Pr \times Pr \times 2^{(Pr\times Pr)} \times 2^{(Pr\times Pr\times M)}$$

$$\left.\begin{array}{l} closure \\ matchl \\ matchr \end{array}\right\} \subseteq Pr \times Pr \times 2^{(Pr\times Pr)} \times 2^{(Pr\times Pr)} \times 2^{(Pr\times Pr\times M)} \times 2^{(Pr\times Pr\times M)}$$

$$\left.\begin{array}{l} matchl^+ \\ matchr^+ \end{array}\right\} \subseteq Pr \times Pr \times 2^{(Act\times Pr)} \times 2^{(Pr\times Pr)} \times 2^{(Pr\times Pr)} \times 2^{(Pr\times Pr\times M)} \times 2^{(Pr\times Pr\times M)}$$

be the smallest relations closed under the rules given in figure 4.1. It is helpful to think of *bisim*, *closure*, *matchl*, and *matchr* as being function from the arguments not over lined to the over lined arguments. An informal description of each of the four functions can thus be:

bisim: Given two processes p and q, *bisim* will try to build a bisimulation, C, containing the pair (p, q). If this succeeds then $\mathcal{F}_{out} \not\models p \not\sim q$ otherwise $\mathcal{F}_{out} \models p \not\sim q$.

closure: Given two processes p and q and an approximate bisimulation, B, with $(p, q) \in B$, and a set of argumented inequivalences, $\mathcal{F}_{in}$, *closure* will try to extend B to a genuine bisimulation D. If this is impossible then $\mathcal{F}_{in}$ will be extended to a set of argumented inequivalences, $\mathcal{F}_{out}$, such that $\mathcal{F}_{out} \models p \not\sim q$.

matchl: Given two processes p and q and an approximate bisimulation, B, with $(p, q) \in B$, a subset, M, of p's derivations such that q only remains to match those derivations of p which still are in M, and a set of argumented inequivalences, $\mathcal{F}_{in}$, *matchl* will try to extend B to an approximate (bi)simulation, D, closed under $\mathcal{L}$. If this is impossible then *matchl* will extend $\mathcal{F}_{in}$ to a set of argumented inequivalences, $\mathcal{F}_{out}$, such that $\mathcal{F}_{out} \models p \not\sim q$.

matchr: Given two processes p and q and an approximate bisimulation, B, with $(p, q) \in B$, a subset, N, of q's derivations such that p only remains to match those derivations of q which still are in N, and a set of argumented inequivalences, $\mathcal{F}_{in}$, *matchr* will try to extend B to an approximate (bi)simulation, D, closed under $\mathcal{R}$. If this is impossible then *matchr* will extend $\mathcal{F}_{in}$ to a set of argumented inequivalences, $\mathcal{F}_{out}$, such that $\mathcal{F}_{out} \models p \not\sim q$.

4.1.1 Soundness

The system is sound in the sense that whenever $bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}})$ 'succeeds' (terminates) then either C is a bisimulation containing p and q, or $\mathcal{F}_{out}$ arguments their inequivalence. Now, we will introduce four verification conditions Bis, Cl, Ml, and Mr, and show that these verification conditions are closed under the rules of the inference system (see figure 4.1). Then we will have that the following inclusions hold:

$$\begin{array}{lclclcl} bisim & \subseteq & Bis & \quad & matchl & \subseteq & Ml \\ closure & \subseteq & Cl & \quad & matchr & \subseteq & Mr \end{array}$$

due to leastness of *bisim*, ... The soundness then follows directly.

$$B \ : \quad \frac{closure(p,q,\{(p,q)\},\overline{C},\emptyset,\overline{\mathcal{F}_{out}})}{bisim(p,q,\overline{C},\overline{\mathcal{F}_{out}})}$$

$$C \ : \quad \frac{matchl(p,q,M,B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchr(p,q,N,C,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{closure(p,q,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} \mathcal{F}_1 \not\models p \not\sim q \ \wedge \\ M = \{(a,p') \mid p \xrightarrow{a} p'\} \ \wedge \\ N = \{(a,q') \mid q \xrightarrow{a} q'\} \end{array} \right.$$

$$\frac{matchl(p,q,M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}{closure(p,q,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} \mathcal{F}_{out} \models p \not\sim q \ \wedge \\ M = \{(a,p') \mid p \xrightarrow{a} p'\} \end{array} \right.$$

$$ML \ : \quad \frac{}{matchl(p,q,\emptyset,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})}$$

$$\frac{matchl(p,q,M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} \exists q' : q \xrightarrow{a} q' \ \wedge \\ (p',q') \in B \ \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \end{array} \right.$$

$$\frac{closure(p',q',\{(p',q')\} \cup B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} q \xrightarrow{a} q' \ \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \\ \mathcal{F}_{in} \not\models p' \not\sim q' \\ \mathcal{F}_1 \not\models p' \not\sim q' \end{array} \right.$$

$$\frac{closure(p',q',\{(p',q')\} \cup B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchl(p,q,M,C,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} q \xrightarrow{a} q' \ \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \\ \mathcal{F}_{in} \not\models p' \not\sim q' \\ \mathcal{F}_1 \not\models p' \not\sim q' \end{array} \right.$$

$$\frac{}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{B},\mathcal{F}_{in},\{(p,q,\langle a \rangle F')\} \cup \mathcal{F}_{in})} \quad \left\{ \begin{array}{l} \mathcal{F}_{in} \not\models p \not\sim q \\ F' = \bigwedge\{F_1,\ldots,F_n\} \\ \text{where } \{q'_1,\ldots,q'_n\} = \{q' \mid q \xrightarrow{a} q'\} \\ \text{and } \forall i : (p',q'_i,F_i) \in \mathcal{F}_{in} \end{array} \right.$$

$$\frac{}{matchl(p,q,M,B,\overline{B},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})} \quad \{ \ \mathcal{F}_{in} \models p \not\sim q$$

$$MR \ : \quad \frac{}{matchr(p,q,\emptyset,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})}$$

$$\frac{matchr(p,q,N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}{matchr(p,q,\{(a,q')\} \cup N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} \exists q' : q \xrightarrow{a} q' \ \wedge \\ (p',q') \in B \ \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \end{array} \right.$$

$$\frac{closure(p',q',\{(p',q')\} \cup B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchr(p,q,\{(a,q')\} \cup N,B,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchr(p,q,\{(a,q')\} \cup N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} p \xrightarrow{a} p' \ \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \\ \mathcal{F}_{in} \not\models p' \not\sim q' \\ \mathcal{F}_1 \not\models p' \not\sim q' \end{array} \right.$$

$$\frac{closure(p',q',\{(p',q')\} \cup B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchr(p,q,N,C,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchr(p,q,\{(a,q')\} \cup N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{ \begin{array}{l} p \xrightarrow{a} p' \ \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \\ \mathcal{F}_{in} \not\models p' \not\sim q' \\ \mathcal{F}_1 \not\models p' \not\sim q' \end{array} \right.$$

$$\frac{}{matchr(p,q,\{(a,q')\} \cup N,B,\overline{B},\mathcal{F}_{in},\{(p,q,[a]F')\} \cup \mathcal{F}_{in})} \quad \left\{ \begin{array}{l} \mathcal{F}_{in} \not\models p \not\sim q \\ F' = \bigvee\{F_1,\ldots,F_n\} \\ \text{where } \{p'_1,\ldots,p'_n\} = \{p' \mid p \xrightarrow{a} p'\} \\ \text{and } \forall i : (p'_i,q',F_i) \in \mathcal{F}_{in} \end{array} \right.$$

$$\frac{}{matchr(p,q,N,B,\overline{B},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})} \quad \{ \ \mathcal{F}_{in} \models p \not\sim q$$

Figure 4.1: Inference rules for new system.

B : We want to find a bisimulation C containing the pair (p, q). We are starting off from 'nothing' i.e., we do not know whether $p \sim q$ is true or not. Also, we have no knowledge of any other processes being (in)equivalent, i.e. $\mathcal{F}_{in} = \emptyset$. So, we assume that $p \sim q$, i.e. $(p, q) \in B$ $(= \{(p, q)\})$ and try to 'close' $\{(p, q)\}$ with respect to (p, q). If this is not possible $\mathcal{F}_{out} \models p \not\sim q$.

C : We will try to 'close' B with respect to (p, q). This can be achieved by first assuring C is B's closure with respect to p's derivations, M, and then in turn clode C with respect to q's derivations, N, to obtain D using C_1. If B cannot be closed with respect to p's derivations, then $\mathcal{F}_1 \models p \not\sim q$ which is reported back through $\mathcal{F}_{out}$ of C_2.

ML_1 : If $M = \emptyset$ then B is already closed with respect to p's derivations and B is returned as the 'new' closure. Also, we have gained no further knowledge concerning inequivalence of processes, so we simply return $\mathcal{F}_{in}$ as our new knowledge.

ML_2 : We are concentrating on the derivation $p \xrightarrow{a} p'$. if we can find an a-derivation of q such that $q \xrightarrow{a} q'$ and $(p', q') \in B$ then we will choose this q'! Now, we only need to find matching derivations of q for those derivations of p still in M.

ML_3 : We want to collect as much inequivalence information as possible. Again, we are considering the derivation $p \xrightarrow{a} p'$. If we can find an a-derivation of q such that $q \xrightarrow{a} q'$ and $\mathcal{F}_1 \models p \not\sim q$, then we will try to close $\{(p', q')\} \cup B$ with respect to (p', q'). However, we want this to fail in order for $\mathcal{F}_1 \models p' \not\sim q'$ so that we can use this additional knowledge in future work.

ML_4 : Again, we are considering the derivation $p \xrightarrow{a} p'$. If we can find an a-derivation of q such that $q \xrightarrow{a} q'$ and $\mathcal{F}_1 \not\models p \not\sim q$, then we will try to close $\{(p', q')\} \cup B$ with respect to (p', q'). If we succeed, then we can concentrate on the remaining derivations of p, M, with the extended knowledge contained in C and $\mathcal{F}_1$.

ML_5 : We know that q cannot match p's a-derivation p' and must try to find a valid argument. The argument is constructed from our inequivalence knowledge, $\mathcal{F}_{in}$, containing arguments of inequivalence of p' and q'_i $\forall i \leq n$ where $\{q_1, \ldots, q_n\} = \{q' \mid q \xrightarrow{a} q'\}$ (please refer to section 3.3 for further details.

ML_5 : Our 'secret' door out whenever we know $p \not\sim q$.

MR_i : Similar to ML_i.

Figure 4.1(cont.): Intuitive description of the rules.

From the informal description of *bisim*, …we can determine what the verification conditions at least ought to satisfy e.g. for *closure Cl* ought to be true whenever $\mathcal{F}_{in}$ is AI, $\mathcal{F}_{out}$ is AI, $\mathcal{F}_{in} \subseteq \mathcal{F}_{out}$, and if $\mathcal{F}_{out} \not\models p \not\sim q$ then D should (at least) be an approximate bisimulation whenever B is. The actual verification conditions are given in figure 4.2.

Note that we in these verification conditions actually use a very weak

$$
\begin{array}{l}
Bis(p,q,\overline{C},\overline{\mathcal{F}_{out}}) \overset{\Delta}{\Leftrightarrow} \\
\qquad \{\} \Rightarrow \\
\qquad \left[\begin{array}{l} (\,(p,q) \in \overline{C} \;\wedge\; \overline{C} \subseteq \mathcal{B}(\overline{C})\,) \vee \\ (\overline{\mathcal{F}_{out}} \models p \not\sim q) \;\wedge\; \overline{\mathcal{F}_{out}} \text{ is AI} \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
Cl(p,q,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}}) \overset{\Delta}{\Leftrightarrow} \\
\qquad \left\{ \begin{array}{l} (p,q) \in B \wedge \\ \mathcal{F}_{in} \text{ is AI} \end{array} \right\} \Rightarrow \\
\qquad \left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is AI} \;\wedge\; \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus (B \setminus \{(p,q)\}) \subseteq \mathcal{B}(\overline{D}) \end{array} \right] \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
Ml(p,q,M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}}) \overset{\Delta}{\Leftrightarrow} \\
\qquad \left\{ \begin{array}{l} (p,q) \in B \wedge \\ M \subseteq \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_M,q) \in \mathcal{L}(B) \;\wedge\; \mathcal{F}_{in} \text{ is AI} \end{array} \right\} \Rightarrow \\
\qquad \left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is AI} \;\wedge\; \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus B \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{L}(\overline{D}) \end{array} \right] \end{array} \right]
\end{array}
$$

$$
\begin{array}{l}
Mr(p,q,M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}}) \overset{\Delta}{\Leftrightarrow} \\
\qquad \left\{ \begin{array}{l} (p,q) \in B \wedge \\ N \subseteq \{(a,q') \mid q \xrightarrow{a} q'\} \wedge \\ (p,q_N) \in \mathcal{R}(B) \;\wedge\; \mathcal{F}_{in} \text{ is AI} \end{array} \right\} \Rightarrow \\
\qquad \left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is AI} \;\wedge\; \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus B \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{R}(\overline{D}) \end{array} \right] \end{array} \right]
\end{array}
$$

where $p_M = \sum\{(a.p' \mid p \xrightarrow{a} p' \;\wedge\; (a,p') \notin M\}$.

Figure 4.2: Verification conditions for the new system.

notion of approximate bisimulations as we only require that $(p, q) \in B$ in order for B to be an approximate bisimulation for (p, q). This, in fact, makes the verification conditions stronger than they would have been using the definition of approximate bisimulation.

We are now able to prove that Bis, Cl, Ml, and Mr are closed under the inference rules and thus we obtain the following *soundness* theorem

Theorem 4.1.3 ***[Soundness]***

$$bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}}) \Rightarrow \left[\begin{array}{l} (\,(p, q) \in \overline{C} \ \wedge \ \overline{C} \subseteq \mathcal{B}(\overline{C})\,) \vee \\ (\overline{\mathcal{F}_{out}} \models p \not\sim q) \ \wedge \ \overline{\mathcal{F}_{out}} \text{ is } AI \end{array} \right]$$

PROOF: *It is obvious that the four relations are mutually dependent, thus, in order to complete the soundness proof an (simultaneous) induction proof is needed. The induction principle associated with the inference system is straight forward (see [Aczel]) and the outline of the proof, naturally, resembles that of [Larsen].*

The rules are considered in turn. (It is advisory to have the two figures 4.1 on page 49 and 4.2 on page 51 at hand while reading the proof).

First a word of notation; throughout the following sections we will write aCL instead of 'the antecedent of Cl' and cCl instead of 'the conclusion of Cl'.

Proof of rule B: We must show

$$Cl(p, q, \{(p, q)\}, \overline{C}, \emptyset, \overline{\mathcal{F}_{out}}) \Rightarrow Bis(p, q, \overline{C}, \overline{\mathcal{F}_{out}})$$

or equivalently that

$$Cl(p, q, \{(p, q)\}, \overline{C}, \emptyset, \overline{\mathcal{F}_{out}}) \Rightarrow \left[\begin{array}{l} (\,(p, q) \in \overline{C} \ \wedge \ \overline{C} \subseteq \mathcal{B}(\overline{C})\,) \vee \\ (\overline{\mathcal{F}_{out}} \models p \not\sim q \ \wedge \ \overline{\mathcal{F}_{out}} \text{ is } AI) \end{array} \right]$$

But this follows immediately from the definition of Cl as

$$aCl \left\{ \begin{array}{l} (p, q) \in \{(p, q)\} \ \wedge \\ \emptyset \text{ is } AI \end{array} \right\}$$

clearly holds.

Proof of rule C_1: Assume that aCL, $aMl \Rightarrow cMl$, and $aMr \Rightarrow cMr$ holds. We must show that cCl holds. First

$$\begin{array}{ll} aCl & \left\{ \begin{array}{l} (p,q) \in B \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array} \right\} \Rightarrow \\ aMl & \left\{ \begin{array}{l} (p,q) \in B \ \wedge \ M \subseteq \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_M, q) \in \mathcal{L}(B) \ \wedge \ \mathcal{F}_{in} \text{ is } AI \end{array} \right\} \end{array}$$

This follows immediately, as $M = \{(a,p') \mid p \xrightarrow{a} p'\}$ by the side-condition and $(p_M, q) = (NIL, q) \in \mathcal{L}(B)$ as anything can simulate NIL. We must now prove that aCl, aMl and

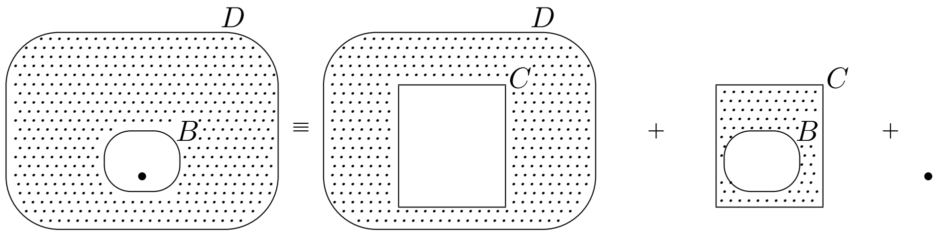
$$\begin{array}{ll} cMl & \left[\begin{array}{l} \overline{\mathcal{F}_1} \text{ is } AI \ \wedge \ \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_1} \ \wedge \\ (\overline{\mathcal{F}_1} \not\models p \not\sim q) \Rightarrow \left[\begin{array}{l} B \subseteq \overline{C} \ \wedge \ \overline{C} \setminus B \subseteq \mathcal{B}(\overline{C}) \wedge \\ (p,q) \in \mathcal{L}(\overline{C}) \end{array} \right] \end{array} \right] \Rightarrow \\ aMr & \left\{ \begin{array}{l} (p,q) \in C \ \wedge \ N \subseteq \{(a,q') \mid q \xrightarrow{a} q'\} \wedge \\ (p, q_N) \in \mathcal{R}(C) \ \wedge \ \mathcal{F}_1 \text{ is } AI \end{array} \right\} \end{array}$$

This follows from the side-condition, $\mathcal{F}_1 \not\models p \not\sim q$, and from the fact that $(p,q) \in B$ according to aMl and $B \subseteq C$ by cMl. Furthermore, $N = \{(a,q') \mid q \xrightarrow{a} q'\}$ and $(p, q_N) = (p, NIL) \in \mathcal{R}(C)$. It now remains to show that aCl, aMl, cMl, aMr, and

$$\begin{array}{ll} cMr & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \ \wedge \ \mathcal{F}_1 \subseteq \overline{\mathcal{F}_{out}} \ \wedge \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow \left[\begin{array}{l} C \subseteq \overline{D} \ \wedge \ \overline{D} \setminus C \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{R}(\overline{D}) \end{array} \right] \end{array} \right] \\ cCl & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \ \wedge \ \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} \ \wedge \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus (B \setminus \{(p,q)\}) \subseteq \mathcal{B}(\overline{D}) \end{array} \right] \end{array} \right] \end{array}$$

$\overline{\mathcal{F}_{out}}$ is AI by cMr. $\overline{\mathcal{F}_{in}} \subseteq \overline{\mathcal{F}_{out}}$ and $B \subseteq$ follows from cMl and cMr. We can end this part of the proof by using the following rewriting

$$D \setminus (B \setminus \{(p,q)\}) \ = \quad (D \setminus C) \qquad \cup \qquad (C \setminus B) \ \cup \ \{(p,q)\}$$



$D \setminus C \subseteq \mathcal{B}(D)$ by cMr, $(C \setminus B) \subseteq \mathcal{B}(C)$ by cMl, but $C \subseteq D$ by cMr and $\mathcal{B}$ is monotone i.e. $(C \setminus B) \subseteq \mathcal{B}(D)$. Notice, that $(p,q) \in \mathcal{L}(C)$ by cMl, $C \subseteq D$ by cMr i.e. $(p,q) \in D$ and by the monotony of $\mathcal{L}$ it follows that $(p,q) \in \mathcal{L}(D)$. Moreover, $(p,q) \in \mathcal{R}(D)$ by cMr and as $\mathcal{B}(D) = \mathcal{L}(D) \cap \mathcal{R}(D)$, then $(p,q) \in \mathcal{B}(D)$, so we can conclude that $D \setminus (B \setminus \{(p,q)\}) \subseteq \mathcal{B}(B)$.

Proof of rule C_2: Assume that aCl and $aMl \Rightarrow cMl$ holds. We must show that cCl holds. Trivially, we have $aCl \Rightarrow aMl$ (see proof of C_1). Then, due to the side-condition $\mathcal{F}_{out} \models p \not\sim q$, we only have to show that aCl, aMl, and

$$\begin{array}{ll} cMl & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \\ \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} \end{array}\right] \Rightarrow \\ cCl & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \\ \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} \end{array}\right] \end{array}$$

which holds trivially. Note that this rule in fact is necessary; assume that we only had rule C_1 with the side-condition $\mathcal{F}_1 \not\models p \not\sim q$ removed. Then we would not be able to state anything about C in case $\mathcal{F}_1 \models p \not\sim q$ and thus the system cannot be sound!

Proof of rule ML_1: We must show $Ml(p, q, \emptyset, B, \overline{B}, \mathcal{F}_{in}, \overline{\mathcal{F}_{in}})$ i.e.

$$\begin{array}{ll} aMl & \left\{\begin{array}{l} (p,q) \in B \wedge \\ \emptyset \subseteq \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_\emptyset, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array}\right\} \Rightarrow \\ cMl & \left[\begin{array}{ll} \overline{\mathcal{F}_{in}} \text{ is } AI \wedge \ \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{in}} \wedge & \\ (\overline{\mathcal{F}_{in}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B \subseteq \overline{B} \wedge \\ \overline{B} \setminus B \subseteq \mathcal{B}(\overline{B}) \wedge \\ (p,q) \in \mathcal{L}(\overline{B}) \end{array}\right] \end{array}\right] \end{array}$$

which clearly holds, as $(p_\emptyset, q) \in \mathcal{L}(B) \Leftrightarrow (p,q) \in \mathcal{L}(B)$. This rule is used whenever $M = \emptyset$ i.e. there are no more derivations of p left for q to match.

Proof of rule ML_2: Assume that aML and $aMl' \Rightarrow cMl'$ holds. (Note that we use a prime '$'$' to denote the second Ml-call in the rule). We must show *that cMl holds but as $cMl' = cMl$ it is sufficient to show $aMl \Rightarrow aMl'$, i.e.*

$$\begin{array}{ll} aMl & \left\{\begin{array}{l} (p,q) \in B \wedge \\ \{(a,p')\} \cup M \ \subseteq \ \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_{\{(a,p')\} \cup M}, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array}\right\} \Rightarrow \\ aMl' & \left\{\begin{array}{l} (p,q) \in B \wedge \\ M \ \subseteq \ \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_M, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array}\right\} \end{array}$$

Only $(p_M, q) \in \mathcal{L}(B)$ does not follow immediately. But note that

$$p_M = p_{\{(a,p')\} \cup M} + a.p'$$

and by the side-conditions for ML_2, $q \xrightarrow{a} q' \ \wedge \ (p', q') \in B$, we see that $(a.p', q) \in \mathcal{L}(B)$ and thus we can conclude that $(p_M, q) \in \mathcal{L}(B)$. This rule is used whenever q already has a match in B for the derivation (a, p').

Proof of rule ML_3: Assume that aMl, $aCl \Rightarrow cCl$, and $aMl' \Rightarrow cMl'$ holds. We must show that cMl holds. First,

$$\begin{array}{ll} aMl & \left\{\begin{array}{l} (p,q) \in B \wedge \\ \{(a,p')\} \cup M \ \subseteq \ \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_{\{(a,p')\} \cup M}, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array}\right\} \Rightarrow \\ aCl & \left\{\begin{array}{l} (p',q') \in \{(p',q')\} \cup B \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array}\right\} \Rightarrow \end{array}$$

which is trivially true. Furthermore, we must show that aMl, aCl, and

$$\begin{array}{ll} cCl & \left[\begin{array}{l} \mathcal{F}_{in} \text{ is } AI \ \wedge \\ \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_1} \\ \overline{\mathcal{F}_1} \not\models p \not\sim q \end{array}\right] \Rightarrow \\ aMl' & \left\{\begin{array}{l} (p,q) \in B \wedge \\ \{(a,p')\} \cup M \ \subseteq \ \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_{\{(a,p')\} \cup M}, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_1 \text{ is } AI \end{array}\right\} \end{array}$$

which follows directly from the side-condition for ML_3, cCl, and aMl. (Notice how the side-condition simplifies cCl). We now just need to prove that *aMl*, *aCl*, *cCl*, *aMl'*, and

$$\begin{array}{ll} cMl' & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \mathcal{F}_1 \subseteq \overline{\mathcal{F}_{out}} \wedge \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus B \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{L}(\overline{D}) \end{array}\right] \end{array}\right] \Rightarrow \\ cMl & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} \wedge \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus B \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{L}(\overline{D}) \end{array}\right] \end{array}\right] \end{array}$$

$\mathcal{F}_{in} \subseteq \mathcal{F}_1$ *according to cCl and* $\mathcal{F}_1 \subseteq \mathcal{F}_{out}$ *by cMl'*, *so* $\mathcal{F}_{in} \subseteq \mathcal{F}_{out}$ *holds. The rest follows trivially. Notice that this rule is used for one purpose only; to collect all information possible concerning equivalence of p's and q's derivations.*

Proof of rule ML_4*: Assume that aMl, aCl* $\Rightarrow$ *cCl, and aMl'* $\Rightarrow$ *cMl' holds. then we must show that cMl holds. Clearly, aMl* $\Rightarrow$ *aCl (see proof of* ML_3*). Thus we must prove that aMl, aCl, and (by the side-conditions of* ML_4*)*

$$\begin{array}{ll} cCl & \left[\begin{array}{l} \overline{\mathcal{F}_1} \text{ is } AI \;\; \wedge \; \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_1} \\ \{(p',q')\} \cup B \subseteq \overline{C} \wedge \\ \overline{C} \setminus (\; (B \cup \{(p',q')\} \setminus \{(p',q')\}) \subseteq \mathcal{B}(\overline{C}) \;) \end{array}\right] \Rightarrow \\ aMl' & \left\{\begin{array}{l} (p,q) \in C \wedge \\ M \;\subseteq\; \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_M, q) \in \mathcal{L}(C) \wedge \\ \mathcal{F}_1 \text{ is } AI \end{array}\right\} \end{array}$$

$(p,q) \in B$ *by aMl and* $B \subseteq C$ *by cCl, so we have that* $(p,q) \in C$*. Only* $(p_M, q) \in \mathcal{L}(C)$ *does not follow immediately. Note that*

$$p_M = p_{\{(a,p')\} \cup M} + a.p'$$

as by aMl we know that $(p_{\{(a,p')\} \cup M}, q) \in \mathcal{L}(B)$ *and* $B \subseteq C$ *by cCl and by the monotony of* $\mathcal{L}$ *we can thus deduce* $(p_{\{(a,p')\} \cup M}, q) \in \mathcal{L}(C)$*. By cCl we get* $(p',q') \in C$ *and in conjunction with the side-condition for rule* ML_4*,* $q \xrightarrow{a} q'$*, we conclude that* $(a.p', q) \in \mathcal{L}(C)$*.*

In order to end the proof of rule ML_4*, we must show that aMl, aCl, cCl,*

aMl', and

$$\begin{array}{ll} cMl' & \left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \mathcal{F}_1 \subseteq \overline{\mathcal{F}_{out}} \wedge & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} C \subseteq \overline{D} \wedge \\ \overline{D} \setminus C \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{L}(\overline{D}) \end{array}\right] \end{array}\right] \Rightarrow \\ cMl & \left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} \wedge & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B \subseteq \overline{D} \wedge \\ \overline{D} \setminus B \subseteq \mathcal{B}(\overline{D}) \wedge \\ (p,q) \in \mathcal{L}(\overline{D}) \end{array}\right] \end{array}\right] \end{array}$$

By cCl we know that $\mathcal{F}_{in} \subseteq \overline{\mathcal{F}_1}$ and cMl' gives us $\mathcal{F}_1 \subseteq \overline{\mathcal{F}_{out}}$, i.e. $\mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}}$. Similarly, we get $B \subseteq D$. Finally,

$$D \setminus B = (D \setminus C) \cup (C \setminus B)$$

Now, $(D \setminus C) \subseteq \mathcal{B}(D)$ by cMl'. $(C \setminus B) \subseteq \mathcal{B}(C)$ by cCl and $C \subseteq D$ by cMl'. By the monotony of $\mathcal{B}$ we therefore know that $(C \setminus B) \subseteq \mathcal{B}(D)$. so we can conclude that $(D \setminus B) \subseteq \mathcal{B}(D)$. This rule tries to 'close' $B \cup \{(p', q')\}$ with respect to (p', q') before dealing with the remaining derivations of M. (Backtracking may be necessary at this point should it later be discovered that $p' \not\sim q'$).

Proof of rule ML_5: We must prove $aMl \Rightarrow cMl$ i.e. we must show that

$$\begin{array}{ll} aMl & \left\{\begin{array}{l} (p,q) \in B \wedge \\ \{(a,p')\} \cup M \ \subseteq \ \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_{\{(a,p')\} \cup M}, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array}\right\} \Rightarrow \\ cMl & \left[\begin{array}{l} \overline{\mathcal{F}_{out}} \text{ is } AI \wedge \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{out}} \wedge \\ \overline{\mathcal{F}_{out}} \not\models p \not\sim q \end{array}\right] \end{array}$$

Notice, that cMl can be reduced as shown above as by the side-condition we know that $\mathcal{F}_{out}$ must contain a triple (p, q, F) arguing the inequivalence of p and q. The proof then follows immediately from the earlier discussed construction of $\mathcal{F}_{out}$ (see section 3.3). This rule is used to construct an argument for $p \not\sim q$ whenever q cannot match the derivation (a, p').

Proof of rule ML_6: We must prove that $aMl \Rightarrow cMl$ when $\mathcal{F}_{in} \models p \not\sim q$, i.e. we *get this reduced implication*

$$\begin{array}{ll} aMl & \left\{ \begin{array}{l} (p,q) \in B \wedge \\ M \subseteq \{(a,p') \mid p \xrightarrow{a} p'\} \wedge \\ (p_M, q) \in \mathcal{L}(B) \wedge \\ \mathcal{F}_{in} \text{ is } AI \end{array} \right\} \Rightarrow \\ cMl & \left[\begin{array}{l} \overline{\mathcal{F}_{in}} \text{ is } AI \wedge \mathcal{F}_{in} \subseteq \overline{\mathcal{F}_{in}} \wedge \\ \overline{\mathcal{F}_{in}} \models p \not\sim q \end{array} \right] \end{array}$$

This clearly holds. Thus, this rule is out 'secret door' out whenever $\mathcal{F}_{in}$ already argues $p \not\sim q$.

Proof of rule MR_1–MR_6: Similar to ML_1–ML_6 and therefore omitted.

This concludes the soundness proof of the inference system. □

By the above theorem we now know that whenever the *bisim*-predicate 'succeeds' (terminates) this implies that the two processes in question are either equivalent—thus implying that C is a bisimulation—or inequivalent—implying that $\mathcal{F}_{out}$ argues their inequivalence. We would, however, also like to be certain that no matter which process we choose the *bisim*-predicate *will* 'succeed'. This is to say, that we would like the system to be *complete*.

4.1.2 Completeness

Before turning to the completeness proof of the system we will introduce some additional convenient concepts.

Definition 4.1.4 *For $p \in Pr$ we define*

$$DER(p) = \{p' \in Pr \mid \exists s \in Act^* : p \xrightarrow{s} p'\}$$

which also will be defined on subsets S of $Act \times Pr$, such that

$$DER(S) = \{p' \in Pr \mid \exists (a,p) \in S\ \exists s \in Act^* : p \xrightarrow{s} p'\}$$

□

Definition 4.1.5 *The proces system* $\mathbf{P} = (Pr, Act, \longrightarrow)$ *is finite iff* Pr *and* Act *both are finite.* $\Box$

Definition 4.1.6 *A process* p *has* finite state-transition diagram *iff*

$$\mathbf{P} \downarrow DER(p)$$

is a finite transition system. $\Box$

Notice, that if p has a finite state-transition diagram then p will have a finite number of states and each state will have finite branching (i.e. have a finite number of possible actions).

Now, by using the induction principle associated with the inference system one more time, it is easy to show that the following finiteness conditions hold:

$$bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}}) \Rightarrow \qquad \overline{C} \text{ and } \overline{\mathcal{F}_{out}} \text{ are finite}$$

$$\begin{array}{l} closure(p, q, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}}) \Rightarrow \\ \quad \left\{ \begin{array}{l} B \text{ is finite} \wedge \\ \mathcal{F}_{in} \text{ is finite} \end{array} \right\} \Rightarrow \qquad \overline{C} \text{ and } \overline{\mathcal{F}_{out}} \text{ are finite} \end{array}$$

$$\begin{array}{l} matchl(p, q, M, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}}) \Rightarrow \\ \quad \left\{ \begin{array}{l} B \text{ is finite} \wedge \\ M \text{ is finite} \wedge \\ \mathcal{F}_{in} \text{ is finite} \end{array} \right\} \Rightarrow \qquad \overline{C} \text{ and } \overline{\mathcal{F}_{out}} \text{ are finite} \end{array}$$

$$\begin{array}{l} matchr(p, q, N, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}}) \Rightarrow \\ \quad \left\{ \begin{array}{l} B \text{ is finite} \wedge \\ N \text{ is finite} \wedge \\ \mathcal{F}_{in} \text{ is finite} \end{array} \right\} \Rightarrow \qquad \overline{C} \text{ and } \overline{\mathcal{F}_{out}} \text{ are finite} \end{array}$$

However, this fact causes troubles as the most direct way to show the completeness of the system, i.e. to show the following inclusions:

$$\begin{array}{lllllll} Bis & \subseteq & bisim & \qquad & Ml & \subseteq & matchl \\ Cl & \subseteq & closure & \qquad & Mr & \subseteq & matchr \end{array}$$

is not possible, because Bis, Cl, Ml, and Mr do not satisfy these finiteness conditions. Therefore, the above inclusions are not valid.

We will, however, be content with the inclusions to hold whenever the preconditions of the verification conditions hold and the size of the input-arguments are within appropriate limits. We will thus be content if implications of the following kind hold:

$$\left[\begin{array}{l} aCl(p,q,B,\_,\mathcal{F}_{in},\_) \wedge \\ \parallel \text{input-arguments} \parallel \ \leq n \end{array}\right] \ \Rightarrow \ \exists C \exists \mathcal{F}_{out} : closure(p,q,B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})$$

The size-function for each of the verification conditions is defined by:

$$\sharp_{Bis}(p,q,\_,\_) = \\ \qquad \parallel DER(p) \times DER(q) \parallel$$

$$\sharp_{Cl}(p,q,B,\_,\mathcal{F}_{in},\_) = \\ \qquad \parallel DER(p) \times DER(q) \setminus B \setminus \{(p,q) \mid (p,q,F) \in \mathcal{F}_{in}\} \parallel + 1$$

$$\sharp_{Ml}(\_,q,M,B,\_,\mathcal{F}_{in},\_) = \\ \qquad \parallel DER(M) \times DER(q) \setminus B \setminus \{(p,q) \mid (p,q,F) \in \mathcal{F}_{in}\} \parallel + 1$$

$$\sharp_{Mr}(p,\_,N,B,\_,\mathcal{F}_{in},\_) = \\ \qquad \parallel DER(N) \times DER(p) \setminus B \setminus \{(p,q) \mid (p,q,F) \in \mathcal{F}_{in}\} \parallel + 1$$

Notice, the size functions depend only on the input-arguments—which is why the other arguments are just indicated by an '_'. For $\sharp_{Cl}$, for instance, we have that $DER(p) \times DER(q)$ is a set of state-pairs to investigate. B is the already constructed part of the bisimulation, i.e. all pairs $(p,q) \in B$ do not be processed again. $\{(p,q) \mid (p,q,F) \in \mathcal{F}_{in}\}$ is the set of state-pairs already found to be inequivalent so they do not need any further processing either. $DER(p) \times DER(q) \setminus B \setminus \{(p,q) \mid (p,q,F) \in \mathcal{F}_{in}\}$ is thus the maximal set of state-pairs which remains to be investigated. Also, note that $\sharp_{Ml}$ ($\sharp_{Mr}$) is independent of input-argument p (q), instead the set of derivations M (N) of p (q) each of which has not yet been matched by a derivation of q (p) is used.

We claim the inference system is complete for processes with finite state-transition diagram. This claim will follow as a corollary from the following theorem.

Theorem 4.1.7 *If the processes p and q have finite state-transition dia-*

grams, then $\forall n \geq 1$*:*

$$\text{1)} \quad [\, \sharp_{Bis}(p,q,\_,\_) \leq n \,] \;\Rightarrow\; \exists C \exists \mathcal{F}_{out} : bisim(p,q,\overline{C},\overline{\mathcal{F}_{out}})$$

$$\text{2)} \quad \left[\begin{array}{l} aCl(p,q,B,\_,\mathcal{F}_{in},\_) \wedge \\ \sharp_{Cl}(p,q,B,\_,\mathcal{F}_{in},\_) \leq n \end{array} \right] \Rightarrow \exists C \exists \mathcal{F}_{out} : closure(p,q,B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})$$

$$\text{3)} \quad \left[\begin{array}{l} aMl(\_,q,M,B,\_,\mathcal{F}_{in},\_) \wedge \\ \sharp_{Ml}(\_,q,M,B,\_,\mathcal{F}_{in},\_) \leq n \end{array} \right] \Rightarrow \exists C \exists \mathcal{F}_{out} : matchl(p,q,M,B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})$$

$$\text{4)} \quad \left[\begin{array}{l} aMr(p,\_,N,B,\_,\mathcal{F}_{in},\_) \wedge \\ \sharp_{Mr}(p,\_,N,B,\_,\mathcal{F}_{in},\_) \leq n \end{array} \right] \Rightarrow \exists C \exists \mathcal{F}_{out} : matchr(p,q,N,B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})$$

PROOF: *We will prove the theorem by proving the implications—starting with (4) ending with (1)—by induction on* n*, with sub induction on* $\| M \|$ *(*$\| N \|$*) for (4) and (3). Thus is each step we will first establish (4) and (3), use these results for establishing (2) from witch (1) easily follows.*

$n = 1$*:*

We must prove that the implications (4)–(1) hold for $n = 1$*.*

(4): *Proved by sub induction on the size of* N *(*$\| N \|$*). So assume*

$$\left. \begin{array}{l} aMr(p,\_,N,B,\_,\mathcal{F}_{in},\_) \wedge \\ \sharp_{Mr}(p,\_,N,B,\_,\mathcal{F}_{in},\_) \leq 1 \end{array} \right\} \Rightarrow$$

$$(DER(p) \times DER(N) \setminus B) \setminus \{(p,q) \mid \mathcal{F}_{in} \models p \not\sim q\} = \emptyset \Leftrightarrow$$

$$DER(p) \times DER(N) \subseteq B \cup \{(p,q) \mid \mathcal{F}_{in} \models p \not\sim q\}$$

$\| N \| = 0$*: Ok, as inference rule* MR_1 *(which is an axiom) is applicable.*
Sub induction hypothesis: Implication (4) holds for $\| N \| = m$*.*

Sub step $N = N' \cup \{(a, q')\}$ *where* $\| N \| = m + 1$, $\| N' \| = m$*: Thus, we assume that (4) holds for* N'.

Now, if $\mathcal{F}_{in} \models p \not\sim q$*, then all is well, as we simply apply rule (axiom)* MR_6 *obtaining*

$$matchr(p, q, N, B, \overline{B}, \mathcal{F}_{in}, \overline{\mathcal{F}_{in}})$$

So assume that $\mathcal{F}_{in} \not\models p \not\sim q$ *and* $p \xrightarrow{a} q$ *such that* $(p', q') \in B$*. Then the side-conditions of rule* MR_2 *are satisfied and clearly*

$$\begin{aligned} &\sharp_{Mr}(p, \_, N', B, \_, \mathcal{F}_{in}, \_) \;\le \\ &\qquad \sharp_{Mr}(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) = 1 \end{aligned}$$

From the soundness proof we know

$$\begin{aligned} &aMr(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \;\Rightarrow \\ &\qquad aMr(p, \_, N', B, \_, \mathcal{F}_{in}, \_) \end{aligned}$$

Thus by the sub induction hypothesis $(\| N' \| < \| N |)$ *we have that*

$$matchr(p, q, N', B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

holds for some D *and* $\mathcal{F}_{out}$*. Then using* MR_2 *we have that the predicate*

$$matchr(p, q, \{(a, q')\} \cup N', B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

holds.

Assume $\mathcal{F}_{in} \not\models p \not\sim q$ *and* $(p', q') \notin B$ *for* $p \xrightarrow{a} p'$*. As* $n = 1$ *(by induction hypothesis) we thus have that*

$$(p', q') \;\in\; DER(p) \times DER(q) \;\subseteq\; B \cup \{(p, q) \mid \mathcal{F}_{in} \models p \not\sim q\}$$

which gives us $\mathcal{F}_{in} \models p' \not\sim q'$*. Thus, rules* MR_3 *and* MR_4 *are not applicable. However, rule* MR_5 *can be applied. Moreover, this rule is an axiom, so*

$$matchr(p, q, N, B, \overline{B}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

holds for $\mathcal{F}_{out}$ *constructed as discussed in section 3.3.*

This concludes the proof of implication (4) for $n = 1$.

(3): *Similar to the proof of (4).*

(2): *We have shown that (4) and (3) holds for $n = 1$. We must show that (2) holds for $n = 1$. So assume*

$$\left[\begin{array}{l} aCl(p, q, B, \_, \mathcal{F}_{in}, \_) \wedge \\ \sharp_{Cl}(p, q, B, \_, \mathcal{F}_{in}, \_) \ \le 1 \end{array}\right]$$

From the soundness proof we know

$$aCl(p, q, B, \_, \mathcal{F}_{in}, \_) \ \Rightarrow \\ aMl(p, q, M, B, \_, \mathcal{F}_{in}, \_)$$

Clearly, $DER(M) \le DER(p)$ so

$$\sharp_{Ml}(\_, q, M, B, \_, \mathcal{F}_{in}, \_) \ \le \\ \sharp_{Cl}(p, q, B, \_, \mathcal{F}_{in}, \_) \le 1$$

Since implication (3) has already been established for $n = 1$, we have for some C and $\mathcal{F}_1$ that

$$matchl(p, q, M, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_1}) \tag{4.1}$$

holds. If $\mathcal{F}_1 \models p \not\sim q$ then by using C_2 we can establish

$$closure(p, q, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_1})$$

Otherwise, we know from the soundness that $\mathcal{F}_{in} \subseteq \mathcal{F}_1$ and $B \subseteq C$ so

$$\sharp_{Mr}(p, q, N, B, \_, \mathcal{F}_{in}, \_) \ \le \\ \sharp_{Ml}(p, q, M, B, \_, \mathcal{F}_{in}, \_) \le 1$$

Thus from implication (4) we have for some D and $\mathcal{F}_{out}$ that

$$matchr(p, q, N, C, \overline{D}, \mathcal{F}_1, \overline{\mathcal{F}_{out}}) \tag{4.2}$$

holds. Thus, by using rule C_1 on 4.1 and 4.2 we obtain

$$closure(p, q, B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

(1): *Follows from (2) as*

$$\sharp_{Cl}(p, q, \{(p, q)\}, \_, \emptyset, \_) \le \sharp_{Bis}(p, q, \_, \_) \le 1$$

so for some C and $\mathcal{F}_{out}$

$$closure(p, q, \{(p, q)\}, \overline{C}, \emptyset, \overline{\mathcal{F}_{out}})$$

holds and thus

$$bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}})$$

can be established by using rule B.

Thus we have completed the proof for $n = 1$.

Induction hypothesis: Implications (4)–(1) holds for $n = k$.
Step $n = k + 1$:

We must show that the implications hold for the case $n = k + 1$ under the induction hypothesis.

(4): *Again, proved by sub induction on the size of N ($\| N \|$). So assume*

$$\left[\begin{array}{l} aMr(p, \_, N, B, \_, \mathcal{F}_{in}, \_) \wedge \\ \sharp_{Mr}(p, \_, N, B, \_, \mathcal{F}_{in}, \_) \;\leq\; k+1 \end{array} \right]$$

$\| N \| = 0$*: Ok, as inference rule MR_1 (which is an axiom) is applicable.*

Sub induction hypothesis: Implication (4) holds for $\| N \| = m$.
Sub step $N = N' \cup \{(a, q')\}$ where $\| N \| = m + 1$, $\| N' \| = m$:

Thus, we assume that (4) holds for N'.

Now, if $\mathcal{F}_{in} \models p \not\sim q$, then all is well, as we simply apply rule (axiom) MR_6.

So assume that $\mathcal{F}_{in} \not\models p \not\sim q$ and $p \xrightarrow{a} q$ such that $(p', q') \in B$. Then the side-conditions of rule MR_2 are satisfied and clearly

$$\begin{array}{l} \sharp_{Mr}(p, \_, N', B, \_, \mathcal{F}_{in}, \_) \;\leq \\ \qquad \sharp_{Mr}(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \leq n \end{array}$$

From the soundness proof we know

$$\begin{array}{c} aMr(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \;\Rightarrow \\ aMr(p, \_, N', B, \_, \mathcal{F}_{in}, \_) \end{array}$$

Now, by the sub induction hypothesis ($\| N' \| < \| N |$) we have that

$$matchr(p, q, N', B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

holds for some D and $\mathcal{F}_{out}$. Then using MR_2 we have that the predicate

$$matchr(p, q, \{(a, q')\} \cup N', B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

holds.

If $\mathcal{F}_{in} \models p' \not\sim q'$ for all p' such that $p \xrightarrow{a} p'$ then we simply use rule (axiom) $MR5$.

Otherwise, let $p \xrightarrow{a} p'$ such that $\mathcal{F}_{in} \not\models p \not\sim q$. Then either MR_3 or MR_4 is applicable. The thus have that

$$\sharp_{Cl}(p', q', \{(p', q')\} \cup B, \_, \mathcal{F}_{in}, \_) \ \le \\ \sharp_{Mr}(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \le n$$

as $\| B \| \le \| \{(p', q')\} \cup B \|$ and by the soundness proof

$$aMr(p, q, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \ \Rightarrow \\ aCl(p', q', \{(p', q')\} \cup B, \_, \mathcal{F}_{in}, \_)$$

Thus, by the induction hypothesis we have that

$$closure(p', q', \{(p', q')\} \cup B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_1}) \tag{4.3}$$

holds for some C and $\mathcal{F}_1$.

Assume $\mathcal{F}_1 \models p \not\sim q$, then MR_3 is applicable and clearly

$$\sharp_{Mr}(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_1, \_) \ < \\ \sharp_{Mr}(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \le n$$

as $\| \mathcal{F}_{in} \| < \| \mathcal{F}_1 \|$. Furthermore, from the soundness proof we know that

$$aMr(p, q, \{(a, q')\} \cup N', B, \_, \mathcal{F}_1, \_)$$

holds and thus by the induction hypothesis

$$matchr(p, q, \{(a, q')\} \cup N', B, \overline{D}, \mathcal{F}_1, \overline{\mathcal{F}_{out}}) \tag{4.4}$$

holds for some D and $\mathcal{F}_{out}$. So by using MR_3 on 4.3 and 4.4 we have that

$$matchr(p, q, \{(a, q')\} \cup N', B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}}$$

holds.

However, if $\mathcal{F}_1 \not\models p' \not\sim q'$, *then rule* MR_4 *is applicable and from the soundness we have that* $B \subset C$ *and* $\mathcal{F}_{in} \subseteq \mathcal{F}_1$ *so clearly*

$$\begin{aligned}&\sharp_{Mr}(p, \_, N', C, \_, \mathcal{F}_1, \_) \ < \\ &\qquad \sharp_{Mr}(p, \_, \{(a, q')\} \cup N', B, \_, \mathcal{F}_{in}, \_) \leq n\end{aligned}$$

Again, from the soundness proof we know that

$$aMr(p, q, N', C, \_, \mathcal{F}_1, \_)$$

holds. Thus, by the sub induction hypothesis we have that

$$matchr(p, q, N', C, \overline{D}, \mathcal{F}_1, \overline{\mathcal{F}_{out}}) \tag{4.5}$$

holds for some D *and* $\mathcal{F}_{out}$. *Finally, by using rule* MR_4 *on* 4.3 *and* 4.5 *we establish*

$$matchr(p, q, \{(a, q')\} \cup N', C, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

This completes the proof of implication (4).

(3): *Similar to the proof of (4).*

(2): *We have shown that (4) and (3) hold for* $n \leq k+1$. *We must now show that (2) hold for* $n \leq k+1$. *So assume*

$$\left[\begin{array}{l} aCl(p, q, B, \_, \mathcal{F}_{in}, \_) \wedge \\ \sharp_{Cl}(p, q, B, \_, \mathcal{F}_{in}, \_) \ \leq n \end{array}\right]$$

From the soundness proof we get

$$\begin{aligned}&aCl(p, q, B, \_, \mathcal{F}_{in}, \_) \ \Rightarrow \\ &\qquad aMl(p, q, M, B, \_, \mathcal{F}_{in}, \_)\end{aligned}$$

Clearly, $DER(M) \leq DER(p)$ *so*

$$\begin{aligned}&\sharp_{Ml}(\_, q, M, B, \_, \mathcal{F}_{in}, \_) \ \leq \\ &\qquad \sharp_{Cl}(p, q, B, \_, \mathcal{F}_{in}, \_) \leq n\end{aligned}$$

Since implication (3) has already been established for $n = k+1$, *we have for some* C *and* $\mathcal{F}_1$ *that*

$$matchl(p, q, M, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_1}) \tag{4.6}$$

holds. If $\mathcal{F}_1 \models p \not\sim q$ then by using C_2 we can establish

$$closure(p, q, B, \overline{C}, \mathcal{F}_{in}, \overline{\mathcal{F}_1})$$

Otherwise, we know from the soundness that $\mathcal{F}_{in} \subseteq \mathcal{F}_1$ and $B \subseteq C$ so

$$\begin{aligned}&\sharp_{Mr}(p, q, N, C, \_, \mathcal{F}_{in}, \_) \leq \\ &\qquad \sharp_{Ml}(p, q, M, B, \_, \mathcal{F}_{in}, \_) \leq n\end{aligned}$$

Thus from implication (4) we have for some D and $\mathcal{F}_{out}$ that

$$matchr(p, q, N, C, \overline{D}, \mathcal{F}_1, \overline{\mathcal{F}_{out}}) \tag{4.7}$$

holds for some D and $\mathcal{F}_{out}$. Thus, by using rule C_1 on 4.6 and 4.7 we have that

$$closure(p, q, B, \overline{D}, \mathcal{F}_{in}, \overline{\mathcal{F}_{out}})$$

holds.

(1): *Follows from (2) as*

$$\sharp_{Cl}(p, q, \{(p, q)\}, \_, \emptyset, \_) \leq \sharp_{Bis}(p, q, \_, \_) \leq n$$

so for some C and $\mathcal{F}_{out}$

$$closure(p, q, \{(p, q)\}, \overline{C}, \emptyset, \overline{\mathcal{F}_{out}})$$

holds and thus

$$bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}})$$

can be established by using rule B.

Thus we have completed the proof for $n = k + 1$ and thus established that implications (1)–(4) are valid for all n which completes the proof. □

As the above theorem states the inference system will be able to either find a bisimulation C of p and q in case they are equivalent, or find a reason why they are inequivalent, $\mathcal{F}_{out} \models p \not\sim q$, if p and q have finite state-transition diagrams. However, as each bisimulation C containing the pair (p, q) also must contain a pair for each derivative of p (q), the inference system cannot be complete if p and/or q have infinitely many derivations ($\| DER(p) \|$ is infinite). By the definition of *matchl* (*matchr*) we see, that each of p's (q's) derivations must have finite branching in order for the inference system to be complete as M (N) is at most reduced by one element in each call.

We are now ready to state the following completeness result.

Corollary 4.1.8 ***[Completeness]***
For all processes p and q having finite state-transition diagrams we have

$$\exists C \exists \mathcal{F}_{out} : bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}})$$

PROOF: *Follows immediately from theorem 4.1.7.* $\square$

Moreover, from the soundness we know that

$$bisim(p, q, \overline{C}, \overline{\mathcal{F}_{out}}) \Rightarrow$$

$$\begin{array}{lcl} p \sim q & \Rightarrow & C \text{ is a bisimulation} \\ p \not\sim q & \Rightarrow & \mathcal{F}_{out} \models p \not\sim q \end{array}$$

We have now established proofs of soundness and restricted completeness of the inference system of figure 4.1. In the next section we will present the PROLOG-implementation of the system and demonstrate its usefulness through several examples.

Chapter 5

The PROLOG implementation

In this section we will present the PROLOG implementation of the improved inference system, and demonstrate its usefulness through some examples. We will discuss the necessary changes to the system in order for it to find weak bisimulations and demonstrate this (much more interesting) system by further examples.

5.1 CCS in PROLOG

It is (surprisingly) simple to represent CCS and its operational semantics in PROLOG. For each process construction a corresponding PROLOG-operator is introduced. Of cause, it is not possible to use the standard notation (e.g. as prefixing; '.' has a special meaning in PROLOG) and the notation used i shown in figure 5.1. The `a` in prefixing can be any action. Complementary actions, used in connection with parallel composition, are implemented by the two predicates `in` and `out`. The set Act_{PROLOG} is thus the set of (lower case) names

```
        atom   |   in(atom)   |   out(atom)   |   tau
```

Construction	CCS-notation	PROLOG-notation
inaction	NIL	`nil`
prefixing	$a.p$	`a;p`
summation	$p + q$	`p + q`
parallel	$p \mid q$	`p / q`
restriction	$p \setminus (Act \setminus S)$	`p\S`
renaming	$p[a/b]$	`p-[a:=b]`

Figure 5.1: Standard CSS-notation versus PROLOG-notation.

with `atom` $\in (Act \setminus \tau)$, where `tau` is the special action, τ, representing an unobservational action. Note, that for convenience *restriction to* is used in the PROLOG implementation rather than *restriction from* in standard CCS manner. Sets and operations on sets are implemented as lists and operations on lists. The members of `S` in restriction and the actions in renaming may only be atoms (i.e. `in()`/`out()`-forms and `tau` are not allowed).

The assignment operator, `:=:`, is also available assigning process expressions (behaviours) to process identifiers allowing the definition of (mutually) recursive processes.

Parentheses are used to override the introduced operator precedence:

$$prefix > restriction > renaming > summation > parallel$$

In order to complete the implementation of the operational semantics, the derivation relation $\longrightarrow$ has to be implemented. $\longrightarrow$ is in PROLOG represented as the predicate `der` with an almost one-to-one correspondence between the inference rules for $\longrightarrow$ (see figure 2.2) and the PROLOG clauses `der` shown in figure 5.2.

5.2 Using the system

By combining the representation of CCS in PROLOG with the PROLOG implementation of the new inference system (of figure 4.1) and some auxiliary predicates we obtain a new system for proving or disproving bisimulation equivalence between CCS-processes. As with $\longrightarrow$, the inference system can readily be represented in PROLOG wit an (almost) one-to-one correspondence between the inference rules and the corresponding PROLOG

```
der(A;P,A,P).
der(P + Q,A,R) :- der(P,A,R).
der(P + Q,A,R) :- der(Q,A,R).
der(P / Q,A,R / Q) :- der(P,A,R).
der(P / Q,A,P / S) :- der(Q,A,S).
der(P / Q,tau,R / S) :- der(P,in(A),R), der(Q,out(A),S).
der(P / Q,tau,R / S) :- der(P,out(A),R), der(Q,in(A),S).
der(P\L,in(A),Q\L) :- in(A,L), der(P,in(A),Q).
der(P\L,out(A),Q\L) :- in(A,L), der(P,out(A),Q).
der(P\L,A,Q\L) :- in(A,L), der(P,A,Q).
der(P\L,tau,Q\L) :- der(P,tau,Q).
der(P-[A:=B],in(B),Q-[A:=B]) :- der(P,in(A),Q).
der(P-[A:=B],out(B),Q-[A:=B]) :- der(P,out(A),Q).
der(P-[A:=B],B,Q-[A:=B]) :- der(P,A,Q).
der(P-[A:=B],in(C),Q-[A:=B]) :- der(P,in(C),Q), A\= C.
der(P-[A:=B],out(C),Q-[A:=B]) :- der(P,out(C),Q), A\= C.
der(P-[A:=B],C,Q-[A:=B]) :- der(P,C,Q), atom(C), A\= C.
```

Figure 5.2: Standard CSS' operational semantics in PROLOG.

clauses (see figure 5.3 or appendix B for the full PROLOG implementation). Note, however, in PROLOG the side-conditions are included as a part of the premise.

We will demonstrate the usefulness of the system through a couple of examples. First, consider the processes of example 2.2.7. They can be represented in PROLOG by

```
p1 :=: a; b; nil + a; nil.
q1 :=: a; b; nil.
```

In example 2.2.7 we found that $p1$ and $q1$ have different deadlock properties and they can thus not be bisimulation equivalent. Lets us verify this by using the system:

```
bisim(p1,q1).

Searching for a strong bisimulation...
The process
     p1
enjoys some properties which the process
     q1
```

```
bisim(P,Q) :-
   nl, wstring("Searching for a strong bisimulation..."), nl,
   closure(P,Q,[[P,Q]],C,[],Fout), prettyprint(P,Q,C,Fout).

closure(P,Q,B,D,Fin,Fout) :-
   derset(P,M), matchl(P,Q,M,B,C,Fin,F1), not(in([P,Q,_],F1)), !,
   derset(Q,N), matchr(P,Q,N,C,D,F1,Fout).
closure(P,Q,B,D,Fin,Fout) :-
   derset(P,M), matchl(P,Q,M,B,D,Fin,Fout).

matchl(P,Q,[],B,B,Fin,Fin).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1), in([P1,Q1],B), !, matchl(P,Q,M,B,D,Fin,Fout).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   in([P1,Q1,_],F1), !,
   matchl(P,Q,[[A,P1]|M],B,D,F1,Fout).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   not(in([P1,Q1,_],F1)),
   matchl(P,Q,M,C,D,F1,Fout).
matchl(P,Q,[[A,P1]|M],B,B,Fin,[[P,Q,A&G]|Fin]) :-
   not(in([P,Q,_],Fin)), !,
   findall(F1, ( der(Q,A,Q1), in([P1,Q1,F1],Fin) ), LF),
   reduce(LF,RLF), conjunctlist(RLF,G).
matchl(P,Q,[[A,P1]|M],B,B,Fin,Fin).

matchr(P,Q,[],B,B,Fin,Fin).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1), in([P1,Q1],B), !, matchr(P,Q,N,B,D,Fin,Fout).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   in([P1,Q1,_],F1), !,
   matchr(P,Q,[[A,Q1]|N],B,D,F1,Fout).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   not(in([P1,Q1,_],F1)),
   matchr(P,Q,N,C,D,F1,Fout).
matchr(P,Q,[[A,Q1]|N],B,B,Fin,[[P,Q,A;G]|Fin]) :-
   not(in([P,Q,_],Fin)), !,
   findall(F1, ( der(P,A,P1), in([P1,Q1,F1],Fin) ), LF),
   reduce(LF,RLF), disjunctlist(RLF,G).
matchr(P,Q,[[A,Q1]|M],B,B,Fin,Fin).
```

Figure 5.3: PROLOG representation of the extended inference system.

```
don't. One property is:
     <a>[b]ff


yes
```

As we can see, p_1, can reach a `b`-deadlocked state after an `a`-experiment which q_1 cannot. If we, however, prefer to have a property that q_1 enjoys which p_1 does not enjoy, then the order of the parameters of `bisim` should just be reversed i.e

```
bisim(q1,p1).

Searching for a strong bisimulation...
The process
     q1
enjoys some properties which the process
     p1
don't. One property is:
     [a]<b>tt


yes
```

and we see that no matter how an `a`-experiment is performed on q_1 it is always possible to perform a `b`-experiment afterwards, which truly distinguishes the two processes.

Consider now an automatic vending machine offering hot drinks e.g. coffee or tea. Such a machine could for instance be represented in PROLOG by

```
vendonce_machine1 :=: coin;(coffee; nil + tea; nil).
```

A vending machine of a different brand could in PROLOG be

```
vendonce_machine2 :=: coin; coffee; nil + coin; tea; nil.
```

(This example corresponds to examples 3.1.3 and 3.1.4). Now, verifying the inequivalence of the two processes using the system results in the following modal argument:

```
bisim(vendonce_machine1,vendonce_machine2).

Searching for a strong bisimulation...
```

```
The process
     vendonce_machine1
enjoys some properties which the process
     vendonce_machine2
don’t. One property is:
     <coin>(<tea>tt and <coffee>tt)


yes
```

As we can see, the two processes are inequivalent and `vendonce_machine1` enjoys the property that after a coin there is a free choice of either coffee or tea, which `vendonce_machine2` does not enjoy. `vendonce_machine2`, of cause, enjoys a property `vendonce_machine1` does not enjoy which can be found by

```
bisim(vendonce_machine2,vendonce_machine1).

Searching for a strong bisimulation...
The process
     vendonce_machine2
enjoys some properties which the process
     vendonce_machine1
don’t. One property is:
     <coin>[tea]ff


yes
```

Again, it is obvious that the two processes are inequivalent; the reason being that, `vendonce_machine2` can after a coin has been inserted reach a state where it is impossible to buy tea (`tea`-deadlocked). This example compared to example 3.1.4 indicates that this system produces modal properties which are significantly ‘smaller’ than the ones produced by [Vestmar,Olesen]’s system.

The two examples presented so far have been rather trivial and serves mainly as easy-to-understand-introductory-examples. However, most interesting examples are the ones best studied by using weak bisimulation.

```
obder(P,tau,P).
obder(P,A,Q) :-
   der(P,tau,R), P\==R, obder(R,A,Q).
obder(P,A,Q) :-
   der(P,A,R), A\==tau, obder(R,tau,Q).
```

Figure 5.4: Observational derivation in PROLOG.

5.2.1 Weak bisimulation in PROLOG

By proposition 2.2.23 it is possible to change the system in order to find weak bisimulations by exchanging the `der`-predicate in `matchl` and `matchr` clauses with a PROLOG implementation of $\Longrightarrow$, `obder`, possibly represented as in figure 5.4. We use M_O as our modal language using the following simplification equations (follows from the definition of $\Longrightarrow$):

$$\begin{array}{rcl} \langle\!\langle\epsilon\rangle\!\rangle\langle\!\langle\alpha\rangle\!\rangle F & \equiv & \langle\!\langle\alpha\rangle\!\rangle F \\ \langle\!\langle\alpha\rangle\!\rangle\langle\!\langle\epsilon\rangle\!\rangle F & \equiv & \langle\!\langle\alpha\rangle\!\rangle F \\ \\ [\![\epsilon]\!][\![\alpha]\!]F & \equiv & [\![\alpha]\!]F \\ [\![\alpha]\!][\![\epsilon]\!]F & \equiv & [\![\alpha]\!]F \end{array}$$

Now, by using a system for finding weak bisimulations similar to the previous system we are able to handle somewhat more interesting examples.

Again, consider the vending machines from above. Normally, though, a vending machine is constructed in order to serve several customers. Cyclic versions of the above vending machines can be defined by

```
good_machine :=: in(coin); (out(coffee); good_machine +
                             out(tea); good_machine).
bad_machine :=: in(coin); out(coffee); bad_machine +
                in(coin); out(tea); bad_machine.
```

By placing the machines in a university environment we can measure the usefulness of the respective machines by installing them in departments with coffee-drinking computer-scientists and see if they are publishing enough papers (as everybody know, computer-scientists cannot work, e.g. make publications, without having coffee to drink). Also, computer-scientists are widely known to be somewhat absent-minded, so the vending machine will be placed in a separate room in order to let the scientists be undisturbed when vending their coffee. We thus define a (semaphore) `door` as

```
door :=: in(close); in(open); door.
```

and the computer-scientist as

```
computer_scientist :=: out(close); out(coin); in(coffee);
                       out(open); pub; computer_scientist.
```

Then we can define the two departments and an ideal department as

```
good_department :=: (computer_scientist/door/good_machine)\[pub].
bad_department :=: (computer_scientist/door/bad_machine)\[pub].
ideal_department:=: pub; ideal_department.
```

concentrating on whether or not publications are continuously produced as by the `ideal_department`.

```
bisim(good_department,ideal_department).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     good_department     and     ideal_department


good_department
               ideal_department

(out(coin);in(coffee);out(open);pub;computer_scientist/
 in(open);door/good_machine)\[pub]
               ideal_department

(in(coffee);out(open);pub;computer_scientist/in(open);door/
 out(coffee);good_machine+out(tea);good_machine)\[pub]
               ideal_department

(out(open);pub;computer_scientist/in(open);door/good_machine)\[pub]
               ideal_department

(pub;computer_scientist/door/good_machine)\[pub]
               ideal_department

(computer_scientist/door/good_machine)\[pub]
               ideal_department


yes
```

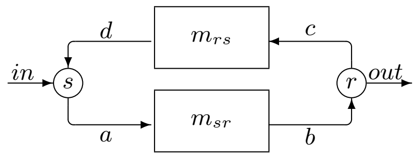


Figure 5.5: Alternating Bit Protocol scenario.

As we can see, the system found a bisimulation of the `good_department` and `ideal_department` consisting of the pairs listed. Each pair is listed with the lefthand argument (e.g. `good_department`) appearing on the line above the righthand argument (e.g. `ideal_department`) which is output on the immediately following indented line. For `bad_department` we have

```
bisim(bad_department,ideal_department).

Searching for a weak bisimulation...
The process
     bad_department
enjoys some properties which the process
     ideal_department
don't. One property is:
     <<epsilon>>[[pub]]ff


yes
```

that it can be `pub`-deadlocked, which certainly is not desirable. Thus, if we want the department to continuously produce publications a vending machine of a type similar to `good_machine` should be chosen.

Let us now turn to a different type of problems arising in distributed systems, namely networking. Especially we consider some simple versions of the Alternating Bit Protocol [Tanenbaum]. We have a scenario like the one in figure 5.5. s is the sender getting the input '*in*' sending a message 'a' through the medium m_{sr}, which in turn sends a message 'b' to the receiver, r. r outputs '*out*' and gives an acknowledge 'c' through the medium, m_{rs}, returning a 'd' to s. The protocol must fulfil the specification *spec* given by

```
spec :=: in; out; spec.
```

Now, let us pretend that the media, m_{sr} and m_{rs}, both are perfect i.e. they do not loose or corrupt any messages.

```
m_sr0 :=: in(a); out(b); m_sr0.
m_rs0 :=: in(c); out(d); m_rs0.
```

Then the sender and receiver are given by

```
s0 :=: in; out(a); in(d); s0.
r0 :=: in(b); out; out(c); r0.
```

and as we would expect the composite system

```
sys :=: (s0/m_sr0/r0/m_rs0)\[in,out].
```

is observation equivalent with the specification `spec` proved by our system

```
bisim(sys,spec).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     sys     and     spec


sys
               spec

(out(a);in(d);s0/m_sr0/r0/m_rs0)\[in,out]
               out;spec

(in(d);s0/out(b);m_sr0/r0/m_rs0)\[in,out]
               out;spec

(in(d);s0/m_sr0/out;out(c);r0/m_rs0)\[in,out]
               out;spec

(in(d);s0/m_sr0/out(c);r0/m_rs0)\[in,out]
               spec

(in(d);s0/m_sr0/r0/out(d);m_rs0)\[in,out]
               spec

(s0/m_sr0/r0/m_rs0)\[in,out]
               spec
```

```
yes
```

A more realistic system would not expect all of its components to be perfect. Let us allow the media to loose messages i.e.

```
m_sr1 :=: in(a); (out(b); m_sr1 + tau; m_sr1).
m_rs1 :=: in(c); (out(d); m_rs1 + tau; m_rs1).
```

but let `s` and `r` still believe that they have perfect media. Naturally, this new system

```
sys1 :=: (s0/m_sr1/r0/m_rs1)\[in,out].
```

does not fulfil the specification the reason being that

```
bisim(sys1,spec).

Searching for a weak bisimulation...
The process
     sys1
enjoys some properties which the process
     spec
don't. One property is:
     <<in>>[[out]]ff


yes
```

i.e. `sys1` can become `out`-deadlocked after having performed an `in`-experiment. Obviously, the sender and the receiver must be able to repeat their messages until they receive a (new) message i.e.

$$s :=: d^*.in.\overline{a}^*.d.s$$
$$r :=: \overline{c}^*.b.b^*.out.r$$

where $a^*.P$ is given by $X :=: a.X + P$ where X is any process identifier not occurring in P. Will this system fulfil the specification? The new sender and receiver are thus given by

```
aux_s1 :=: out(a); aux_s1 + in(d); s1.
s1 :=: in(d); s1 + in; aux_s1.
aux_r1 :=: in(b); aux_r1 + out; r1.
r1 :=: out(c); r1 + in(b); aux_r1.
```

and the new system is

```
sys2 :=: (s1/m_sr1/r1/m_rs1)\[in,out].
```

This system can now be tested against the specification

```
bisim(sys2,spec).

Searching for a weak bisimulation...
The process
     sys2
enjoys some properties which the process
     spec
don't. One property is:
     <<in>><<out>><<out>>tt


yes
```

and we see that `sys2` is not equivalent to `spec`! What went wrong? Well, out system claims that it is possible to perform two sequential `out`-experiments following an `in`-experiment on `sys2` which is not the expected bahaviour. But how can that happen? Of cause, the sender keeps on sending although the receiver already has received the message once, and the receiver ‘thinks’ that each messae received after the first is a new one, thus it outputs `out` each time. This problem is actually solved by the Alternating Bit Protocol (please refer to [Tanenbaum]).

The last example we will consider origins from some lecture notes by Robin Milner [Larsen] and involves the representation of a workshop comprising two men, a hammer, and a mallet. We will use it as a stepping stone for introducing the last improvement to the system. A man can either use a hammer or a mallet to perform a job. `gh` and `ph` represents getting and putting the hammer, likewise `gm` and `pm` for the mallet. The behaviour of a `man` is given by

```
man :=: injob;(in(gh); out(ph); outjob; man +
               in(gm); out(pm); outjob; man).
```

The hammer and he mallet are given by

```
hammer :=: out(gh); in(ph); hammer.
mallet :=: out(gm); in(pm); mallet.
```

Now, the two men and their tools comprises the `closedshop`

```
closedshop :=: (man/man/hammer/mallet)\[injob,outjob].
```

The specification for `closedshop` is given by the process `donothing`

```
one :=: injob; outjob; one + outjob; injob; one.
donothing :=: injob; one.
```

The following shows that `closedshop` is observational equivalent with `donothing`

```
bisim(closedshop,donothing).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     closedshop     and     donothing



closedshop
               donothing

(man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 hammer/mallet)\[injob,outjob]
               one

(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/in(gh);out(ph);outjob;man+
 in(gm);out(pm);outjob;man/hammer/mallet)\[injob,outjob]
               outjob;one

(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 out(ph);outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               outjob;one

(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 outjob;man/hammer/mallet)\[injob,outjob]
               outjob;one

(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 man/hammer/mallet)\[injob,outjob]
               one

(out(ph);outjob;man/man/in(ph);hammer/mallet)\[injob,outjob]
               one

(out(ph);outjob;man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 in(ph);hammer/mallet)\[injob,outjob]
               outjob;one

(outjob;man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 hammer/mallet)\[injob,outjob]
               outjob;one

(outjob;man/out(ph);outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               outjob;one

(man/out(ph);outjob;man/in(ph);hammer/mallet)\[injob,outjob]
```

```
               one

(man/outjob;man/hammer/mallet)\[injob,outjob]
               one

(man/man/hammer/mallet)\[injob,outjob]
               injob;one

(outjob;man/outjob;man/hammer/mallet)\[injob,outjob]
               outjob;one

(outjob;man/man/hammer/mallet)\[injob,outjob]
               one

(outjob;man/out(pm);outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               outjob;one

(man/out(pm);outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               one

(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 out(pm);outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               outjob;one

(out(ph);outjob;man/out(pm);outjob;man/in(ph);hammer/
 in(pm);mallet)\[injob,outjob]
               outjob;one

(out(ph);outjob;man/outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               outjob;one

(out(pm);outjob;man/man/hammer/in(pm);mallet)\[injob,outjob]
               one

(out(pm);outjob;man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 hammer/in(pm);mallet)\[injob,outjob]
               outjob;one

(out(pm);outjob;man/out(ph);outjob;man/in(ph);hammer/
 in(pm);mallet)\[injob,outjob]
               outjob;one

(out(pm);outjob;man/outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               outjob;one


yes
```

The (vast) number of pairs in the bisimulation indicates that it is necessary to make further improvements to the system, as fitting the bisimulation pairs together in order to obtain a proof will most certainly become an extremly teadious job even for moderate examples like the `closedshop`. It is like having a jigsaw puzzle with 3000 pieces without a pricture to go with it—it is possible to fit the pieces together but hardly worth the trouble. We need a way of relating the bisimulation pairs. This will be dealt with in the following chapter.

Chapter 6

A new extended system

—arguing for extended bisimulation equivalence and inequivalence

We will in this section present the notion of *extended bisimulation* as a faithful extension of ordinary bisimulation. Furthermore, we will construct and implement a new system arguing for extended bisimulation equivalence and inequivalence. The usefulness of this system is demonstrated by reconsidering the `closedshop`-example (and some further examples in appendix D).

6.1 Extended bisimulation

As indicated by the `closedshop`-example in section 5.2.1, some way of relating the bisimulation pairs is required. By the definition of bisimulation (see definition 2.2.8), we see that each pair, (p', q'), is included in the bisimulation, $\mathcal{R}$, as a result of matching another pairs's, (p, q), derivations. It will therefore be natural to use this 'matching information' as a means of relating the pairs of the bisimulation.

The suggested solution of the above problem is actually inspired by a notational convenience used in [Prasad] and by a report of L. Hallnäs [Halläs]. The idea is essentially to extend the bisimulation pairs to quadruples. Intuitively, for each pair, $(p, q) \in \mathcal{R}$, we want information relating to p's derivations to q's and vice versa. Now, before defining the notion of *extended* *bisimulation* we will state some notational conveniences.

Notation 6.1.1 *Let* $\mathbf{Q} = Pr \times Pr \times 2^{((Act \times Pr) \times Pr)} \times 2^{((Act \times Pr) \times Pr)}$. *For each* $\mathcal{R} \subseteq \mathbf{Q}$ *we define it's projected image* $\mathcal{R}^{\downarrow}$ *by*

$$\begin{aligned}(p, q) \in \mathcal{R}^{\downarrow} \;\Leftrightarrow& \\ &\exists F, G : (p, q, F, G) \in \mathcal{R}\end{aligned}$$

□

Notation 6.1.2 *For each* $F \subseteq ((Act \times Pr) \times Pr)$ *we let* $DOM(F)$ *denote the set*

$$\{(a, p') \mid \exists q' : (\,(a, p'), q') \in F\}$$

□

Notation 6.1.3 *For each process* $p \in Pr$ *let*

$$DER(p) = \{(a, p') \mid p \xrightarrow{a} p'\}$$

□

Now, an *extended bisimulation* is given by the following definition.

Definition 6.1.4 ***[Extended bisimulation]***
A set $\mathcal{R} \subseteq \mathbf{Q}$ *is called an* extended bisimulation *iff*

$$\begin{array}{ll}(p, q, F, G) \in \mathcal{R} \;\Rightarrow & \\ \quad 1) & DER(p) \subseteq DOM(F) \\ \quad 2) & DER(q) \subseteq DOM(G) \\ \quad 3) & \begin{array}{l} p \xrightarrow{a} p' \Rightarrow \\ \qquad \forall (\,(a, p'), q') \in F : q \xrightarrow{a} q' \;\wedge\; (p', q') \in \mathcal{R}^{\downarrow} \end{array} \\ \quad 4) & \begin{array}{l} q \xrightarrow{a} q' \Rightarrow \\ \qquad \forall (\,(a, q'), p') \in G : p \xrightarrow{a} p' \;\wedge\; (p', q') \in \mathcal{R}^{\downarrow} \end{array}\end{array}$$

□

As for ordinary bisimulations we are able to define an ordering on extended bisimulations.

Definition 6.1.5 *$\sqsubseteq \subseteq \mathbf{Q} \times \mathbf{Q}$ is the relation on $\mathbf{Q}$ such that for $\mathcal{R}, \mathcal{S} \in \mathbf{Q}$*

$$\begin{aligned}&\mathcal{R} \sqsubseteq \mathcal{S} \;\Leftrightarrow\\ &\quad (p,q,F,G) \in \mathcal{R} \;\Rightarrow\; \exists F', G' : (p,q,F',G') \in \mathcal{S} \;\wedge\; F \subseteq F' \;\wedge\; G \subseteq G'\end{aligned}$$

□

Corollary 6.1.6 *For all $\mathcal{R}, \mathcal{S} \in \mathbf{Q}$ we have that*

$$\mathcal{R} \sqsubseteq \mathcal{S} \;\Rightarrow \mathcal{R}^{\downarrow} \subseteq \mathcal{S}^{\downarrow}$$

□

Proposition 6.1.7 *$(\mathbf{Q}, \sqsubseteq)$ is a partial ordering.*

Proof: *We must show that $\sqsubseteq$ is a reflexive and anti-symmetrical ordering on $\mathbf{Q}$, which follows straight forward from its definition.* □

Furthermore, $\sqsubseteq$ makes $\mathbf{Q}$ into a complete lattice.

Proposition 6.1.8 *$(\mathbf{Q}, \sqsubseteq)$ is a complete lattice.*

Proof: *Let $\{\mathcal{R}_i \mid i \in I\}$ be a family of relations from $\mathbf{Q}$. Obviously, $\bigcup_i \mathcal{R}_i$ is an upper bound for any $\mathcal{R}_i$. Now, assume $\mathcal{R}_i \sqsubseteq U$ for all $i \in I$ for some $U \in \mathbf{Q}$. We must show $\bigcup_i \mathcal{R}_i \sqsubseteq U$. However, this follows trivially since $(p,q,F,G) \in \mathcal{R}_i$ for some $i \in I$ whenever $(p,q,F,G) \in \bigcup_i \mathcal{R}_i$* □

Now, for $\mathcal{R} \subseteq \mathbf{Q}$ we define $\mathcal{B}_E \subseteq \mathbf{Q}$ as

$$\mathcal{B}_E(\mathcal{R}) = \{(p,q,F,G) \mid \text{(1)–(4) of definition 6.1.4 holds}\} \tag{6.1}$$

Then $\mathcal{B}_E$ has the following properties.

Proposition 6.1.9 *$\mathcal{B}_E$ is a monotonic endofunction on the complete lattice $(\mathbf{Q}, \sqsubseteq)$.*

Proof: *Follows easily from corollary 6.1.6 and from the definition of $\mathcal{B}_E$ (equation 6.1) as*

$$\begin{aligned}\mathcal{R} \sqsubseteq \mathcal{S} \;\Rightarrow\; \mathcal{R}^{\downarrow} \subseteq \mathcal{S}^{\downarrow} \;\Rightarrow\\ \mathcal{B}_E(\mathcal{R}) \subseteq \mathcal{B}_E(\mathcal{S})\end{aligned}$$

by the definition. □

Using the standard fixed-point result, due to Tarski, this implies

Proposition 6.1.10 *$\mathcal{B}_E$ has a maximal fixed-point $\sim_E$ given by*

$$\sim_E = \bigcup \{\mathcal{R} \mid \mathcal{R} \sqsubseteq \mathcal{B}_E(\mathcal{R})\}$$

□

To see that this notion is a faithful extension of the standard notion of bisimulation, we state the following lemma.

Lemma 6.1.11

1. *$\mathcal{R}$ is an extended bisimulation $\Rightarrow$*
   $\mathcal{R}^{\downarrow}$ is a bisimulation
2. *$\mathcal{S}$ is an extended bisimulation $\Rightarrow$*
   $\exists \mathcal{R} : \mathcal{R}$ is an extended bisimulation $\wedge$ $\mathcal{R}^{\downarrow} = \mathcal{S}$

PROOF: *(1) follows readily from the definition of extended bisimulation and $^{\downarrow}$. To see (2), assume $\mathcal{S}$ is a bisimulation. Then a corresponding extended bisimulation, $\mathcal{R}$, can be constructed by for each pair $(p, q) \in \mathcal{S}$ adding the quadruple (p, q, F, G) to $\mathcal{R}$ where F and G are constructed such that*

1. $(a, p') \in DER(p) \ \wedge \ (p', q') \in S \ \Rightarrow \ (\,(a, p'), q') \in F$
2. $(a, q') \in DER(q) \ \wedge \ (p', q') \in S \ \Rightarrow \ (\,(a, q'), p') \in G$

Clearly, $\mathcal{R}$ is an extended bisimulation and $\mathcal{R}^{\downarrow} = \mathcal{S}$. □

From this lemma it follows as an easy corollary that $(\sim_E)^{\downarrow}$.

In the following section we will use the above results by briefly discussing the construction and implementation of a system similar to the previously constructed system (see figure 4.1) arguing for extended bisimulation equivalence and inequivalence.

6.2 Construction and implementation

In order to construct F and G ‘incrementally’ while ‘running through’ the inference system some auxiliary functions are needed for convenience. Let

$$\left.\begin{array}{l} add_{to}F \\ add_{to}G \end{array}\right\} : Pr \times Pr \times ((Act \times Pr) \times Pr) \times 2^Q \mapsto 2^Q$$

be defined as

$$\begin{array}{l} add_{to}F(p, q, (\,(a, p'), q'), B) = \\ \qquad \{(p, q, \{(\,(a, p'), q')\} \cup F, G)\} \cup (B \setminus \{(p, q, F, G)\}) \\ add_{to}G(p, q, (\,(a, q'), p'), B) = \\ \qquad \{(p, q, F, \{(\,(a, q'), p')\} \cup G)\} \cup (B \setminus \{(p, q, F, G)\}) \end{array}$$

where $(p, q, F, G) \in B$ for some F and G.

Now, by making appropriate changes to the verification conditions of figure 4.2, we obtain the verification conditions in figure 6.1. Then the soundness and restricted completeness of the new inference system in figure 6.2 and 6.3 follows by using argument similar to the arguments used proving the inference system of figure 4.1.

We are thus for each bisimulation pair $(p, q) \in \mathcal{R}$ able to printout exactly which pair $(p', q') \in \mathcal{R}$ of processes a specific action will result in i.e. how the derivations are matched. We want this information to be displayed along with the bisimulation pairs. However, in order to obtain a good layout it is not the matching process pairs (p', q') we want to display but rather a (unique) number corresponding to the bisimulation pair (p', q') in order to minimize the amount of redundant information displayed.

The easiest way to implement this seems to be by performing a post-processing on the extended bisimulation relation found. This processing can be made in two stages:

1. Assign a (unique) number to each quadruple.
2. For each of the processes in each pair of the enumerated extended bisimulation generate a set $\{(a, no), \ldots\}$ where a is the action performed and no is the number of the related quadruple.

$$Bis(p,q,\overline{C},\overline{\mathcal{F}_{out}}) \Leftrightarrow^{\Delta}$$
$$\{\} \Rightarrow$$
$$\left[\begin{array}{l} (\,(p,q)\in\overline{C}^{\downarrow} \;\wedge\; \overline{C}^{\downarrow} \subseteq \mathcal{B}(\overline{C}^{\downarrow})\,)\vee \\ (\overline{\mathcal{F}_{out}} \models p \not\sim q) \;\wedge\; \overline{\mathcal{F}_{out}} \text{ is AI} \end{array}\right]$$

$$Cl(p,q,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}}) \Leftrightarrow^{\Delta}$$
$$\left\{\begin{array}{l} (p,q)\in B^{\downarrow}\wedge \\ \mathcal{F}_{in} \text{ is AI} \end{array}\right\} \Rightarrow$$
$$\left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is AI} \;\wedge\; \mathcal{F}_{in}\subseteq\overline{\mathcal{F}_{out}}\;\wedge & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B\sqsubseteq\overline{C}\;\wedge \\ \overline{C}^{\downarrow}\setminus(B^{\downarrow}\setminus\{(p,q)\})\subseteq\mathcal{B}(\overline{C}^{\downarrow}) \end{array}\right] \end{array}\right]$$

$$Ml(p,q,M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}}) \Leftrightarrow^{\Delta}$$
$$\left\{\begin{array}{l} (p,q)\in B^{\downarrow}\wedge \\ M\subseteq\{(a,p')\mid p\xrightarrow{a}p'\}\wedge \\ (p_M,q)\in\mathcal{L}(B^{\downarrow})\;\wedge\;\mathcal{F}_{in} \text{ is AI} \end{array}\right\} \Rightarrow$$
$$\left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is AI} \;\wedge\; \mathcal{F}_{in}\subseteq\overline{\mathcal{F}_{out}}\;\wedge & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B\sqsubseteq\overline{D}\;\wedge \\ \overline{D}^{\downarrow}\setminus B^{\downarrow}\subseteq\mathcal{B}(\overline{D}^{\downarrow})\;\wedge \\ (p,q)\in\mathcal{L}(\overline{D}^{\downarrow}) \end{array}\right] \end{array}\right]$$

$$Mr(p,q,N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}}) \Leftrightarrow^{\Delta}$$
$$\left\{\begin{array}{l} (p,q)\in B^{\downarrow}\wedge \\ N\subseteq\{(a,q')\mid q\xrightarrow{a}q'\}\wedge \\ (p,q_N)\in\mathcal{R}(B^{\downarrow})\;\wedge\;\mathcal{F}_{in} \text{ is AI} \end{array}\right\} \Rightarrow$$
$$\left[\begin{array}{ll} \overline{\mathcal{F}_{out}} \text{ is AI} \;\wedge\; \mathcal{F}_{in}\subseteq\overline{\mathcal{F}_{out}}\;\wedge & \\ (\overline{\mathcal{F}_{out}} \not\models p \not\sim q) \Rightarrow & \left[\begin{array}{l} B\sqsubseteq\overline{D}\;\wedge \\ \overline{D}^{\downarrow}\setminus B^{\downarrow}\subseteq\mathcal{B}(\overline{D}^{\downarrow})\;\wedge \\ (p,q)\in\mathcal{R}(\overline{D}^{\downarrow}) \end{array}\right] \end{array}\right]$$

where $p_M = \sum\{(a.p' \mid p \xrightarrow{a} p' \;\wedge\; (a,p')\notin M\}$.

Figure 6.1: Verification conditions for extended bisimulations

$$B \;:\quad \frac{closure(p,q,\{(p,q,\emptyset,\emptyset)\},\overline{C},\emptyset,\overline{\mathcal{F}_{out}})}{bisim(p,q,\overline{C},\overline{\mathcal{F}_{out}})}$$

$$C \;:\quad \frac{matchl(p,q,M,B,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchr(p,q,N,C,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{closure(p,q,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{\begin{array}{l} \mathcal{F}_1 \not\models p \not\sim q \;\wedge \\ M = \{(a,p') \mid p \xrightarrow{a} p'\} \;\wedge \\ N = \{(a,q') \mid q \xrightarrow{a} q'\} \end{array}\right.$$

$$\frac{matchl(p,q,M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}{closure(p,q,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{\begin{array}{l} \mathcal{F}_{out} \models p \not\sim q \;\wedge \\ M = \{(a,p') \mid p \xrightarrow{a} p'\} \end{array}\right.$$

$$ML \;:\quad \frac{}{matchl(p,q,\emptyset,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})}$$

$$\frac{matchl(p,q,M,B_1,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{\begin{array}{l} \exists q' : q \xrightarrow{a} q' \;\wedge \\ (p',q',F',G') \in B \;\wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \;\wedge \\ B_1 = add_{to}F(p,q,(\,(a,p'),q'),B) \end{array}\right.$$

$$\frac{closure(p',q',\{(p',q',\emptyset,\emptyset)\} \cup B_1,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{\begin{array}{l} q \xrightarrow{a} q' \;\wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \;\wedge \\ \mathcal{F}_{in} \not\models p' \not\sim q' \;\wedge \\ \mathcal{F}_1 \models p' \not\sim q' \;\wedge \\ B_1 = add_{to}F(p,q,(\,(a,p'),q'),B) \end{array}\right.$$

$$\frac{closure(p',q',\{(p',q',\emptyset,\emptyset)\} \cup B_1,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1}) \quad matchl(p,q,M,C,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})} \quad \left\{\begin{array}{l} q \xrightarrow{a} q' \;\wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \;\wedge \\ \mathcal{F}_{in} \not\models p' \not\sim q' \;\wedge \\ \mathcal{F}_1 \not\models p' \not\sim q' \;\wedge \\ B_1 = add_{to}F(p,q,(\,(a,p'),q'),B) \end{array}\right.$$

$$\frac{}{matchl(p,q,\{(a,p')\} \cup M,B,\overline{B},\mathcal{F}_{in},\{(p,q,\langle a\rangle F')\} \cup \mathcal{F}_{in})} \quad \left\{\begin{array}{l} \mathcal{F}_{in} \not\models p \not\sim q \;\wedge \\ F' = \bigwedge\{F_1,\ldots,F_n\} \;\wedge \\ \text{where } \{q'_1,\ldots,q'_n\} = \{q' \mid q \xrightarrow{a} q'\} \\ \text{and } \forall i : (p',q'_i,F_i) \in \mathcal{F}_{in} \end{array}\right.$$

$$\frac{}{matchl(p,q,M,B,\overline{B},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})} \quad \left\{\; \mathcal{F}_{in} \models p \not\sim q \right.$$

Figure 6.2: Inference rules for extended bisimulations

$$MR\ :\quad \frac{}{matchr(p,q,\emptyset,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})}$$

$$\frac{matchr(p,q,N,B_1,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}{matchr(p,q,\{(a,q')\}\cup N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}\quad\left\{\begin{array}{l}\exists q' : q \xrightarrow{a} q' \wedge \\ (p',q',F',G') \in B \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \wedge \\ B_1 = add_{to}G(p,q,(\,(a,q'),p'),B)\end{array}\right.$$

$$\frac{closure(p',q',\{(p',q',\emptyset,\emptyset)\}\cup B_1,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1})\quad matchr(p,q,\{(a,q')\}\cup N,B,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchr(p,q,\{(a,q')\}\cup N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}\quad\left\{\begin{array}{l}p \xrightarrow{a} p' \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \wedge \\ \mathcal{F}_{in} \not\models p' \not\sim q' \wedge \\ \mathcal{F}_1 \models p' \not\sim q' \wedge \\ B_1 = add_{to}G(p,q,(\,(a,q'),p'),B)\end{array}\right.$$

$$\frac{closure(p',q',\{(p',q',\emptyset,\emptyset)\}\cup B_1,\overline{C},\mathcal{F}_{in},\overline{\mathcal{F}_1})\quad matchr(p,q,N,C,\overline{D},\mathcal{F}_1,\overline{\mathcal{F}_{out}})}{matchr(p,q,\{(a,q')\}\cup N,B,\overline{D},\mathcal{F}_{in},\overline{\mathcal{F}_{out}})}\quad\left\{\begin{array}{l}p \xrightarrow{a} p' \wedge \\ \mathcal{F}_{in} \not\models p \not\sim q \wedge \\ \mathcal{F}_{in} \not\models p' \not\sim q' \wedge \\ \mathcal{F}_1 \not\models p' \not\sim q' \wedge \\ B_1 = add_{to}G(p,q,(\,(a,q'),p'),B)\end{array}\right.$$

$$\frac{}{matchr(p,q,\{(a,q')\}\cup N,B,\overline{B},\mathcal{F}_{in},\{(p,q,[a]F')\}\cup\mathcal{F}_{in})}\quad\left\{\begin{array}{l}\mathcal{F}_{in} \not\models p \not\sim q \wedge \\ F' = \bigvee\{F_1,\ldots,F_n\} \wedge \\ \text{where } \{p'_1,\ldots,p'_n\} = \{p' \mid p \xrightarrow{a} p'\} \\ \text{and } \forall i : (p'_i,q',F_i) \in \mathcal{F}_{in}\end{array}\right.$$

$$\frac{}{matchr(p,q,N,B,\overline{B},\mathcal{F}_{in},\overline{\mathcal{F}_{in}})}\quad\{\ \mathcal{F}_{in} \models p \not\sim q$$

Figure 6.3: Inference rules for extended bisimulations (cont.)

By using a system implementing the above principle the (extended) bisimulation of the `closedshop` will now be represented as `bisim(closedshop,donothing)`.

```
Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     closedshop     and     donothing

0
closedshop
               donothing
                         LHS-> injob ==> 5
                         LHS-> injob ==> 1
                         RHS-> injob ==> 5
1
(man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 hammer/mallet)\[injob,outjob]
               one
                         LHS-> epsilon ==> 16
                         LHS-> epsilon ==> 10
                         LHS-> injob ==> 2
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 4
2
(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/in(gh);out(ph);outjob;man+
 in(gm);out(pm);outjob;man/hammer/mallet)\[injob,outjob]
```

```
               outjob;one
                         LHS-> epsilon ==> 21
                         LHS-> epsilon ==> 7
                         LHS-> epsilon ==> 17
                         LHS-> epsilon ==> 3
                         RHS-> outjob ==> 11
3
(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 out(ph);outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 22
                         LHS-> epsilon ==> 4
                         RHS-> outjob ==> 11
4
(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 outjob;man/hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 23
                         LHS-> epsilon ==> 19
                         LHS-> outjob ==> 5
                         RHS-> outjob ==> 11
5
(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 man/hammer/mallet)\[injob,outjob]
               one
                         LHS-> epsilon ==> 20
                         LHS-> epsilon ==> 6
                         LHS-> injob ==> 2
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 8
6
(out(ph);outjob;man/man/in(ph);hammer/mallet)\[injob,outjob]
               one
                         LHS-> epsilon ==> 14
                         LHS-> injob ==> 7
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 8
7
(out(ph);outjob;man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 in(ph);hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 18
                         LHS-> epsilon ==> 8
                         RHS-> outjob ==> 11
8
(outjob;man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 15
                         LHS-> epsilon ==> 9
                         LHS-> outjob ==> 1
                         RHS-> outjob ==> 11
```

```
9
(outjob;man/out(ph);outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 13
                         LHS-> outjob ==> 10
                         RHS-> outjob ==> 11
10
(man/out(ph);outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               one
                         LHS-> epsilon ==> 11
                         LHS-> injob ==> 3
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 4
11
(man/outjob;man/hammer/mallet)\[injob,outjob]
               one
                         LHS-> outjob ==> 12
                         LHS-> injob ==> 4
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 4
12
(man/man/hammer/mallet)\[injob,outjob]
               injob;one
                         LHS-> injob ==> 5
                         LHS-> injob ==> 1
                         RHS-> injob ==> 5
13
(outjob;man/outjob;man/hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> outjob ==> 14
                         LHS-> outjob ==> 11
                         RHS-> outjob ==> 11
14
(outjob;man/man/hammer/mallet)\[injob,outjob]
               one
                         LHS-> outjob ==> 12
                         LHS-> injob ==> 8
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 8
15
(outjob;man/out(pm);outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 13
                         LHS-> outjob ==> 16
                         RHS-> outjob ==> 11
16
(man/out(pm);outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               one
                         LHS-> epsilon ==> 11
                         LHS-> injob ==> 17
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 4
```

```
17
(in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 out(pm);outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 18
                         LHS-> epsilon ==> 4
                         RHS-> outjob ==> 11
18
(out(ph);outjob;man/out(pm);outjob;man/in(ph);hammer/
 in(pm);mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 19
                         LHS-> epsilon ==> 15
                         RHS-> outjob ==> 11
19
(out(ph);outjob;man/outjob;man/in(ph);hammer/mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 13
                         LHS-> outjob ==> 6
                         RHS-> outjob ==> 11
20
(out(pm);outjob;man/man/hammer/in(pm);mallet)\[injob,outjob]
               one
                         LHS-> epsilon ==> 14
                         LHS-> injob ==> 21
                         RHS-> outjob ==> 12
                         RHS-> injob ==> 8
21
(out(pm);outjob;man/in(gh);out(ph);outjob;man+in(gm);out(pm);outjob;man/
 hammer/in(pm);mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 22
                         LHS-> epsilon ==> 8
                         RHS-> outjob ==> 11
22
(out(pm);outjob;man/out(ph);outjob;man/in(ph);hammer/
 in(pm);mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 23
                         LHS-> epsilon ==> 9
                         RHS-> outjob ==> 11
23
(out(pm);outjob;man/outjob;man/hammer/in(pm);mallet)\[injob,outjob]
               outjob;one
                         LHS-> epsilon ==> 13
                         LHS-> outjob ==> 20
                         RHS-> outjob ==> 11


yes
```

Note, that `LHS` refers to the left hand side of the bisimulation pair (`closedshop` in this example) and `RHS` the right hand side. For pair `0` then, `closedshop` can perform an `injob`-experiment which is matched by pair number `5` but it can also perform an `injob`-experiment which is matched by pair `1`. `donothing` can only perform one action, `injob`, which is matched by the pair numbered `1`. Similarly, we can study each of the pairs and thereby get a better insight into *how* the bisimulation found actually fits together. Appendix D shows the result of running the previous examples shown to be bisimulation equivalent through this system. Appendix C contains the full listing of the PROLOG-program corresponding to the inference system of figures 6.2 and 6.3.

Chapter 7

Conclusion and future work

The main achievement of this thesis is the construction and (rigorous) verification of an operational based inference system for reasoning about bisimulation equivalence and (more important) inequivalence. The fact that this system not only produces a bisimulation containing a given pair, (p, q), in case $p \sim q$, but also, in case $p \not\sim q$, is capable of giving a modal property (an argument of inequivalence) which only one of the processes enjoys, makes the system very suitable as an automatic tool aiding verification and debugging of systems. The usefulness was demonstrated through some examples. Also, an extension of the system making it possible to find weak bisimulations was discussed and the usefulness of this much more interesting system was demonstrated through several examples. In particular, the `closedshop`-example, indicated that some sort of structuring of the bisimulations produced by the system was needed. The structure imposed gave for each pair in the bisimulation information about how moves of one process was matched by moved by the other. This resulted in the notion of extended bisimulation which was shown to be faithful to the standard notion of bisimulation. Moreover, a system finding extended bisimulations was implemented and the additional information supplied by the system immediately proved to be superior to that produced by other systems when `closedshop` was reconsidered. The extra information given actually makes it easy to get a good insight into *how* the pairs of the bisimulation fits together, and thereby making it a lot more sensible using the system for theorem proving and a lot easier to take advantage of the produced results.

All in all we must conclude that our system actually fulfill the original goal of constructing a system arguing bisimulation equivalence or inequivalence. Furthermore, the system(s) seems to be a very useful tool assisting the verification of systems.

there are at least two related and interesting problems which are not covered by this thesis:

1. Find the time complexity of the algorithm used.
2. Show (if possible) that the properties generated are minimal (in some sense).

Due to the algorithm following very closely the recursive definition of bisimulation and that backtracking may be necessary (when non-deterministic processes are considered) the time complexity is essentially exponential. However, it would be nice to verify this. Also, it would be interesting if the (expected) minimality of properties could be verified, as this would argue that the properties produced by our system in fact are concise and that our properties are 'smaller' that the ones produced by e.g. the system of [Vestmar,Olesen].

Related to this is the question of whether or not the properties supplied by our system are as informative as we would like to think they are. Surely, they are both correct and they concisely states the property distinguishing the two inequivalent processes, but what are the alternatives? Of course, when asking a question like this you cannot expect to receive a final answer. We will certainly not give 'the' answer, but we would like to put forward an alternative way of presenting the results of the system to the user. Imagine a system which differs from our system in that it initially just will tell whether or not the processes under consideration are equivalent. Now, instead of giving the full argument for the answer, the system will try to *convince* the user of the correctness of the answer by playing a 'game' with him. It is a game in the sense that the competitors tires to make 'life' as hard as possible for each other. The rules are easy. There are two situations to consider; in the first situation the system has determined that the processes are equivalent, and in the other it has been determined that the processes are inequivalent. In the first case the user 'moves' first whereas in the latter case the system has the first 'move'. A move actually consists of selecting a derivation which the opponent has to match. Assume that the processes under consideration are $SYS1$ and $SYS2$ then two typical situations could be:

$SYS1 \not\sim SYS2$: The system moves first. It chooses a process to move, $SYS1$ say. Of course, it selects the worst possible move, namely the move which it knows that $SYS2$ cannot match. Then the user may try to find a move of $SYS2$ matching the move by the system. In case $SYS2$ cannot perform the given action, obviously he must acknowledge the answer given. Otherwise, no matter which derivation of $SYS2$ he chooses, he will of course not succeed, and finally—after sufficiently many repetitions of the game—he will have to admit that the system was right in the first place saying that $SYS1 \not\sim SYS2$.

$SYS1 \sim SYS2$: Now, the user chooses a process to move, $SYS2$ say. He will then try to find a move of $SYS2$ which $SYS1$ cannot match and thereby prove that the system was wrong about saying that $SYS1 \sim SYS2$. However, ind the end—after sufficiently many moves of the game—he will have to give in and admit the system was right all along, as the system will be able to match any move he can perform on $SYS2$.

Now, the point is that a system using the above method of communicating with the user will force the user to explore the behaviour of the two processes to an extent he probably never would have done in another situation and thereby he will learn more about the processes themselves and possibly gain a better understanding of why processes behave in the way the do. Also, this method does not even require the user to know anything about the theory of bisimulation, modal properties etc. All he has to know is how to make a move of a (CCS-)process.

Bibliography

[Aczel]	Peter Aczel, "An Introduction to Inductive Definitions", A Handbook of Mathematical Logic, North Holland Company, 1997 3.2, 4.1.3
[Cardelli]	Luca Cardelli, "ML under UNIX", Murray Hill, New Jersey 07974, 1984
[Clocksin,Mellish]	W. F. Clocksin and C. S. Mellish, "Programming in PROLOG", Springer–Verlag, New York, 1981 3.2
[Gordon]	Michael J. C. Gordon, "The Denotational Description of Programming Languages", Springer–Verlag, New York, 1979 1
[Halläs]	Lars Hallnäs, "An Intentional Characterization ogf Bisimulations in CCS", Uppsala University, 1986 6.1
[Hennessy,Milner]	Matthew Hennessy and Robin Milner, "Algebraic Laws for Nondeterminism and Concurrency", Journal of the ACM, vol.32 no.1, 1985, pp.137–161 2.3, 2.3, 2.3.3, 2.3.5
[Larsen]	Kim Guldstrand Larsen, "Context-depentendt Bisimulation between Processes", Ph. D. Thesis, University of Edinburgh, 1986 1, 1, 1, 2.1.3, 2.2, 2.2.5, 3.1, 3.1, 3.2, 3.3, 4, 4.1, 4.1.3, 5.2.1, A
[Milne,Milner]	G. Milne and R. Milner, "Concurrent Processes and their Syntax", Journal of the ACM, vol.26 no.2, 1979 1
[Milner]	Robin Milner, "A Calculus of Communicating Systems", Lecture Notes in Computer Science, nr.92 1, 2.1, 2.1.1, 2.2.15, 2.2.24, 2.2.26, D

[Paulson] Lawrence Paulson, “A Compiler Generator for Semantic Grammars”, a dissertationsubmitted to Department of Computer Science and the Committe on graduate Studies of Standford University in partial fulfilment of the requirements for the degree of Doctor of Philosophy, 1981 1

[Prasad] K. V. S. Prasad, “Specifications and Proof of a Simple Fault Tolerant System in CCS”, Internal Report, University of Edinburgh, 1984 1, 6.1

[Plotkin] Gordon D. Plotkin, “A structural approach to operational semantics”, Aarhus University, DIAMI FN–19, September 1981 2.1.1

[Tanenbaum] A. S. Tanenbaum, “Computer Networks”, Prentice–Hall, Inc., Englewood Cliffs, New York, 1981 5.2.1

[Vestmar,Olesen] Karsten Vestmar and Jørgen Olesen, “Specifikation og Implementering af Fuldautomatisk Verifikationsværktøj Baseret på en Operationel Semantik”, Marster Thesis, Aalborg University Centre, 1986 1, 3.1, 3.1, 3.1, 5.2, 7

List of Figures

Appendix A

PROLOG implementation of minimal system

—arguing for bisimulation equivalence

Listed below is the PROLOG representation of the inference system constructed and verified by [Larsen]. For comments please refer to this document.

```
bisim(P,Q) :-
   closure(P,Q,[[P,Q]],C),
   prettyprint(P,Q,C,[]).

closure(P,Q,B,D) :-
   derset(P,M), matchl(P,Q,M,B,C),
   derset(Q,N), matchr(P,Q,N,C,D).

matchl(P,Q,[],B,B).
matchl(P,Q,[[A,P1]|M],B,D) :-
   der(Q,A,Q1),
   in([P1,Q1],B), !,
   matchl(P,Q,M,B,D).
matchl(P,Q,[[A,P1]|M],B,D) :-
   der(Q,A,Q1),
   closure(P1,Q1,[[P1,Q1]|B],C),
   matchl(P,Q,M,C,D).

matchr(P,Q,[],B,B).
matchr(P,Q,[[A,Q1]|N],B,D) :-
   der(P,A,P1),
   in([P1,Q1],B), !,
   matchr(P,Q,N,B,D).
matchr(P,Q,[[A,Q1]|N],B,D) :-
   der(P,A,P1),
   closure(P1,Q1,[[P1,Q1]|B],C),
   matchr(P,Q,N,C,D).
```

Appendix B

PROLOG implementation of new system

—arguing for bisimulation equivalence and inequivalence

Listed below is the full PROLOG representation of the new inference system for finding (strong) bisimulations (see figure 4.1), including the definitions of all auxilliary predicates. Also, a `help`-predicate has been included as to give on-line assistance of syntax etc. By entering

```
help.
```

you will be informed of which subjects help is available for and how to retrieve it.

Please notice, that it has been decided not to clobber-up the listing with too much (irrelevant) text, i.e. no explanations are given but to the nost unconventional functions/predicates. Instead, predicates has been grouped together in order to improve readability of the listing.

```
/****************************************************************************/
/*                                                                          */
/*      Expressions denoting Processes is given by the following            */
/*      grammar:                                                            */
/*                                                                          */
/*                                                                          */
/*          P ::=   i           (process identifier defined by :=:)         */
/*               |  nil         (inaction)                                  */
/*               |  a;P         (prefixing)                                 */
/*               |  P+P         (summation of processes)                    */
/*               |  P/P         (concurrent composition of processes)       */
/*               |  P\L         (restricts Ps action to L,                  */
/*                               where L is a list of atoms)                */
/*               |  P-[A:=B]    (renames Ps action according to A:=B,       */
/*                               where A and B are atoms)                   */
/*                                                                          */
/*                                                                          */
/*      The process identifiers are specified by simultaneous recursion:    */
/*                                                                          */
/*          i1 :=: P1.                                                      */
/*             .                                                            */
/*             .                                                            */
/*          in :=: Pn.                                                      */
/*                                                                          */
/*      where P1,...,Pn are process expressions over i1,...,in.             */
/*                                                                          */
/*                                                                          */
/****************************************************************************/



/****************************************************************************/
/*                                                                          */
/*                      O P E R A T I O N S                                 */
/*                                                                          */
/****************************************************************************/

:- op(40,xfy,:=:).
:- op(11,xfy,;).
:- op(20,xfy,+).
:- op(30,xfy,/).
:- op(15,xfy,\).
:- op(15,yfx,-).
:- op(14,xfy,'and').
:- op(13,xfy,'or').
:- op(12,fy,'no').
:- op(11,xfy,';').
:- op(11,xfy,'&').
:- op(10,xfy,:=).

str(and,and).
str(or,or).


/***************************************************************************/
/*                                                                         */
/*               A U X I L I A R Y  P R E D I C A T E S                    */
/*                                                                         */
/***************************************************************************/

in(A,[A|L]).
in(A,[B|L]) :- in(A,L).

append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

delete(A,[],[]).
delete(A,[A|L],M) :- delete(A,L,M).
delete(A,[B|L],[B|M]) :-  delete(A,L,M).

reduce([],[]).
reduce([A|L],[A|RL]) :- delete(A,L,L1), reduce(L1,RL).

reverse(L,RL) :- rev(L,[],RL).
rev([A|L],L2,L3) :- rev(L,[A|L2],L3).
rev([],RL,RL).

action(in(A)) :- atom(A).
action(out(A)) :- atom(A).
action(A) :- atom(A).

listofactions([]).
listofactions([A|L]) :- atom(A), listofactions(L).

listofassign([]).
listofassign([A:=B|L]) :- atom(A), atom(B), listofassign(L).



/***************************************************************************/
/*                                                                         */
/*              O P E R A T I O N A L  S E M A N T I C S                   */
/*                                                                         */
/***************************************************************************/

/*#snipplet: der.inp */
der(A;P,A,P).
der(P + Q,A,R) :- der(P,A,R).
der(P + Q,A,R) :- der(Q,A,R).
der(P / Q,A,R / Q) :- der(P,A,R).
der(P / Q,A,P / S) :- der(Q,A,S).
der(P / Q,tau,R / S) :- der(P,in(A),R), der(Q,out(A),S).
der(P / Q,tau,R / S) :- der(P,out(A),R), der(Q,in(A),S).
der(P\L,in(A),Q\L) :- in(A,L), der(P,in(A),Q).
der(P\L,out(A),Q\L) :- in(A,L), der(P,out(A),Q).
der(P\L,A,Q\L) :- in(A,L), der(P,A,Q).
der(P\L,tau,Q\L) :- der(P,tau,Q).
der(P-[A:=B],in(B),Q-[A:=B]) :- der(P,in(A),Q).
der(P-[A:=B],out(B),Q-[A:=B]) :- der(P,out(A),Q).
der(P-[A:=B],B,Q-[A:=B]) :- der(P,A,Q).
der(P-[A:=B],in(C),Q-[A:=B]) :- der(P,in(C),Q), A\= C.
der(P-[A:=B],out(C),Q-[A:=B]) :- der(P,out(C),Q), A\= C.
der(P-[A:=B],C,Q-[A:=B]) :- der(P,C,Q), atom(C), A\= C.
/*#end*/
der(P,A,Q) :- decl(P,B), der(B,A,Q).

derset(P,M) :- setof([A,Q],der(P,A,Q),M), !.
derset(P,[]).



/****************************************************************************/
/*                                                                          */
/*            O B S E R V A T I O N A L   S E M A N T I C S                 */
/*                                                                          */
/****************************************************************************/

/*#snipplet: obder.inp */
obder(P,tau,P).
obder(P,A,Q) :-
   der(P,tau,R), P\==R, obder(R,A,Q).
obder(P,A,Q) :-
   der(P,A,R), A\==tau, obder(R,tau,Q).
/*#end*/



/****************************************************************************/
/*                                                                          */
/*              R E C U R S I V E   D E F I N I T I O N S                   */
/*                                                                          */
/****************************************************************************/

N :=: B :- asserta(decl(N,B)).



/****************************************************************************/
/*                                                                          */
/*               S T R O N G   B I S I M U L A T I O N S                    */
/*                                                                          */
/*       To get weak bisimulations replace 'der' with 'obder'.              */
/*                                                                          */
/****************************************************************************/
```

```
/*#snipplet: newbisim.inp */
bisim(P,Q) :-
   nl, wstring("Searching for a strong bisimulation..."), nl,
   closure(P,Q,[[P,Q]],C,[],Fout), prettyprint(P,Q,C,Fout).

closure(P,Q,B,D,Fin,Fout) :-
   derset(P,M), matchl(P,Q,M,B,C,Fin,F1), not(in([P,Q,_],F1)), !,
   derset(Q,N), matchr(P,Q,N,C,D,F1,Fout).
closure(P,Q,B,D,Fin,Fout) :-
   derset(P,M), matchl(P,Q,M,B,D,Fin,Fout).

matchl(P,Q,[],B,B,Fin,Fin).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1), in([P1,Q1],B), !, matchl(P,Q,M,B,D,Fin,Fout).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   in([P1,Q1,_],F1), !,
   matchl(P,Q,[[A,P1]|M],B,D,F1,Fout).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   not(in([P1,Q1,_],F1)),
   matchl(P,Q,M,C,D,F1,Fout).
matchl(P,Q,[[A,P1]|M],B,B,Fin,[[P,Q,A&G]|Fin]) :-
   not(in([P,Q,_],Fin)), !,
   findall(F1, ( der(Q,A,Q1), in([P1,Q1,F1],Fin) ), LF),
   reduce(LF,RLF), conjunctlist(RLF,G).
matchl(P,Q,[[A,P1]|M],B,B,Fin,Fin).

matchr(P,Q,[],B,B,Fin,Fin).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1), in([P1,Q1],B), !, matchr(P,Q,N,B,D,Fin,Fout).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   in([P1,Q1,_],F1), !,
   matchr(P,Q,[[A,Q1]|N],B,D,F1,Fout).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1),
   not(in([P1,Q1,_],Fin)),
   closure(P1,Q1,[[P1,Q1]|B],C,Fin,F1),
   not(in([P1,Q1,_],F1)),
   matchr(P,Q,N,C,D,F1,Fout).
matchr(P,Q,[[A,Q1]|N],B,B,Fin,[[P,Q,A;G]|Fin]) :-
   not(in([P,Q,_],Fin)), !,
   findall(F1, ( der(P,A,P1), in([P1,Q1,F1],Fin) ), LF),
   reduce(LF,RLF), disjunctlist(RLF,G).
matchr(P,Q,[[A,Q1]|M],B,B,Fin,Fin).
/*#end*/


conjunctlist([],tt).
conjunctlist([F|[]],F).
conjunctlist([F|LF],F and G) :- conjunctlist(LF,G).

disjunctlist([],no tt).
disjunctlist([F|[]],F).
disjunctlist([F|LF],F or G) :- disjunctlist(LF,G).



/*****************************************************************************/
/*                                                                           */
/*                    H E L P   P R E D I C A T E S                          */
/*                                                                           */
/*****************************************************************************/

showdecl :-
   nl, decl(N,B), write(N), wstring(":=: "), write(B), wstring("."), nl, fail.
showdecl :- nl.

deldecl :- retractall(decl(N,P)), wstring("All declarations deleted."), nl.

showder :- listing(fderset).
cleanup :- retractall(fderset(A,B)).

bisim_type(strong).   /* change to 'weak' if weak bisimulations are desired */
bisim_type :- bisim_type(T), tab(1), write(T), tab(1).

help :-
   wstring("HELP menu for finding"), bisim_type,
   wstring("bisimulations."), nl, nl,
   wstring("To get information on a specific subject"), nl,
   wstring("you should type:   help( subj )"), nl,
   wstring("where 'subj' is one of the following subjects:"), nl, subjects.
help(Subject) :-
   subject(Subject), subject(Subject,info).
help(NoInfo) :-
   wstring("Sorry, no information available on '"), write(NoInfo),
   wstring("'."), nl.

subjects :-
   subject(Subject), write(Subject), nl, fail.
```

```
subjects.

subject(syntax).
subject(syntax,info) :-
   wstring("P ::=   i           (process identifier defined by :=:)"), nl,
   wstring("      |  nil         (inaction)"), nl,
   wstring("      |  a;P         (prefixing)"), nl,
   wstring("      |  P+P         (summation of processes)"), nl,
   wstring("      |  P/P         (concurrent composition of processes)"), nl,
   wstring("      |  P\\L         (restricts Ps action to L,"), nl,
   wstring("                      where L is a list of atoms)"), nl,
   wstring("      |  P-[A:=B]    (renames Ps action according to A:=B,"), nl,
   wstring("                      where A and B are atoms)"), nl, nl,
   wstring("A special action is the one denoting 'silent' actions, tau"), nl,
   wstring("Complementaty actions are denoted by te two predicates"), nl,
   wstring("      in(action)   and   out(action)"), nl,
   wstring("Example:  in(a) is the complement of out(a)."), nl.

subject(modal_properties).
subject(modal_properties,info) :-
   wstring("Modal properties are used in order to describe the"), nl,
   wstring("properties which a process enjoy. The modal language"), nl,
   wstring("used consists of the following constructs"), nl,
   wstring("        tt    the property which all processes enjoy"), nl,
   bisim_type(strong),
   wstring("     <a>tt    it is possible to perform an a-experiment"), nl,
   wstring(" not <a>tt    it is not possible to perform an a-experiment"), nl,
   wstring("The following short hands are used:"), nl,
   wstring("        ff    stands for    not tt"), nl,
   wstring("     [a]ff    stands for    not <a> not tt"), nl,
   wstring("Furthermore, 'and' and 'or' are used as"), nl,
   wstring("conjunction and disjunction of properties."), nl.
subject(modal_properties,info) :-
   wstring("   <<a>>tt    it is possible to perform an a-experiment"), nl,
   wstring("not<<a>>tt    it is not possible to perform an a-experiment"), nl,
   wstring("The following short hands are used:"), nl,
   wstring("        ff    stands for    not tt"), nl,
   wstring("   [[a]]ff    stands for    not <<a>> not tt"), nl,
   wstring("Furthermore, 'and' and 'or' are used as"), nl,
   wstring("conjunction and disjunction of properties."), nl.

subject(declarations).
subject(declarations,info) :-
   wstring("Declarations are use in order to define short hand"), nl,
   wstring("notations of CCS expressions and/or to define recursive"), nl,
   wstring("processes, ex."), nl,
   wstring("    bad_machine :=: in(coin);out(coffee);bad_machine +"), nl,
   wstring("                    in(coin);out(tea);bad_machine."), nl,
   wstring("The set of all declarations defined so far can be viewed by"), nl,
   wstring("   showdecl."), nl,
   wstring("and all declarations are erased by"), nl,
   wstring("   deldecl."), nl.
```

```
/*****************************************************************************/
/*                                                                           */
/*                     P R E T T Y   P R I N T I N G                         */
/*                                                                           */
/*****************************************************************************/

wstring([]).
wstring([A|L]) :-
   put(A), !, wstring(L).

prettyprint(P,Q,C,Fout) :- in([P,Q,F],Fout), sadmsg(P,Q), !, prp(F,O), nl, nl.
prettyprint(P,Q,C,Fout) :- happymsg(P,Q), reverse(C,RC), prettybisim(RC), nl.

happymsg(P,Q) :-
   wstring("Here is a listing of a strong bisimulation of the processes"),
   nl, tab(5), write(P), tab(5), wstring("and"), tab(5), write(Q),
   nl, nl.

sadmsg(P,Q) :-
   wstring("The process"), nl, tab(5), write(P), nl,
   wstring("enjoys some properties which the process"), nl,
   tab(5),
   write(Q), nl, wstring("don't. One property is:"), nl, tab(5).


prettybisim([]).
prettybisim([[P,Q]|L]) :- nl, write(P), nl,
                tab(15), write(Q), nl, prettybisim(L).


prp(F1 or F2,O) :-
   str(or,O),
   prp(F1,or), wstring(" or "), prp(F2,or), !.
prp(F1 or F2,O) :-
   wstring("("), prp(F1,or),
   wstring(" or "), prp(F2,or),
   wstring(")").
prp(F1 and F2,O) :-
   str(and,O),
   prp(F1,and), wstring(" and "), prp(F2,and), !.
prp(F1 and F2,O) :-
   wstring("("), prp(F1,and),
   wstring(" and "), prp(F2,and),
   wstring(")").
prp(no (no F),O) :-
   prp(F,O).
prp(no (F1 or F2),O) :-
   prp((no F1) and (no F2),O).
prp(no (F1 and F2),O) :-
```

```
   prp((no F1) or (no F2),O).
prp(no (A;F),O) :-
   prp(A&(no F),O).
prp(no (A&F),O) :-
   prp(A;(no F),O).
prp(no tt,O) :-
   wstring("ff").
prp(A;F,O) :-
   wstring("["), write(A), wstring("]"), prp(F,;).
prp(A&F,O) :-
   wstring("<"), write(A), wstring(">"), prp(F,&).
prp(tt,O) :-
   wstring("tt"), !.
prp(ff,O) :-
   wstring("ff"), !.
prp(F,O) :-
   nl, nl, wstring("This is an impossible situation!!!"), nl.
```

Appendix C

PROLOG implementation of extended system

—arguing for extended bisimulation equivalence and inequivalence

Listed below is the full PROLOG representation of the extended inference system for finding (strong) bisimulations (see figures 6.2 and 6.3), including the definitions of all auxilliary predicates. Also, a `help`-predicate has been included as to give on-line assistance of syntax etc. By entering

```
help.
```

you will be informed of which subjects help is available for and how to retrieve it.

Please notice, that it has been decided not to clobber-up the listing with too much (irrelevant) text, i.e. no explanations are given but to the nost unconventional functions/predicates. Instead, predicates has been grouped together in order to improve readability of the listing.

```
/****************************************************************************/
/*                                                                          */
/*      Expressions denoting Processes is given by the following           */
/*      grammar:                                                            */
/*                                                                          */
/*                                                                          */
/*          P ::=   i           (process identifier defined by :=:)         */
/*               |  nil         (inaction)                                  */
/*               |  a;P         (prefixing)                                 */
/*               |  P+P         (summation of processes)                    */
/*               |  P/P         (concurrent composition of processes)       */
/*               |  P\L         (restricts Ps action to L,                  */
/*                               where L is a list of atoms)                */
/*               |  P-[A:=B]    (renames Ps action according to A:=B,       */
/*                               where A and B are atoms)                   */
/*                                                                          */
/*                                                                          */
/*      The process identifiers are specified by simultaneous recursion:    */
/*                                                                          */
/*          i1 :=: P1.                                                      */
/*             .                                                            */
/*             .                                                            */
/*          in :=: Pn.                                                      */
/*                                                                          */
/*      where P1,...,Pn are process expressions over i1,...,in.            */
/*                                                                          */
/*                                                                          */
/****************************************************************************/


/****************************************************************************/
/*                                                                          */
/*                      O P E R A T I O N S                                 */
/*                                                                          */
/****************************************************************************/

:- op(40,xfy,:=:).
:- op(11,xfy,;).
:- op(20,xfy,+).
:- op(30,xfy,/).
:- op(15,xfy,\).
:- op(15,yfx,-).
:- op(14,xfy,'and').
:- op(13,xfy,'or').
:- op(12,fy,'no').
:- op(11,xfy,';').
:- op(11,xfy,'&').
:- op(10,xfy,:=).

str(and,and).
str(or,or).


/***************************************************************************/
/*                                                                         */
/*                  A U X I L I A R Y   P R E D I C A T E S                */
/*                                                                         */
/***************************************************************************/

in(A,[A|L]).
in(A,[B|L]) :- in(A,L).

append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

delete(A,[],[]).
delete(A,[A|L],M) :- delete(A,L,M).
delete(A,[B|L],[B|M]) :-  delete(A,L,M).

reduce([],[]).
reduce([A|L],[A|RL]) :- delete(A,L,L1), reduce(L1,RL).

reverse(L,RL) :- rev(L,[],RL).
rev([A|L],L2,L3) :- rev(L,[A|L2],L3).
rev([],RL,RL).

action(in(A)) :- atom(A).
action(out(A)) :- atom(A).
action(A) :- atom(A).

listofactions([]).
listofactions([A|L]) :- atom(A), listofactions(L).

listofassign([]).
listofassign([A:=B|L]) :- atom(A), atom(B), listofassign(L).



/***************************************************************************/
/*                                                                         */
/*                 O P E R A T I O N A L   S E M A N T I C S               */
/*                                                                         */
/***************************************************************************/

der(A;P,A,P).
der(P + Q,A,R) :- der(P,A,R).
der(P + Q,A,R) :- der(Q,A,R).
der(P / Q,A,R / Q) :- der(P,A,R).
der(P / Q,A,P / S) :- der(Q,A,S).
der(P / Q,tau,R / S) :- der(P,in(A),R), der(Q,out(A),S).
der(P / Q,tau,R / S) :- der(P,out(A),R), der(Q,in(A),S).
der(P\L,in(A),Q\L) :- in(A,L), der(P,in(A),Q).
der(P\L,out(A),Q\L) :- in(A,L), der(P,out(A),Q).
der(P\L,A,Q\L) :- in(A,L), der(P,A,Q).
der(P\L,tau,Q\L) :- der(P,tau,Q).
der(P-[A:=B],in(B),Q-[A:=B]) :- der(P,in(A),Q).
der(P-[A:=B],out(B),Q-[A:=B]) :- der(P,out(A),Q).
der(P-[A:=B],B,Q-[A:=B]) :- der(P,A,Q).
der(P-[A:=B],in(C),Q-[A:=B]) :- der(P,in(C),Q), A\= C.
der(P-[A:=B],out(C),Q-[A:=B]) :- der(P,out(C),Q), A\= C.
der(P-[A:=B],C,Q-[A:=B]) :- der(P,C,Q), atom(C), A\= C.
der(P,A,Q) :- decl(P,B), der(B,A,Q).

derset(P,M) :- setof([A,Q],der(P,A,Q),M), !.
derset(P,[]).



/****************************************************************************/
/*                                                                          */
/*              O B S E R V A T I O N A L   S E M A N T I C S               */
/*                                                                          */
/****************************************************************************/

obder(P,tau,P).
obder(P,A,Q) :-
   der(P,tau,R), P\==R, obder(R,A,Q).
obder(P,A,Q) :-
   der(P,A,R), A\==tau, obder(R,tau,Q).



/****************************************************************************/
/*                                                                          */
/*                R E C U R S I V E   D E F I N I T I O N S                 */
/*                                                                          */
/****************************************************************************/

N :=: B :- asserta(decl(N,B)).



/****************************************************************************/
/*                                                                          */
/*                 S T R O N G   B I S I M U L A T I O N S                  */
/*                                                                          */
/*         To get weak bisimulations replace 'der' with 'obder'.            */
/*                                                                          */
/****************************************************************************/

bisim(P,Q) :-
   nl, wstring("Searching for a strong bisimulation..."), nl,
   closure(P,Q,[[P,Q]],C,[],Fout), prettyprint(P,Q,C,Fout).
closure(P,Q,B,D,Fin,Fout) :-
   derset(P,M), matchl(P,Q,M,B,C,Fin,F1), not(in([P,Q,_],F1)), !,
   derset(Q,N), matchr(P,Q,N,C,D,F1,Fout).
closure(P,Q,B,D,Fin,Fout) :-
   derset(P,M), matchl(P,Q,M,B,D,Fin,Fout).

matchl(P,Q,[],B,B,Fin,Fin).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1), in([P1,Q1,_,_],B),
   add_f(P,Q,[[A,P1],Q1],B,B1), !,
   matchl(P,Q,M,B1,D,Fin,Fout).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1),
   not(in([P1,Q1,_],Fin)),
   add_f(P,Q,[[A,P1],Q1],B,B1),
   closure(P1,Q1,[[P1,Q1,[],[]]|B1],C,Fin,F1),
   in([P1,Q1,_],F1), !,
   matchl(P,Q,[[A,P1]|M],B,D,F1,Fout).
matchl(P,Q,[[A,P1]|M],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(Q,A,Q1),
   not(in([P1,Q1,_],Fin)),
   add_f(P,Q,[[A,P1],Q1],B,B1), !,
   closure(P1,Q1,[[P1,Q1,[],[]]|B1],C,Fin,F1),
   not(in([P1,Q1,_],F1)),
   matchl(P,Q,M,C,D,F1,Fout).
matchl(P,Q,[[A,P1]|M],B,B,Fin,[[P,Q,A&G]|Fin]) :-
   not(in([P,Q,_],Fin)), !,
   findall(F1, ( der(Q,A,Q1), in([P1,Q1,F1],Fin) ), LF),
   reduce(LF,RLF), conjunctlist(RLF,G).
matchl(P,Q,[[A,P1]|M],B,B,Fin,Fin).

matchr(P,Q,[],B,B,Fin,Fin).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1), in([P1,Q1,_,_],B),
   add_g(P,Q,[[A,Q1],P1],B,B1), !,
   matchr(P,Q,N,B1,D,Fin,Fout).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1),
   not(in([P1,Q1,_],Fin)),
   add_g(P,Q,[[A,Q1],P1],B,B1),
   closure(P1,Q1,[[P1,Q1,[],[]]|B1],C,Fin,F1),
   in([P1,Q1,_],F1), !,
   matchr(P,Q,[[A,Q1]|N],B1,D,F1,Fout).
matchr(P,Q,[[A,Q1]|N],B,D,Fin,Fout) :-
   not(in([P,Q,_],Fin)),
   der(P,A,P1),
   not(in([P1,Q1,_],Fin)),
   add_g(P,Q,[[A,Q1],P1],B,B1),
   closure(P1,Q1,[[P1,Q1,[],[]]|B1],C,Fin,F1),
   not(in([P1,Q1,_],F1)),
   matchr(P,Q,N,C,D,F1,Fout).
matchr(P,Q,[[A,Q1]|N],B,B,Fin,[[P,Q,A;G]|Fin]) :-
   not(in([P,Q,_],Fin)), !,
   findall(F1, ( der(P,A,P1), in([P1,Q1,F1],Fin) ), LF),
   reduce(LF,RLF), disjunctlist(RLF,G).
matchr(P,Q,[[A,Q1]|N],B,B,Fin,Fin).

conjunctlist([],tt).
conjunctlist([F|[]],F).
conjunctlist([F|LF],F and G) :- conjunctlist(LF,G).

disjunctlist([],no tt).
disjunctlist([F|[]],F).
disjunctlist([F|LF],F or G) :- disjunctlist(LF,G).

add_f(P,Q,D,[],[]).
add_f(P,Q,D,[[P,Q,F,G]|L],[[P,Q,[D|F],G]|L1]) :-
    add_f(P,Q,D,L,L1).
add_f(P,Q,D,[A|L],[A|L1]) :-
    add_f(P,Q,D,L,L1).

add_g(P,Q,D,[],[]).
add_f(P,Q,D,[[P,Q,F,G]|L],[[P,Q,F,[D|G]]|L1]) :-
    add_g(P,Q,D,L,L1).
add_g(P,Q,D,[A|L],[A|L1]) :-
    add_g(P,Q,D,L,L1).


/*************************************************************************/
/*                                                                       */
/*                      H E L P   P R E D I C A T E S                    */
/*                                                                       */
/*************************************************************************/

showdecl :-
   nl, decl(N,B), write(N), wstring(":=: "), write(B), wstring("."), nl, fail.
showdecl :- nl.

deldecl :- retractall(decl(N,P)), wstring("All declarations deleted."), nl.

showder :- listing(fderset).
cleanup :- retractall(fderset(A,B)).

bisim_type(strong).   /* change to 'weak' if weak bisimulations are desired */
bisim_type :- bisim_type(T), tab(1), write(T), tab(1).

help :-
   wstring("HELP menu for finding"), bisim_type,
   wstring("bisimulations."), nl, nl,
   wstring("To get information on a specific subject"), nl,
   wstring("you should type:   help( subj )"), nl,
   wstring("where 'subj' is one of the following subjects:"), nl, subjects.
help(Subject) :-
   subject(Subject), subject(Subject,info).
help(NoInfo) :-
   wstring("Sorry, no information available on '"), write(NoInfo),
   wstring("'."), nl.

subjects :-
   subject(Subject), write(Subject), nl, fail.
subjects.

subject(syntax).
subject(syntax,info) :-
   wstring("P ::=   i           (process identifier defined by :=:)"), nl,
   wstring("     |  nil         (inaction)"), nl,
   wstring("     |  a;P         (prefixing)"), nl,
   wstring("     |  P+P         (summation of processes)"), nl,
   wstring("     |  P/P         (concurrent composition of processes)"), nl,
   wstring("     |  P\\L         (restricts Ps action to L,"), nl,
   wstring("                     where L is a list of atoms)"), nl,
   wstring("     |  P-[A:=B]    (renames Ps action according to A:=B,"), nl,
   wstring("                     where A and B are atoms)"), nl, nl,
   wstring("A special action is the one denoting 'silent' actions, tau"), nl,
   wstring("Complementaty actions are denoted by te two predicates"), nl,
   wstring("      in(action)   and   out(action)"), nl,
   wstring("Example:  in(a) is the complement of out(a)."), nl.

subject(modal_properties).
subject(modal_properties,info) :-
   wstring("Modal properties are used in order to describe the"), nl,
   wstring("properties which a process enjoy. The modal language"), nl,
   wstring("used consists of the following constructs"), nl,
   wstring("       tt    the property which all processes enjoy"), nl,
   bisim_type(strong),
   wstring("     <a>tt    it is possible to perform an a-experiment"), nl,
   wstring(" not <a>tt    it is not possible to perform an a-experiment"), nl,
   wstring("The following short hands are used:"), nl,
   wstring("       ff    stands for    not tt"), nl,
   wstring("     [a]ff    stands for    not <a> not tt"), nl,
   wstring("Furthermore, 'and' and 'or' are used as"), nl,
   wstring("conjunction and disjunction of properties."), nl.
subject(modal_properties,info) :-
   wstring("   <<a>>tt    it is possible to perform an a-experiment"), nl,
   wstring("not<<a>>tt    it is not possible to perform an a-experiment"), nl,
   wstring("The following short hands are used:"), nl,
   wstring("       ff    stands for    not tt"), nl,
   wstring("   [[a]]ff    stands for    not <<a>> not tt"), nl,
   wstring("Furthermore, 'and' and 'or' are used as"), nl,
   wstring("conjunction and disjunction of properties."), nl.
```

```
subject(declarations).
subject(declarations,info) :-
   wstring("Declarations are use in order to define short hand"), nl,
   wstring("notations of CCS expressions and/or to define recursive"), nl,
   wstring("processes, ex."), nl,
   wstring("    bad_machine :=: in(coin);out(coffee);bad_machine +"), nl,
   wstring("                    in(coin);out(tea);bad_machine."), nl,
   wstring("The set of all declarations defined so far can be viewed by"), nl,
   wstring("   showdecl."), nl,
   wstring("and all declarations are erased by"), nl,
   wstring("   deldecl."), nl.



/****************************************************************************/
/*                                                                          */
/*                     P R E T T Y   P R I N T I N G                        */
/*                                                                          */
/****************************************************************************/

wstring([]).
wstring([A|L]) :-
   put(A), !, wstring(L).

prettyprint(P,Q,C,Fout) :- in([P,Q,F],Fout), sadmsg(P,Q), prettyproperties(F,0),
   nl, nl.
prettyprint(P,Q,C,Fout) :- happymsg(P,Q), !, postnumber(No,C,C1), !,
                           postprocess(C1,C1,C2), !, reverse(C2,RC2), prettybisim(RC2),
   nl.

happymsg(P,Q) :-
   wstring("Here is a listing of a strong bisimulation of the processes"),
   nl, tab(5), write(P), tab(5), wstring("and"), tab(5), write(Q),
   nl, nl.

sadmsg(P,Q) :-
   wstring("The process"), nl, tab(5), write(P), nl,
   wstring("enjoys some properties which the process"), nl,
   tab(5),
   write(Q), nl, wstring("doesn't. One property is:"), nl, tab(5).



postnumber(0,[],[]).
postnumber(No1,[[P,Q,F,G]|L],[[No,P,Q,F,G|L1]) :-
    postnumber(No,L,L1), No1 is No+1.


f_action_to_pair(B,[],[]).
f_action_to_pair(B,[[[A,P],Q]|L],[[A,No]|L1]) :-
    in([No,P,Q,F,G],B), f_action_to_pair(B,L,L1).
```

```
g_action_to_pair(B,[],[]).
g_action_to_pair(B,[[[A,Q],P]|L],[[A,No]|L1]) :-
    in([No,P,Q,F,G],B), g_action_to_pair(B,L,L1).


postprocess(C,[],[]).
postprocess(C,[[No,P,Q,F,G]|L],[[No,P,Q,F1,G1]|L1]) :-
    reduce(F,RF), f_action_to_pair(C,RF,F1), !
    reduce(G,RG), g_action_to_pair(C,Rg,G1), !
    postprocess(C,L,L1).


prettybisim([]).
prettybisim([[No,P,Q,F,G]|L]) :-
    write(No), nl,
    write(P), nl,
    tab(15), write(Q), nl,
    print_action_to_pair("LHS",F), !,
    print_action_to_pair("RHS",G), !,
    prettybisim(L).

print_action_to_pair(Side,[]).
print_action_to_pair(Side,[[A,No]|L]) :-
    tab(25), wstring(Side), wstring("-> "), write(A),
    wstring(" ==> "), write(No), nl, !, print_action_to_pair(Side,L).


prp(F1 or F2,O) :-
   str(or,O),
   prp(F1,or), wstring(" or "), prp(F2,or), !.
prp(F1 or F2,O) :-
   wstring("("), prp(F1,or),
   wstring(" or "), prp(F2,or),
   wstring(")").
prp(F1 and F2,O) :-
   str(and,O),
   prp(F1,and), wstring(" and "), prp(F2,and), !.
prp(F1 and F2,O) :-
   wstring("("), prp(F1,and),
   wstring(" and "), prp(F2,and),
   wstring(")").
prp(no (no F),O) :-
   prp(F,O).
prp(no (F1 or F2),O) :-
   prp((no F1) and (no F2),O).
prp(no (F1 and F2),O) :-
   prp((no F1) or (no F2),O).
prp(no (A;F),O) :-
   prp(A&(no F),O).
prp(no (A&F),O) :-
   prp(A;(no F),O).
prp(no tt,O) :-
   wstring("ff").
prp(A;F,O) :-
   wstring("["), write(A), wstring("]"), prp(F,;).
prp(A&F,O) :-
   wstring("<"), write(A), wstring(">"), prp(F,&).
prp(tt,O) :-
   wstring("tt"), !.
prp(ff,O) :-
   wstring("ff"), !.
prp(F,O) :-
   nl, nl, wstring("This is an impossible situation!!!"), nl.
```

Appendix D

Examples of bisimulations

Listed below is the result of running all the bisimulation examples in the report through the system finding extended bisimulations (see appendix C). The listing also contains the results for excersise 1.4 of [Milner].

```
ideal_department:=: pub; ideal_department.
good_department :=: (computer_scientist/door/good_machine)\[pub].

computer_scientist :=: out(close); out(coin); in(coffee);
                       out(open); pub; computer_scientist.
door :=: in(close); in(open); door.

good_machine :=: in(coin); (out(coffee); good_machine +
                                          out(tea); good_machine).


bisim(good_department,ideal_department).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     good_department     and     ideal_department

0
good_department
               ideal_department
                         LHS--> epsilon ==> 1
                         RHS--> pub ==> 5
1
(out(coin);in(coffee);out(open);pub;computer_scientist/
 in(open);door/good_machine)\[pub]
               ideal_department
                         LHS--> epsilon ==> 2
                         RHS--> pub ==> 5
2
(in(coffee);out(open);pub;computer_scientist/in(open);door/
 out(coffee);good_machine+out(tea);good_machine)\[pub]
               ideal_department
                         LHS--> epsilon ==> 3
                         RHS--> pub ==> 5
3
(out(open);pub;computer_scientist/in(open);door/good_machine)\[pub]
               ideal_department
                         LHS--> epsilon ==> 4
                         RHS--> pub ==> 5
4
(pub;computer_scientist/door/good_machine)\[pub]
               ideal_department
                         LHS--> pub ==> 5
                         RHS--> pub ==> 5
5
(computer_scientist/door/good_machine)\[pub]
              ideal_department
                         LHS--> epsilon ==> 1
                         RHS--> pub ==> 5


yes


sys :=: (s0/m_sr0/r0/m_rs0)\[in,out].
spec :=: in; out; spec.
s0 :=: in; out(a); in(d); s0.
r0 :=: in(b); out; out(c); r0.
m_sr0 :=: in(a); out(b); m_sr0.
m_rs0 :=: in(c); out(d); m_rs0.


bisim(sys,spec).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     sys     and     spec

0
sys
              spec
                         LHS--> in ==> 1
                         RHS--> in ==> 1
1
(out(a);in(d);s0/m_sr0/r0/m_rs0)\[in,out]
              out;spec
                         LHS--> epsilon ==> 2
                         RHS--> out ==> 4
2
(in(d);s0/out(b);m_sr0/r0/m_rs0)\[in,out]
              out;spec
                         LHS--> epsilon ==> 3
                         RHS--> out ==> 4
3
(in(d);s0/m_sr0/out;out(c);r0/m_rs0)\[in,out]
              out;spec
                         LHS--> out ==> 4
```

```
                          RHS--> out ==> 4
4
(in(d);s0/m_sr0/out(c);r0/m_rs0)\[in,out]
               spec
                          LHS--> epsilon ==> 5
                          RHS--> in ==> 1
5
(in(d);s0/m_sr0/r0/out(d);m_rs0)\[in,out]
               spec
                          LHS--> epsilon ==> 6
                          RHS--> in ==> 1
6
(s0/m_sr0/r0/m_rs0)\[in,out]
               spec
                          LHS--> in ==> 1
                          RHS--> in ==> 1


yes
```

The following is the results of running exercise 1.4 of [Milner] through the same system. Note that we have run all nonequivalent problems through the system twice with different ordering of the parameters as to find a (unique) modal property for each process.

```
p1 :=: alpha;tau;gamma;nil.
q1 :=: alpha;gamma;nil.

bisim(p1,q1).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     p1     and     q1

0
p1
               q1
                          LHS--> alpha ==> 1
                          RHS--> alpha ==> 1
1
tau;gamma;nil
               gamma;nil
```

```
                        LHS--> epsilon ==> 2
                        RHS--> gamma ==> 3
2
gamma;nil
               gamma;nil
                        LHS--> gamma ==> 3
                        RHS--> gamma ==> 3
3
nil
               nil


yes

p2 :=: alpha;(beta;nil+tau;gamma;nil).
q2 :=: alpha;(beta;nil+gamma;nil).

bisim(p2,q2).

Searching for a weak bisimulation...
The process
     p2
enjoys some properties which the process
     q2
doesn't. One property is:
     <<alpha>>[[beta]]ff


yes

bisim(q2,p2).

Searching for a weak bisimulation...
The process
     q2
enjoys some properties which the process
     p2
doesn't. One property is:
     <<alpha>>([[epsilon]]<<beta>>tt and <<beta>>tt)


yes
```

```
p3 :=: alpha;(beta;nil+tau;gamma;nil).
q3 :=: alpha;(beta;nil+gamma;nil)+alpha;gamma;nil.

bisim(p3,q3).

Searching for a weak bisimulation...
The process
     p3
enjoys some properties which the process
     q3
doesn't. One property is:
     <<alpha>>(<<epsilon>>[[beta]]ff and <<beta>>tt)


yes

bisim(q3,p3).

Searching for a weak bisimulation...
The process
     q3
enjoys some properties which the process
     p3
doesn't. One property is:
     <<alpha>>([[epsilon]]<<beta>>tt and <<beta>>tt)


yes

p4 :=: alpha;nil+beta;nil+tau;beta;nil.
q4 :=: alpha;nil+tau;beta;nil.

bisim(p4,q4).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     p4     and     q4

0
p4
               q4
                         LHS--> epsilon ==> 2
                         LHS--> beta ==> 1
                         LHS--> alpha ==> 1
                         RHS--> epsilon ==> 2
                         RHS--> alpha ==> 1
1
nil
               nil
2
beta;nil
               beta;nil
                         LHS--> beta ==> 1
                         RHS--> beta ==> 1


yes

p5 :=: alpha;nil+beta;nil+tau;beta;nil.
q5 :=: alpha;nil+beta;nil.

bisim(p5,q5).

Searching for a weak bisimulation...
The process
     p5
enjoys some properties which the process
     q5
doesn't. One property is:
     <<epsilon>>[[alpha]]ff


yes

bisim(q5,p5).

Searching for a weak bisimulation...
The process
     q5
enjoys some properties which the process
     p5
doesn't. One property is:
     [[epsilon]]<<alpha>>tt
```

```
yes

p6 :=: alpha;(beta;nil+tau;beta;nil).
q6 :=: alpha;beta;nil.

bisim(p6,q6).

Searching for a weak bisimulation...
Here is a listing of a weak bisimulation of the processes
     p6     and     q6

0
p6
               q6
                         LHS--> alpha ==> 1
                         RHS--> alpha ==> 1
1
beta;nil+tau;beta;nil
               beta;nil
                         LHS--> epsilon ==> 3
                         LHS--> beta ==> 2
                         RHS--> beta ==> 2
2
nil
               nil
3
beta;nil
               beta;nil
                         LHS--> beta ==> 2
                         RHS--> beta ==> 2


yes

p7 :=: alpha;(beta;nil+tau;(gamma;nil+tau;delta;nil)).
q7 :=: alpha;(beta;nil+tau;(gamma;nil+tau;delta;nil))+
          alpha;(gamma;nil+tau;delta;nil)+alpha;delta;nil.

bisim(p7,q7).

Searching for a weak bisimulation...
```

```
Here is a listing of a weak bisimulation of the processes
     p7     and     q7

0
p7
               q7
                         LHS--> alpha ==> 1
                         RHS--> alpha ==> 4
                         RHS--> alpha ==> 3
                         RHS--> alpha ==> 1
1
beta;nil+tau;(gamma;nil+tau;delta;nil)
               beta;nil+tau;(gamma;nil+tau;delta;nil)
                         LHS--> epsilon ==> 3
                         LHS--> beta ==> 2
                         RHS--> epsilon ==> 3
                         RHS--> beta ==> 2
2
nil
               nil
3
gamma;nil+tau;delta;nil
               gamma;nil+tau;delta;nil
                         LHS--> epsilon ==> 4
                         LHS--> gamma ==> 2
                         RHS--> epsilon ==> 4
                         RHS--> gamma ==> 2
4
delta;nil
               delta;nil
                         LHS--> delta ==> 2
                         RHS--> delta ==> 2


yes


p8 :=: alpha;(beta;nil+tau;(gamma;nil+tau;delta;nil)).
q8 :=: alpha;(beta;nil+gamma;nil+delta;nil)+
          alpha;(gamma;nil+delta;nil)+alpha;delta;nil.

bisim(p8,q8).
```

```
Searching for a weak bisimulation...
The process
     p8
enjoys some properties which the process
     q8
doesn't. One property is:
     <<alpha>>(<<epsilon>>[[beta]]ff and <<beta>>tt)


yes

bisim(q8,p8).

Searching for a weak bisimulation...
The process
     q8
enjoys some properties which the process
     p8
doesn't. One property is:
     <<alpha>>([[epsilon]]<<beta>>tt and <<beta>>tt)


yes
```