Abstract

This report develops the theory of a modal process logic first introduced by Kim Larsen and Bent Thomsen. The logic can be seen as an extension of $\mathcal{CCS}$ and has an operational semantics upon which a *preorder* $\lhd$ on formulae in the logic is defined. A *denotational description* originally put forward by Kim Larsen and Bent Thomsen is examined, and it is shown to what extent $\lhd$ explains the ordering introduced by this kind of semantic description. In particular it is shown how the modal process logic semantically subsumes the language of partial specifications also introduced by Kim Larsen and Bent Thomsen. The *static constructs* of $\mathcal{CCS}$ are added to the logic, and a new *'observational' preorder* $\unlhd$ is introduced and used on three examples. Finally, sound and complete *proof systems* are given for $\lhd$ and the equality defined by the denotational semantics.

Master of Science Thesis
Department of Mathematics and Computer Science

Aalborg University Centre

Operational and Denotational Properties of a Modal Process Logic

Hans Hüttel

May 1988

Preface

This report was originally published under the same title as my M.Sc. thesis in Computer Science from Aalborg University Center. Some minor corrections have been made, but apart from these the text is unaltered.

References to literature in the bibliography (page 104) are written in brackets as e.g. [Larsen 87]. Most proofs of theorems in this report are found in the Appendix, beginning on page 90.

Hans Hüttel

Contents

Chapter 1

Introduction

In this report we develop the theory of the Modal Process Logic (MPL for short), introduced by Kim Larsen and Bent Thomsen, and intended for specification of nondeterministic and concurrent systems [Larsen and Thomsen 88]. In this chapter we start out with a discussion of the use of and requirements to mathematical models used in the specification of concurrent systems. We discuss the pros and cons of process calculi and modal logic as specification tools. This leads to a short presentation of MPL and a presentation of the topics dealt with in the rest of the report.

1.1 Requirements to specification languages

Program development is the activity where people (try to) solve a problem through the construction of an executable computer program. In an earlier work on interactive programming tools [Bækgaard et al 87] we saw program development as consisting of three major activities: An *analysis* of the problem, the *design* of a solution and the *realization* of the design through the implementation of the program. The products of program development activities are various *descriptions*. We will call the products of the design activities *specifications* and the products of the realization activities *implementations*.

The activities mentioned are of course not at all independent. The main direction is, of course, from analysis through design towards the realization, but a lot of feed-back is normal (and essential). An important activity which may lead to a renewed design is a *formal verification* of the implementation.

The above description gives rise to a series of requirements to the languages used for specifications. In general, the requirement is that a specification languages should support the steps taken towards an implementation and the verification thereof. An important requirement is then that a specification language must have a formal semantic description so that it itself is precisely determined. This is essential in the development of the theory, in verification, and in providing methods for automating aspects of program development.

One should also be able to tell when and if an implementation satisfies a specification. In the same vein, one should be able to compare the strength of specifications through an imposed ordering, saying that one specification is more concrete than another in the sense that it is more specific (i.e. restrictive) w.r.t. the implementations it permits. This will aid in a stepwise refinement of the design towards an implementation. Such satisfiability and refinement relations between specifications and implementations should be decidable, since the decision procedures might form the basis for programming tools. Another good tool when working with the ordering relations would be sound and complete *proof systems* for these.

A specification language should be *expressibly abstract*; it must not put unwanted restrictions on the possible implementations. When designing the solution one should not have to worry too much about details in the implementation, details that may not be known.

Last but certainly not least, a specification should support *compositionality* w.r.t. the implementation in design and verification. The designer should be able to decompose a specification (and thus the problem to be solved) into subspecifications that can be separately implemented and verified. It is particularly important that the same kind of compositionality is available in both the specification and implementation languages.

1.2 Specification languages for concurrency

For sequential systems the theories behind specification languages are already relatively well-established, in part because of the results in theoretical computer science aiming to establish semantic theories that specifically support the design and verification of sequential systems. This is *not* the case when we discuss concurrent systems, since the field of study is relatively new and no unified theory exists. However, at least two main theories have emerged, namely the various process calculi and the modal and temporal logics. One should bear in mind that the theories so far have only been used on small examples and that it may be a long way until they can be used in realistic-size specifications. Nevertheless, a theoretical development will provide us with a lot of important insight which may later turn out useful in actual program development. Well then, how do the process calculi and modal logics meet our requirements from the previous section ?

1.2.1 Process calculi as specification languages

Several process calculi have been proposed, inspired by the λ-calculus and related formalisms. One of the most important is Robin Milner's $\mathcal{CCS}$ [Milner 80] and its spin-offs [Milner 83]. Various equivalences between processes have been proposed in order to capture various observational aspects of their behaviour; in $\mathcal{CCS}$ strong and weak bisimulation ($\sim$ and $\approx$, respectively). These have been defined through a structural operational description of $\mathcal{CCS}$. Decision procedures for $\sim$ and $\approx$ exist and have been implemented [Larsen 86]. (These decision procedures only work for finite-state processes. Though a restrictive class, it is considered an important one in the study of network protocols.)

The equivalences allow $\mathcal{CCS}$ itself to be used for specification purposes. One simply says that an implementation p (written in $\mathcal{CCS}$) satisfies a specification F (also written in $\mathcal{CCS}$) iff $p \sim F$ (or $p \approx F$).

$\mathcal{CCS}$ is thus not abstract enough: The possible implementations with respect to a specification are simply members of an equivalence class. Recently Kim Larsen has shown how one can overcome this problem by relaxing the equivalence conditions through a context-dependent equivalence [Larsen 86]. Still, this does not remove the fact that the *language* $\mathcal{CCS}$ is too concrete. The only known, interesting orderings one can give on $\mathcal{CCS}$ -processes are, in fact, equivalences.

On the other hand $\mathcal{CCS}$ supports compositionality in the structure of the implementation through various process constructs with respect to which the equivalences are congruence relations.

1.2.2 Modal logics as specification languages

The usefulness of modal/temporal logic as a program specification language was first perceived by Amir Pnueli [Pnueli 77]. Temporal logic specifications are logical formulae dealing with dynamic properties of programs such as invariants ('this holds throughout the execution') and eventualities ('this will hold sometime during the execution')

The semantics of temporal logic is normally given in terms of a Kripke-style possible-worlds description where the possible worlds are the states in a program execution. The semantics of temporal logic is denotational in the sense that it describes the meaning of a formula through the meanings of its immediate constituents. It is, however, not compositional w.r.t. the structure of the implementations.

The programs satisfying a temporal logic specification are the ones for which the formulae in the specification are valid in the start state. Temporal logic is very abstract and does not limit the possible implementation to one single equivalence class.

One may define an ordering on modal logic formulae through use of the logical implication $\supset$, saying that specification F is a refinement of specification G if $F \supset G$ is a valid formula. This means that the refinement ordering itself becomes part of the language. It also means that a decision procedure for this ordering must check the validity of a logical formula. However, validity-checking can be shown to be coNP-hard [Garey and Johnson 1979] even for propositional modal logics.

A problem with temporal logic has often been that the implementation language with which it has been used has been rather ad hoc. However, in [Hennessy and Milner 85] Hennessy and Milner described a modal logic, now known as Hennessy-Milner logic (or just HML), with $\mathcal{CCS}$ as the implementation language. They also gave a denotational semantics of their logic; the meaning of a formula is a set of $\mathcal{CCS}$ -processes. The satisfiability relation is written as $\models p : F$ meaning that process p satisfies the HML specification F. It turned out that the logic characterizes $\sim$ in that

$$p \sim q \Leftrightarrow \{\forall F \in \mathcal{M} \mid \ \models p : F \Leftrightarrow \models q : F\} \tag{1.1}$$

where $\mathcal{M}$ is the set of HML formulae.

Recently Kim Larsen showed [Larsen 87] how one, by extending the logic with recursion, could express the properties normally considered in temporal logic. He also gave sound and complete proof systems for this extended HML.

Based on this, Godskesen, Ingolfsdottir and Zeeberg showed that the above modal characterization of $\sim$ also holds in recursive HML [Godskesen et al 87]. Furthermore, they showed how one for an HML-formula F could generate a $\mathcal{CCS}$ -process p such that $\models p : F$ (if any such p exists) and how one for a process p can generate a strongest formula F_p such that $\models p : F_p$. Finally they showed how one for any given HML formula F and $\mathcal{CCS}$ -context $C[\ ]$ can find another formula $I_C(F)$ such that $\models q : I_C(F) \Leftrightarrow \models C[q] : F$, i.e. a property transformation from a specification and an unfinished implementation to a specification describing how to ‘finish’ the implementation. Thus Hennessy-Milner logic supports compositionality.

All this shows that HML is indeed a very powerful specification language. Besides, its theoretical foundation is very firm. However, the ordering of HML specifications is still based on the denotational description and is thus not useful for effective decision procedures.

1.3 The Modal Process Logic

The Modal Process Logic (MPL) was introduced by Kim Larsen and Bent Thomsen in [Larsen and Thomsen 88], based on earlier experience with so-called partial specifications [Thomsen 87][Larsen and Thomsen 87]. They were also inspired by the work of Graf and Sifakis [Graf and Sifakis 86] in which the set of programs is seen as a subset of the set of specifications.

The original MPL is a simple extension of the regular expressions in $\mathcal{CCS}$ in that it contains exactly the dynamic process constructs of that language (action prefixing, nondeterminism and recursion). It is through the process constructs that compositionality is sought. (The syntax of MPL, along with the other theoretical concepts in this section, will be defined formally in the chapters to come) The implementation language considered consists of the processes of a process system with process constructs and equivalences as for $\mathcal{CCS}$.

The idea with MPL is that specifications are to describe the operational behaviour of implementations in a precise way, This has lead to a *structural operational semantics* (in the tradition of [Plotkin 81]) for MPL.

An MPL formula describes the transitions of possible $\mathcal{CCS}$ - implementations through its own transitions. One can specify two kinds of transitions, the ones that all implementations *must* possess and the ones that some implementations *may* have. In the operational semantics of MPL one uses a labelled transition system with two transition relations indexed with the modalities $\Box$ and $\Diamond$, $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$. $\Box$ - transitions are the necessary transitions, $\Diamond$ -transitions the permissible ones. These modalities are thus *deontic* [1] rather than temporal.

In the MPL language itself the action prefixing construct is correspondingly augmented with *modalities*, so prefix-formulae can be of the form

$a_\Box.F$ meaning that any implementation satisfying this specification *must* have a-transitions, all leading to implementations satisfying the specification F.

$a_\Diamond.F$ meaning that an implementation satisfying this specification *may* have a-transitions, all leading to implementations satisfying the specification F.

Specifications in MPL are ordered by means of a bisimulation-like preorder, $\lhd$, often referred to as the *refinement* ordering. Loosely speaking,

[1]The kind of modal logic where $\Box$ and $\Diamond$ are interpreted as 'it ought to be the case that' and 'it is permissible for it to be the case that' is called deontic logic [Hughes and Creswell 84]

$F \lhd G$ iff all transitions required by G are also required by F and result in derived specifications that are still related by $\lhd$ and all transitions allowed by F are also allowed by G with results of matching transitions also related by $\lhd$. One can also compare $\mathcal{CCS}$ - processes and MPL-formulae through $\lhd$ simply by saying that all moves made by a process are both $\Box$ - and $\Diamond$ - moves. One then has an *operationally defined* way of expressing 'p satisfies F'.

Kim Larsen and Bent Thomsen also *denotationally defined* that 'p satisfies F'. Their semantic description is HML-like in that the meaning of a formula F, $[\![F]\!]$, is a set of processes. It turns out that their semantic function has the pleasant property that

$$[\![F]\!] = \{p \mid p \lhd F\} \tag{1.2}$$

showing that the two viewpoints are related.

1.4 Open problems in the theory of MPL

There are, however, several important open problems left in the theory of MPL. These are the ones we set out to solve in this report:

- How good is the result on full abstractness (1.2) ? Since $\lhd$, being a preorder, is transitive, we get as a corollary that

  $$F \lhd G \Rightarrow [\![F]\!] \subseteq [\![G]\!]$$

  Does the converse

  $$[\![F]\!] \subseteq [\![G]\!] \Rightarrow F \lhd G \tag{1.3}$$

  also hold ? It would be nice indeed for then $[\![\ ]\!]$ would characterize the refinement ordering completely.

- The partial specifications introduced in [Larsen and Thomsen 87] use the same process constructs as MPL, except for the modalities. A refinement ordering on partial specifications is defined, and it is also called $\lhd$. How are the two languages and their refinement orderings related ? Can we give a denotational description of partial specifications having the same pleasant properties of full abstractness w.r.t. $\lhd$?

- The process constructs introduced in [Larsen and Thomsen 88] are the dynamic constructs of $\mathcal{CCS}$ and yield a language of *regular* specifications corresponding to the finite automata. In general, however, systems often have infinitely many states. This occurs for instance when one introduces value passing or dynamically changing configurations, A regular MPL is therefore not rich enough for the description of concurrent systems. How can one introduce static operators in MPL ? Is $\triangleleft$ then still the optimal relation on specifications ? When static constructs are introduced into $\mathcal{CCS}$, one usually also introduces the observational bisimulation, $\approx$. What would an 'observational $\triangleleft$ ' look like ?

- $\triangleleft$ is defined so that it resembles $\sim$ closely. Can we give sound and complete proof systems for $\triangleleft$ like one can for $\sim$ [Milner 82] ?

- In [Graf and Sifakis 86] Graf and Sifakis introduce the Boolean connectives of the propositional calculus into the language of regular $\mathcal{CCS}$ -formulae. This extended language is given a denotational description and a sound proof system for semantic equality is presented. What is gained if we augment MPL with (one or more of) the Boolean connectives ? What will the denotational semantics look like ? Can we give a sound and complete proof system for semantic equality ? (The proof system given in [Graf and Sifakis 86] is *not* shown to be complete) And how about the the feasibility of operational semantics in a language with logical connectives ?

- And last, but certainly not least : How can we use MPL for the specification of nondeterministic and concurrent systems ? How well are we off using MPL instead of, say, $\mathcal{CCS}$?

1.5 Overview of the rest of the report

Chapter 2 deals with transition systems for describing the operational semantics of specifications and processes. In particular, we define the notions of bisimulation and refinement.

Chapter 3 contains a description of the syntax and operational semantics of the languages for defining specifications and processes.

Chapter 4 starts out with a definition of the denotational semantics of MPL. Counterexamples show that (1.3) does not hold in MPL. However, it turns out that we can give a characterization of a class of counterexamples of (1.3) in MPL. Then the language of partial specifications is related to MPL through a presentation of its operational and denotational semantics. We show that (1.2) holds for the partial specifications as well as the converse result (1.3).

Chapter 5 introduces static constructs into MPL and develops the theory of a 'weak $\lhd$ ', illustrated with examples.

Chapter 6 gives proof systems for $\lhd$ for MPL's with dynamic constructs with or without recursion. Disjunction ($\vee$) is introduced into MPL, and it turns out that one can now give a sound and complete proof system for semantic equality in a language without recursion or static constructs, based on the results on full abstractness in Chapter 4.

Chapter 8 is the conclusion. We make a review of the goals achieved and future work to be done.

Chapter 2

Transition systems as a model of concurrency

In this chapter we introduce the concept of and notations for transition systems, used in the structural operational semantics of MPL and the language used for describing implementations, a subset of $\mathcal{CCS}$. We also define bisimulation equivalence ($\sim$) along with the refinement ordering ($\lhd$). This should show the reader the relatedness of the two concepts.

2.1 Labelled transition systems

The languages we will present next chapter describe the behaviour of concurrent systems through the set of actions they offer at any given instant. This idea originates with Milner [Milner 80] and leads naturally to the use of structural operational semantics as introduced by Plotkin [Plotkin 81], using the concept of *labelled transition systems*:

Definition 2.1 *A* labelled transition system $\mathcal{T} = (\Gamma, Act, \rightarrow)$ *is a triple where* Γ *is the set of* states, Act *is the set of* actions *and* $\rightarrow$ *is the* transition relation *satisfying*

$$\rightarrow \subseteq \Gamma \times Act \times \Gamma$$

Instead of writing $(p, a, q) \in \rightarrow$ one usually writes $p \xrightarrow{a} q$ and interprets this as 'from state p we can perform an a-action leading to the state q'. q is then called an *a-derivation* of p. One also uses the predicates '$p \xrightarrow{a}$ ' instead of '$\exists q : p \xrightarrow{a} q$' and '$p \not\xrightarrow{a}$ ' instead of '$\neg\exists q : p \xrightarrow{a} q$'.

In this report we will often depict labelled transition systems as directed graphs with labelled edges. Then the nodes correspond to the states and the edges, being labelled with actions, correspond to the transitions.

When modelling the behaviour of processes, we identify the state of a process with the process itself. Γ is then the set of processes Pr and we refer to the labelled transition system as a *process system* $\mathcal{P} = (Pr, Act, \rightarrow)$. One then gives the operational semantics of a process language by exhibiting a process system where the processes are the process expressions of the language and the transition relation is defined as the least fixed point of a set of rules and axioms called operational rules.

For MPL specification we will need a special kind of labelled transition system. As revealed in Chapter 1, MPL specifications impose restrictions on the possible implementations by telling which transitions possible implementations *must* have and which transitions they *may* have. Firstly, this means that the action set in the new transition system must contain the actions in the process system considered. Secondly, it means that we need *two* transition relations representing the different kinds of transitions: $\rightarrow_\Box$ describing the *required* transitions and $\rightarrow_\Diamond$ describing the *allowed* transitions. This is formalized as

Definition 2.2 *A* modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *is a quadruple where Sp is the set of* specifications*, Act is the set of* actions *and $\rightarrow_\Diamond$ and $\rightarrow_\Box$ are the* modal transition relations *satisfying*

$$\rightarrow_\Box \ \subseteq Sp \times Act \times Sp$$

$$\rightarrow_\Diamond \ \subseteq Sp \times Act \times Sp$$

$$\rightarrow_\Box \ \subseteq \ \rightarrow_\Diamond$$

The last condition reflects the intuition that all the transitions that an implementation *must* have are also transitions that it *may* have.

As for 'ordinary' labelled transition systems we will use the notations $S \xrightarrow{a}_\Box T$ and $S \xrightarrow{a}_\Diamond T$, interpreted in the obvious way, and talk of T as an $a_\Box$- (or $a_\Diamond$-) derivation of S. Often we will also need to talk about the transition relations independently of their modalities. We then use an index for the modalities, writing

$$S \xrightarrow{a}_m T, \quad m \in \{\Box, \Diamond\}$$

When comparing processes and specifications, as will be done in the following, it is essential to note that processes can be seen as 'fully determined' specifications, so that we for any process system $\mathcal{P} = (Pr, Act, \rightarrow)$ can define a derived modal transition system $\mathcal{S}_{\mathcal{P}} = (Pr, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ where $\rightarrow_\Box = \rightarrow_\Diamond = \rightarrow$. Thus anything allowed is also required - no looseness.

In the development of the theory of $\lhd$ we shall at times impose a restriction on the modal transition systems considered, namely that they are image-finite. This means that any specification has at most finitely many derivations for any action. Since we have the restriction $\rightarrow_\Box \ \subseteq \ \rightarrow_\Diamond$ this amounts to

Definition 2.3 $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *is* image-finite *iff the set of derivations*

$$\{T \in Sp \mid S \xrightarrow{a}_\Diamond T\}$$

is finite for all $S \in Sp$ and $a \in Act$.

2.2 Bisimulations and refinements

The refinement ordering $\lhd$ is a slight variation on the theme of bisimulation equivalence, a notion we will also need in our theoretical development. The concept of bisimulation was invented by David Park; longer introductions to bisimulation than the one given here can be found in [Milner 83] and [Larsen 86].

2.2.1 Bisimulation equivalence

In $\mathcal{CCS}$ the concept of bisimulation equivalence is *the* way of formalizing the idea of 'having the same operational behaviour as'. A bisimulation is a binary relation on a process system expressing that any pair of processes in the relation can 'simulate each other' in the following way:

Definition 2.4 *A relation $R \subseteq Pr \times Pr$ on process system $\mathcal{P} = (Pr, Act, \rightarrow)$ is a* bisimulation *iff, whenever pRq, the following holds:*

1. $p \xrightarrow{a} p' \Rightarrow \exists q' : q \xrightarrow{a} q' \wedge p'Rq'$
2. $q \xrightarrow{a} q' \Rightarrow \exists p' : p \xrightarrow{a} p' \wedge p'Rq'$

One way of defining bisimulation equivalence is now that processes p and q are bisimulation equivalent iff there exists a bisimulation such that pRq. It is easy to see that there is a largest bisimulation, for inherent in the above definition is a functional $\mathcal{B}$:

Definition 2.5 *The functional* $\mathcal{B} : \mathbf{2}^{Pr \times Pr} \rightarrow \mathbf{2}^{Pr \times Pr}$ *is defined by*
$(p, q) \in \mathcal{B}(R)$ *iff*

1. $p \xrightarrow{a} p' \Rightarrow \exists q' : q \xrightarrow{a} q' \wedge p'Rq'$
2. $q \xrightarrow{a} q' \Rightarrow \exists p' : p \xrightarrow{a} p' \wedge p'Rq'$

It is now easy to see that the following holds:

Proposition 2.1 *R is a bisimulation iff* $R \subseteq \mathcal{B}(R)$

meaning that the bisimulations are exactly the post-fixed points of $\mathcal{B}$. $\mathcal{B}$ is easily seen to be a monotonic functional on the Boolean lattice $(\mathbf{2}^{Pr}, \subseteq)$ so that $\mathcal{B}$ by Tarski's fixed point theorem has a maximal fixed point given by the union of the post-fixed points. We can thus define the maximal bisimulation $\sim$ as

$$\sim = \bigcup \{R \mid R \subseteq \mathcal{B}(R)\}$$

Then p and q are bisimulation equivalent according to the previous definition iff $p \sim q$. This gives us as a corollary the definition of $\sim$ that many texts start out with:

Corollary 2.1 *The relation* $\sim \subseteq Pr \times Pr$ *on process system* $\mathcal{P} = (Pr, Act, \rightarrow)$ *satisfies*
$p \sim q$ *iff*

1. $p \xrightarrow{a} p' \Rightarrow \exists q' : q \xrightarrow{a} q' \wedge p' \sim q'$
2. $q \xrightarrow{a} q' \Rightarrow \exists p' : p \xrightarrow{a} p' \wedge p' \sim q'$

If one now wants to prove that $p \sim q$, we see by this short discussion that all one needs is to exhibit a bisimulation R in the sense of Definition 2.4 such that pRq. The proof of R being a bisimulation consists in showing that the closure property of Proposition 2.1 holds.

This proof technique is sometimes called Park's Induction Principle. In what follows we will define our refinement orderings using the ideas sketched above, namely defining the orderings as maximal fixed points of functionals on relations, and use Park's Induction Principle accordingly. Let us here show its use by showing that $\sim$ is indeed an equivalence:

Proposition 2.2 $\sim$ *is an equivalence relation on* Pr.

PROOF: $\sim$ is *reflexive*, since $\{(p,p) \mid p \in Pr\}$ is a bisimulation. $\sim$ is *symmetric*, since the transpose of a bisimulation R defined by

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

is again a bisimulation. $\sim$ is *transitive*, for if we define composition of relations by

$$R_1 \circ R_2 = \{(p,q) \mid \exists q_1 : (p,q_1) \in R_1 \wedge (q_1,q) \in R_2\}$$

we see that $R_1 \circ R_2$ is a bisimulation when R_1 and R_2 are. □

Example 2.1 Let a process system be given by the transition diagram in Figure 2.1:

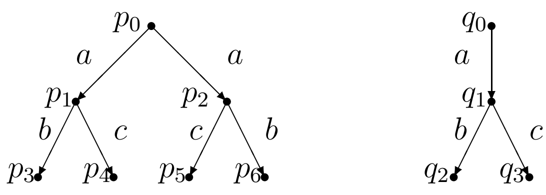

Figure 2.1: Processes p_0 and q_0 satisfy $p_0 \sim q_0$

Then p_0 and q_0 are bisimulation equivalent, for $(p_0, q_0) \in R$ where

$$R = \{(p_0,q_0),(p_1,q_1),(p_3,q_2),(p_4,q_3),(p_2,q_1),(p_5,q_3),(p_6,q_2)\}$$

is a bisimulation.

□

2.2.2 Refinements

How would one define one MPL specification S as being more refined than another specification T ? Intuitively speaking one might say that all transitions *allowed* by S must also be allowed by T, the result of matching transitions being also in the refinement relation. And conversely, all transitions *required* by T must also be required by S, the result of matching transitions being in the refinement relation. Following [Larsen and Thomsen 88] we will define [1] the concept of *refinement*:

Definition 2.6 *A relation* $R \subseteq Sp \times Sp$ *on a modal transition system* $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *is a* refinement *iff whenever* SRT *the following holds:*

1. $S \xrightarrow{a}_\Diamond S' \Rightarrow \exists T' : T \xrightarrow{a}_\Diamond T' \wedge S'RT'$
2. $T \xrightarrow{a}_\Box T' \Rightarrow \exists S' : S \xrightarrow{a}_\Box S' \wedge S'RT'$

The functional $\mathcal{RE}$ inherent in this definition is

Definition 2.7 *The functional* $\mathcal{RE} : \mathbf{2}^{Sp \times Sp} \rightarrow \mathbf{2}^{Sp \times Sp}$ *is defined by* $(S, T) \in \mathcal{RE}(R)$ *iff*

1. $S \xrightarrow{a}_\Diamond S' \Rightarrow \exists T' : T \xrightarrow{a}_\Diamond T' \wedge S'RT'$
2. $T \xrightarrow{a}_\Box T' \Rightarrow \exists S' : S \xrightarrow{a}_\Box S' \wedge S'RT'$

Again it is easy to see that we are dealing with a monotonic functional, this time over the Boolean lattice $(\mathbf{2}^{Sp \times Sp}, \subseteq)$ and that this implies that a largest refinement, $\lhd$, exists, being the union of all refinements. We then have

Corollary 2.2 *The relation* $\lhd \ \subseteq Sp \times Sp$ *on a modal transition system* $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *satisfies*
$S \lhd T$ *iff*

1. $S \xrightarrow{a}_\Diamond S' \Rightarrow \exists T' : T \xrightarrow{a}_\Diamond T' \wedge S' \lhd T'$
2. $T \xrightarrow{a}_\Box T' \Rightarrow \exists S' : S \xrightarrow{a}_\Box S' \wedge S' \lhd T'$

$\lhd$ is the refinement relation we will investigate the properties of from now on. One should note that whenever $\rightarrow_\Box = \rightarrow_\Diamond$ (e.g. when the modal transition system is derived from a process system) $\lhd$ reduces to $\sim$ in the derived 'ordinary' labelled transition system.

$\lhd$ is the desired preorder on specifications:

Proposition 2.3 $\lhd$ *is a preorder on Sp.*

[1]...or refine??

PROOF: $\lhd$ is *reflexive*, since $\{(S,S) \,|\, S \in Sp\}$ is a refinement. $\lhd$ is *transitive*, since composition of refinements (defined as in the proof of Proposition 2.2) again yields a refinement. $\square$

In this report we also need an *equivalence* on specifications. This we do by simply defining the bisimulation equivalence on specifications, $\simeq$:

Definition 2.8 *Let $\simeq$ be the maximal relation $\simeq \subseteq Sp \times Sp$ on a modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ satisfying*
$S \simeq T$ iff

1. $S \xrightarrow{a}_m S' \Rightarrow \exists T' : T \xrightarrow{a}_m T' \wedge S' \simeq T'$
2. $T \xrightarrow{a}_m T' \Rightarrow \exists S' : S \xrightarrow{a}_m S' \wedge S' \simeq T'$

where $m \in \{\Box, \Diamond\}$.

By the previous discussion it is obvious that such a relation exists. $\simeq$ is related to $\lhd$ in the obvious way:

Proposition 2.4

$$S \simeq T \Rightarrow S \lhd T \wedge T \lhd S$$

PROOF: It is easy to see that both $\{(S,T) \mid S \simeq T\}$ and its transpose are refinements. $\square$

One might ask why we did not simply use the equivalence $\bowtie$ induced by $\lhd$, defined by $S \bowtie T \Leftrightarrow S \lhd T \wedge T \lhd S$. The reason is that this definition does not guarantee that matching transitions w.r.t. $\lhd$ yield specifications that are equivalent under $\bowtie$.

An alternative characterization of $\lhd$, namely as the limit of a decreasing chain of approximating relations is needed in Chapter 4. When the functional $\mathcal{RE}$ is anti-continuous we can represent the maximal fixed point $\lhd$ as the limit of such a chain:

Definition 2.9 *A monotonic functional $f : A \rightarrow A$ on a complete lattice is anti-continuous iff we for all decreasing chains $\{B_i\}$ have that*

$$f(\sqcap_{i=1}^{\infty}(B_i)) = \sqcap_{i=1}^{\infty} f(B_i)$$

Lemma 2.1 *Let A be a complete lattice with a maximal element, $\top$, and let the functional $f : A \rightarrow A$ be anti-continuous. Then f has a maximal fixed point given by*

$$\text{FIX } f = \sqcap_{i=0}^{\infty} f^i(\top)$$

PROOF: See Appendix. □

A sufficient condition for anti-continuity of $\mathcal{RE}$ is image-finiteness:

Theorem 2.1 *If the modal transition system* $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *is image-finite, the refinement function* $\mathcal{RE}$ *is anti-continuous.*

PROOF: See Appendix. □

Corollary 2.3 *When the modal transition system* $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *is image-finite, we have that*

$$\lhd\ = \bigcap_{i=0}^{\infty} \mathcal{RE}^i(Sp^2)$$

Example 2.2 The least informative specification w.r.t. $\lhd$ is the 'wild' specification that allows all transitions, requiring none. This 'unspecified' specification $\mathcal{U}$, which we will encounter often in this report, is completely determined by $\mathcal{U} \xrightarrow{a}_\Diamond \mathcal{U}$ for all $a \in Act$ (i.e. $\mathcal{U}$ has no $\Box$-transitions). We have

$$S \lhd \mathcal{U}$$

for all $S \in Sp$, since $\{(S,\mathcal{U}) \mid S \in Sp\}$ is a refinement. This holds, since $S \xrightarrow{a}_\Diamond S'$ implies that $\mathcal{U} \xrightarrow{a}_\Diamond \mathcal{U}$ with $(S',\mathcal{U})$ still in the relation. The second condition in Definition 2.6 is vacuously satisfied. □

Example 2.3 (From [Larsen and Thomsen 88])

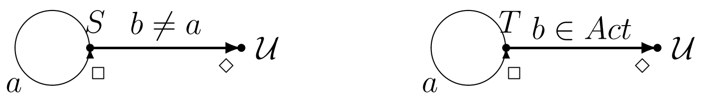

Figure 2.2: Modal transition diagrams for a-sender and -transmitter

An *a-sender* is any process which will never refuse to perform an a-action as long as nothing else is attempted. Using the $\mathcal{U}$ -specification defined in the previous example, we can give a modal transition system specifying the behaviour of any a-sender (Figure 2.2). $S \xrightarrow{a}_\Box S'$ expresses that any a-sender after an a-action still must be an a-sender. $S \xrightarrow{b}_\Diamond \mathcal{U}$, for $b \neq a$, expresses that if any other action is tried, the behaviour of the a-sender is unspecified.

An *a-transmitter* is any process allowing an infinite series of a-transitions. We can specify the a-transmitters by the modal transition system (also in Figure 2.2). Here $T \xrightarrow{a}_{\Box} T$ says that a transmitter is still a transmitter after some a-action. On the other hand, $T \xrightarrow{b}_{\Diamond} \mathcal{U}$ describes that an a-transmitter is allowed to leave the series of a-transition whenever it feels like it, then being unspecified. Intuitively, the concept of being an a-sender is a refinement of the concept of being an a-transmitter. This is reflected by $\lhd$, since we have

$$S \lhd T$$

following from the fact that $\{(S,T),(\mathcal{U},\mathcal{U})\}$ is a refinement. $\Box$

As mentioned earlier, we can regard processes as fully determined specifications by using the derived modal transition system $\mathcal{S}_{\mathcal{P}} = (Pr, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$. We can then compare them with specifications under $\lhd$ by forming the disjoint union with the modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$, allowing us to write

$$p \lhd F$$

we will then say that *p is an implementation of F.*

Example 2.3 (continued) Take a look at the process diagrams in Figure 2.3 below:

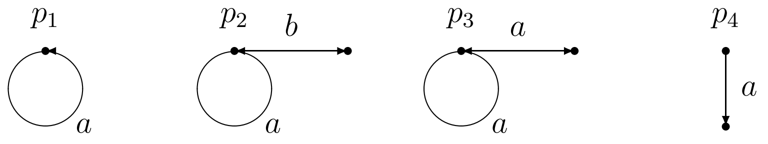

Figure 2.3: How many of these processes are a-senders or a-transmitters ?

Some of the processes are implementations of the specifications just considered. It is easy to see that p_1 and p_2 are implementations of the a-sender specification, since $\{(p_1,S)\}$ and $\{(p_2,S),(nil,\mathcal{U})\}$ are refinements. By the transitivity of $\lhd$, it is clear that these processes are implementations of the a-transmitter as well. On the other hand, p_3 and p_4 are *not* a-senders; p_3 is an a-transmitter, though, since $\{(p_3,T),(nil,\mathcal{U})\}$ is a refinement. $\Box$

Example 2.4 Consider a general medium which receives information, later releasing it. The medium is faulty, but honest - i.e., the medium may lose information but then the user of the medium is informed of the loss.

If we here ignore the information content, talking only about the presence or absence of information in terms of *pulses*, we can model the medium by a process where reception of a pulse is modelled by a c-action and emission of a pulse is modelled by an $\overline{e}$-action. The medium informs its user of the loss of a pulse by performing a $\overline{d}$-action. How can we specify the class of such processes ?

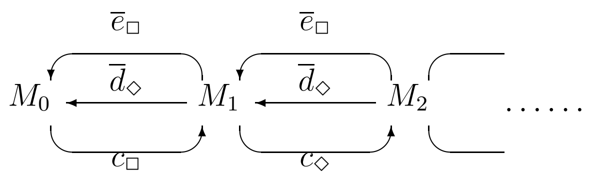

Figure 2.4: Modal transition diagram specifying a general medium.

A medium can be characterized by the number of pulses it is able to contain at the same time, referred to as its *capacity.* The state of the medium can be characterized by the number of pulses it is currently holding. A necessary requirement of a medium is that it *must* be able to take in at least one pulse and release it. Apart from that, a specification should not restrict the capacity of the medium, for one may want a medium process with a variable capacity.

Let us call a medium holding n pulses M_n. Then we expect $M_0 \xrightarrow{c}_\Box M_1$, since a medium *must* be able to take in information. For $n > 0$ we only say that $M_n \xrightarrow{c}_\Diamond M_{n+1}$. However we demand $M_n \xrightarrow{e}_\Box M_{n-1}$ for $n > 0$, since a medium must always be able to output its information. The idea of being faulty, but honest is of course not a requirement of a medium, so we specify that $M_n \xrightarrow{\overline{d}}_\Diamond M_{n-1}$ for $n > 0$. This yields the modal transition diagram in Figure 2.4. $\Box$

Chapter 3

Languages for implementations and specifications

In this chapter we present the syntax and structural operational semantics for a regular MPL and a language for defining processes, a subset of $\mathcal{CCS}$.

3.1 The process language

The implementations we consider form the class Pr of regular $\mathcal{CCS}$ -processes without value-passing [Milner 82]. These behaviours correspond to the class of finite automata.

3.1.1 Syntax of the process language

The language describing Pr contains the dynamic process constructs of $\mathcal{CCS}$ (action prefixing, nondeterminism, and recursion) and has the abstract syntax

$$p ::= nil \mid a.p_1 \mid p_1 + p_2 \mid x \mid \mu x.p$$

In this report we shall follow the convention of using lowercase letters p, q etc. to represent process expressions. A notational abbreviation which we will also adapt is writing $\sum_{i=1}^{n} p_i$ instead of $p_1 + p_2 + \cdots p_n$. [1]

Variables in the expressions should be thought of as 'templates' which may be replaced by expressions through syntactic substitution. The intuition behind the μ-operator is the usual in recursion, namely that of *unfolding*:

[1]This notation, also used in our treatment of MPL, is justified by the commutativity and associativity of $+$ under our operational semantics.

All free occurrences of the recursion variable are expanded as the recursive expression.

The formal definitions of the concepts of free variable and syntactic substitution are similar to those found in the λ-calculus [Stoy 77] since μ is just another variable-binding operator (and the only such):

Definition 3.1 *The set of free variables in an expression is given by the function $FV_{Pr} : Pr \longrightarrow \mathbf{2}^{Var}$ defined structurally on Pr by*

$$\begin{array}{rcl} FV_{Pr}(nil) & = & \emptyset \\ FV_{Pr}(a.p) & = & FV_{Pr}(p) \\ FV_{Pr}(p_1 + p_a) & = & FV_{Pr}(p_1) \cup FV_{Pr}(p_2) \\ FV_{Pr}(x) & = & \{x\} \\ FV_{Pr}(\mu x.p) & = & FV_{Pr}(p) \setminus \{x\} \end{array}$$

A variable z occurring in a process expression p such that $z \notin FV_{Pr}(p)$, is called bound. *An expression containing only bound variables is called* closed.

Substituting the process expression q for all free occurrences of variable x in process expression p is denoted by $p[q/x]$ and defined as follows:

Definition 3.2 (Substitution in the process language)

The substitution function $p[q/x] : Pr \times Pr \times Var \longrightarrow Pr$ is defined structurally on p by

$$\begin{array}{rcl} nil[q/x] & = & nil \\ (a.p_1)[q/x] & = & a.(p_1[q/x]) \\ (p_1 + p_2)[q/x] & = & p_1[q/x] + p_2[q/x] \\ y[q/x] & = & \begin{cases} y & \textit{if } y \neq x \\ q & \textit{if } y = x \end{cases} \\ (\mu y.p_1)[q/x] & = & \begin{cases} \mu y.p_1 & \textit{if } y = x \\ \mu y.(p_1[q/x]) & \textit{if } y \neq x \textit{ and } y \notin FV_{Pr}(q) \\ \mu z.((p_1[z/y])[q/x]) & \textit{if } y \neq x \textit{ and } y \in FV_{Pr}(q) \end{cases} \end{array}$$

where z is a variable such that $z \neq x$, $z \notin FV_{Pr}(q)$ and $z \notin FV_{Pr}(\mu y.p_1)$ and '$=$' denotes syntactic equality up to renaming of bound variables.

3.2 Operational semantics for the process language

We can now give the operational semantics of our process language as a process system whose processes are the process expressions Pr just defined.

Definition 3.3 (Operational semantics for process language) *In the process system $\mathcal{P} = (Pr, Act, \rightarrow)$ the transition relation $\rightarrow$ is defined as the smallest (w.r.t. $\subseteq$) relation, $\rightarrow \subseteq Pr \times Act \times Pr$ satisfying:*

Action prefixing

$$a.p \xrightarrow{a} p$$

Nondeterminism

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \quad \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'}$$

Recursion

$$\frac{p[\mu x.p/x] \xrightarrow{a} p'}{\mu x.p \xrightarrow{a} p'}$$

3.3 The regular MPL

We here only consider a regular Modal Process Logic, i.e. one with the dynamic process constructs mentioned in the previous section. In Chapter 5 we introduce static constructs, and the operational semantics will be extended accordingly. The set of regular MPL expressions will be called Sp.

3.3.1 Syntax of MPL

The MPL language also has the dynamic process constructs. The abstract syntax is here

$$E ::= Nil \mid a_{\Box}.E_1 \mid a_{\Diamond}.E_1 \mid E_1 + E_2 \mid x \mid \mathrm{rec}x.E$$

As one can see, the syntax is almost the same as that of the process language. To avoid confusion, though, we write '*Nil* ' and 'rec' when referring to inaction and recursion. MPL specifications[2] will always be denoted by

[2]We will use the phrases 'MPL specification', 'MPL expression' and 'MPL formula' interchangeably

uppercase letters, and the summation convention of writing $\sum_{i=1}^{n} E_i$ instead of $E_1 + E_2 \cdots E_n$ is also applied.

The concept of free variables extends easily into MPL:

Definition 3.4 *The set of free variables in an MPL expression is given by the function $FV_{Sp} : Sp \rightarrow \mathbf{2}^{Var}$ defined structurally on Sp by*

$$\begin{array}{rcl}
FV_{Sp}(Nil) & = & \emptyset \\
FV_{Sp}(a_{\Box}.E) & = & FV_{Sp}(E) \\
FV_{Sp}(a_{\Diamond}.E) & = & FV_{Sp}(E) \\
FV_{Sp}(E_1 + E_2) & = & FV_{Sp}(E_1) \cup FV_{Sp}(E_2) \\
FV_{Sp}(x) & = & \{x\} \\
FV_{Sp}(\mathrm{rec}x.E) & = & FV_{Sp}(E) \setminus \{x\}
\end{array}$$

As for process expressions, a variable z occurring in an MPL expression E such that $z \notin FV_{Sp}(E)$, is called bound. *An expression containing only bound variables is called* closed.

Substituting the MPL expression F for all free occurrences of variable x in process expression E is denoted by $E[F/x]$ and defined as follows:

Definition 3.5 (Substitution in MPL)

The substitution function $E[F/x] : Sp \times Sp \times Var \rightarrow Sp$ is defined structurally on E by

$$\begin{array}{rcl}
Nil[F/x] & = & Nil \\
(a_{\Box}.E_1)[F/x] & = & a_{\Box}.(E_1[F/x]) \\
(a_{\Diamond}.E_1)[F/x] & = & a_{\Diamond}.(E_1[F/x]) \\
(E_1 + E_2)[F/x] & = & E_1[F/x] + E_2[F/x] \\
y[F/x] & = & \begin{cases} y & \textit{if } y \neq x \\ F & \textit{if } y = x \end{cases} \\
(\mathrm{rec}y.E_1)[F/x] & = & \begin{cases} \mathrm{rec}y.E_1 & \textit{if } y = x \\ \mathrm{rec}y.(E_1[F/x]) & \textit{if } y \neq x \textit{ and } y \notin FV_{Sp}(F) \\ \mathrm{rec}z.((E_1[z/y])[F/x]) & \textit{if } y \neq x \textit{ and } y \in FV_{Sp}(F) \end{cases}
\end{array}$$

where (as before) z is a variable such that $z \neq x$, $z \notin FV_{Sp}(F)$ and $z \notin FV_{Sp}(\mathrm{rec}y.E_1)$ and '$=$' denotes syntactic equality up to renaming of bound variables.

3.3.2 Operational semantics of MPL

The operational semantics of MPL only differs from the process semantics in that we now use modal transition systems. The idea is that $\Box$ -prefixing yields $\Box$ -transitions and $\Diamond$ -prefixing yields $\Diamond$ -transitions.

Definition 3.6 (Operational semantics for MPL) *In the modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ the transition relations $\rightarrow_\Box$ and $\rightarrow_\Diamond$ are defined as the smallest (w.r.t. $\subseteq$) pair of relations $\rightarrow_m \subseteq Pr \times Act \times Pr$, $m \in \{\Box, \Diamond\}$ satisfying:*

$\Diamond$ -prefixing

$$a_\Diamond.E \xrightarrow{a}_\Diamond E$$

$\Box$ -prefixing

$$a_\Box.E \xrightarrow{a}_\Box E$$

$$a_\Box.E \xrightarrow{a}_\Diamond E$$

Nondeterminism

$$\frac{E \xrightarrow{a}_m E'}{E + F \xrightarrow{a}_m E'} \qquad \frac{F \xrightarrow{a}_m F'}{E + F \xrightarrow{a}_m F'}$$

Recursion

$$\frac{E[\mathrm{rec}x.E/x] \xrightarrow{a}_m E'}{\mathrm{rec}x.E \xrightarrow{a}_m E'}$$

where $m \in \{\Box, \Diamond\}$.

The above rules truly define a modal transition system:

Proposition 3.1 *The relations $\rightarrow_\Box$ and $\rightarrow_\Diamond$ in Definition 3.6 satisfy*

$$\rightarrow_\Box \subseteq \rightarrow_\Diamond$$

Proof: A straightforward induction in the length of the inference used to show that $E \xrightarrow{a}_\Box E'$. $\Box$

We shall often restrict our attention to MPL expressions that are well-guarded, since this (natural) restriction gives us a pleasant characterization of the transitions of an MPL specification.

Definition 3.7 *The set of* unguarded *variables in an MPL expression is given by the function $UG_{Sp} : Sp \to Var$ defined structurally by:*

$$\begin{array}{rcl} UG_{Sp}(Nil) & = & \emptyset \\ UG_{Sp}(a_{\Box}.E) & = & \emptyset \\ UG_{Sp}(a_{\Diamond}.E) & = & \emptyset \\ UG_{Sp}(E_1 + E_2) & = & UG_{Sp}(E_1) \cup UG_{Sp}(E_2) \\ UG_{Sp}(x) & = & \{x\} \\ UG_{Sp}(\mathrm{rec}x.E) & = & UG_{Sp}(E) \setminus \{x\} \end{array}$$

If we for a variable x in an expression E have $x \notin UG(E)$ we say that x is guarded *in E. If $UG(E) = \emptyset$ (i.e. all variables in E are guarded), we say that E is* well-guarded.

Well-guardedness means that we cannot access the transitions made possible by the presence of variable without first performing a transition obtained from the prefixing of an action. This is indicated in the following, important

Lemma 3.1 *Whenever x is guarded in $G \in Sp$ we have*

$$G[F/x] \xrightarrow{a}_m E$$

if and only if

$$G \xrightarrow{a}_m G' \text{ and } E = G'[F/x]$$

for some G' (where $m \in \{\Diamond, \Box\}$)

Proof: Induction in the length of inferences establishing $G[F/x] \xrightarrow{a}_m E$ and $G \xrightarrow{a}_m G'$. For details, see Appendix. $\Box$

As a corollary we can characterize rec:

Corollary 3.1 *Whenever x is guarded in G*

$$\mathrm{rec}x.G \xrightarrow{a}_m E$$

if and only if

$$G \xrightarrow{a}_m G' \text{ and } E = G'[\mathrm{rec}x.G/x]$$

for some G' (where $m \in \{\Diamond, \Box\}$)

Example 3.1 Assuming that the action set *Act* is finite, the specification $\mathcal{U}$ introduced in Chapter 2 can be written as

$$\mathcal{U} = \text{rec}x.(\sum_{a \in Act} a_{\Diamond}.x)$$

For we have $\mathcal{U} \xrightarrow{b}_m E$ iff $(\sum_{a \in Act} a_{\Diamond}.x)[\mathcal{U}/x] \xrightarrow{b}_m E$, which by the substitution function, Corollary 3.1 and the operational rules for + and prefixing gives us that $\mathcal{U} \xrightarrow{b}_{\Diamond} \mathcal{U}$ for all $b \in Act$. □

Example 3.2 Again under the assumption that *Act* is finite, the a-sender introduced in Chapter 2, Example 2.3 can be written as

$$S = \text{rec}x.(a_{\Box}.x + \sum_{b \neq a} b_{\Diamond}.\mathcal{U})$$

and the a-transmitter also from this example can be expressed as

$$T = \text{rec}x.(a_{\Box}.x + \mathcal{U})$$

□

The above definitions of guardedness can of course also be given for the process language in an entirely similar way. We also get identical properties to the ones described for MPL (except for the modalities on the transitions) for process expressions. We state the following without proof, since this is entirely analogous to the one for Lemma 3.1:

Lemma 3.2 *Whenever x is guarded in $p \in Pr$ we have*

$$p[q/x] \xrightarrow{a} r$$

iff

$$p \xrightarrow{a} p' \text{ and } r = p'[q/x]$$

for some p'.

Corollary 3.2 *Whenever x is guarded in p*

$$\mu x.p \xrightarrow{a} q$$

iff

$$p \xrightarrow{a} p' \text{ and } q = p'[\mu x.p/x]$$

for some p'.

Chapter 4

Full abstraction in the modal process logic

In this chapter we present a denotational description of MPL, first given in [Larsen and Thomsen 88]. We compare this description with the denotational description of the STL tree logic in [Graf and Sifakis 86]. We also discuss the result presented in [Larsen and Thomsen 88] on full abstractness of the denotational semantics, how well the result applies to the partial specification language described in [Larsen and Thomsen 87] and its limitations.

4.1 Denotational semantics of MPL

The denotational description of MPL is denotational in the sense that the meaning assigned to an expression is a function of the meanings of its immediate constituents. However, it does not involve domain theory, only simple set theory. On the other hand, the description assumes the existence of a process system, since the meaning of an MPL formula is *a set of processes*. In this way the description is similar to the denotational descriptions of Hennessy-Milner logic [Hennessy and Milner 85] and the STL in [Graf and Sifakis 86].

In [Graf and Sifakis 86] Graf and Sifakis require that a logic and its description be *adequate* in the following sense:

- The meaning of a formula in the logic must be a union of equivalence classes of processes, so that equivalent processes satisfy the same formulae.
- Any process can be described by a formula in the logic, so that we for each equivalence class have a formula representing it.

With respect to these requirements the semantic description of MPL is adequate, as we shall see.

4.1.1 Semantic definitions

Our semantic algebra contains functions corresponding to process constructs in MPL:

Definition 4.1 (Semantic algebra)
The semantic algebra consists of the functions

$$\begin{aligned} \mathrm{nil} :&\ \_ \rightarrow \mathbf{2}^{Pr} \\ \lceil \_\diamond \rceil :&\ \mathit{Act} \times \mathbf{2}^{Pr} \rightarrow \mathbf{2}^{Pr} \\ \lceil \_\Box \rceil :&\ \mathit{Act} \times \mathbf{2}^{Pr} \rightarrow \mathbf{2}^{Pr} \\ \_ \oplus \_ :&\ \mathbf{2}^{Pr} \times \mathbf{2}^{Pr} \rightarrow \mathbf{2}^{Pr} \end{aligned}$$

defined by

$$\begin{aligned} \mathrm{nil} &= \{p \mid p \sim \mathit{nil}\} \\ \lceil a_\diamond \rceil U &= \{p \mid p \xrightarrow{b} p' \Rightarrow (b = a \wedge p' \in U)\} \\ \lceil a_\Box \rceil U &= \{p \mid p \xrightarrow{a} \ \wedge\ p \in \lceil a_\diamond \rceil U\} \\ U \oplus V &= \{p \mid \exists p_1, p_2 . p_1 \in U \ \wedge\ p_2 \in V \ \wedge\ p \sim p_1 + p_2\} \end{aligned}$$

where $U, V \subseteq Pr$.

In our further theoretical development it is important that these functions are sufficiently nice:

Lemma 4.1 *The functions of Definition 4.1 are monotonic and anticontinuous on the Boolean lattice* $(\mathbf{2}^{Pr}, \subseteq)$

PROOF: Monotonicity is obvious. For the proof of anticontinuity, see the Appendix. □

Since MPL expressions may contain variables, their denotations must depend on an *environment*:

Definition 4.2 *An environment* σ *is an element of the function space* $\boldsymbol{E} = Var \rightarrow \mathbf{2}^{Pr}$

We can now give the semantic equations:

Definition 4.3 *(Denotational semantics for MPL) Define for $F \in Sp$ the function $[\![\_]\!] : Sp \to (\boldsymbol{E} \to \mathbf{2}^{Pr})$ inductively by:*

$$\begin{aligned}
[\![Nil]\!]\sigma &= \text{nil} \\
[\![x]\!]\sigma &= \sigma(x) \\
[\![a_{\Diamond}.F]\!]\sigma &= \lceil a_{\Diamond} \rceil([\![F]\!]\sigma) \\
[\![a_{\Box}.F]\!]\sigma &= \lceil a_{\Box} \rceil([\![F]\!]\sigma) \\
[\![F + G]\!]\sigma &= [\![F]\!]\sigma \oplus [\![G]\!]\sigma \\
[\![\text{rec}x.F]\!]\sigma &= \text{FIX } U.([\![F]\!]\sigma\{U/x\})
\end{aligned}$$

where $\sigma \in \boldsymbol{E}$ and FIX *is the* maximal *fixed point operator.*

FIX is well-defined here, since $[\![\_]\!]$ is a monotonic function on the Boolean lattice $(\mathbf{2}^{Pr}, \subseteq)$ due to the monotonicity of the semantic algebra functions.

4.1.2 Adequacy of the semantics

The first part of the adequacy requirement stated earlier, namely that the denotation of an MPL formula is a union of equivalence classes, falls right into our laps. The result, of course, requires that the environment σ demands this property of the variables. (This seems like a very reasonable requirement.)

Definition 4.4 *A set $U \subseteq Pr$ is said to be cue (closed under equivalence) iff*

$$p \in U \ \wedge \ q \sim p \Rightarrow q \in U$$

It is obvious that we have that if U is *cue*, it is the union of some equivalence classes, i.e. $U = \bigcup_{p \in S} |p|$ for some set $S \subseteq Pr$. (Here $|p|$ denotes the equivalence class of p w.r.t. to $\sim$).

The property of being *cue* is preserved by union and intersection since any two equivalence classes are either identical or disjoint. An environment is called *cue* iff its images are *cue*:

Definition 4.5 *An environment σ is cue iff we for all $x \in Var$ have that $\sigma(x)$ is cue.*

Theorem 4.1 *If $\sigma \in \boldsymbol{E}$ is cue, we have for any $F \in Sp$ that $[\![F]\!]\sigma$ is cue.*

Proof: Structural induction on F.

$F = Nil$: Obvious, since $[\![Nil]\!] = \{p \mid p \sim nil\}$.

$F = x$: By assumption, σ is *cue*.

$F = F_1 + F_2$: We have

$$\begin{aligned} [\![F_1]\!]\sigma \oplus [\![F_2]\!]\sigma &= \{p \mid \exists p_1 \in [\![F_1]\!]\sigma, \exists p_2 \in [\![F_2]\!]\sigma : p \sim p_1 + p_2\} \\ &= \bigcup_{p_i \in [\![F_i]\!]} |p_1 + p_2| \end{aligned}$$

which is of the desired form

$F = a_\diamond.F_1$: We have

$$[\![a_\diamond.F_1]\!]\sigma = \{p \mid p \xrightarrow{a} p' \Rightarrow b = a \wedge p' \in [\![F_1]\!]\sigma\}$$

Assume for $p \in [\![a_\diamond.F_1]\!]$ that $q \sim p$. Then q has the same derivations as p and $p \xrightarrow{a} p'$ implies $q \xrightarrow{a} q'$ with $p' \sim q'$. By induction hypothesis we then have $q' \in [\![F_1]\!]$, again implying $q \in [\![a_\diamond.F_1]\!]$.

$F = a_\Box.F_1$: As for $a_\diamond.F_1$.

$F = \mathrm{rec}x.F_1$: Define $G : \mathbf{2}^{Pr} \to \mathbf{2}^{Pr}$ by $G(U) = [\![F_1]\!]\sigma\{U/x\}$. Then we have

$$[\![\mathrm{rec}x.F_1]\!]\sigma = \bigcap_{n=0}^{\infty} G^n(Pr)$$

If U is *cue* then, by induction hypothesis, $G(U)$ is also cue. Since Pr is clearly *cue*, and intersection preserves *cue*-ness, the result follows.

$\Box$

A nice consequence is that we have another characterization of the semantics of a $\Box$ -prefixed formula whenever our environment is *cue*:

Proposition 4.1 *When $\sigma \in \boldsymbol{E}$ is cue we have*

$$[\![a_\Box.F]\!]\sigma = \{p \mid p \sim \sum_{i\in I} a.p_i,\ I \neq \emptyset,\ p_i \in [\![F]\!]\sigma\}$$

Proof: Let $S_1 = \{p \mid p \sim \sum_{i\in I} a.p_1, p_1 \in [\![F]\!]\sigma\}$ and $S_2 = [\![a_\Box.F]\!]\sigma$. It is clear that $S_1 \subseteq S_2$. Now suppose $p \in S_2$. Then the only possible derivations are of the type $p \xrightarrow{a} p'$ with $p' \in [\![F]\!]\sigma$. By Theorem 4.1 we have a $p_i \in [\![F]\!]\sigma$ such that $p_i \sim p'$, yielding the proof. $\Box$

(This is actually how [Graf and Sifakis 86] defined the semantics of prefixing).

The second part of the adequacy requirement, that any process is in the denotation of some MPL formula, is proved in subsection 4.2.2.

4.2 Full abstractness of the denotational semantics

In [Larsen and Thomsen 88] Kim Larsen and Bent Thomsen proved that the denotational description just given was fully abstract w.r.t. $\lhd$ in that we have

Theorem 4.2 *Let $G \in Sp$ be closed and well-guarded. Then*

$$[\![G]\!] = \{p \mid p \lhd G\}$$

PROOF: See [Larsen and Thomsen 88] □

Note that this result does not depend on σ. Since $\lhd$ collapses to $\sim$ for processes, this means that we have alternative versions of Theorem 4.1 and Proposition 4.1 for *closed* MPL formulae, not depending on the *cue*-ness of σ:

Corollary 4.1 *For all closed $F \in Sp$ $[\![F]\!]$ is cue.*

Corollary 4.2 *For all closed $F \in Sp$*

$$[\![a_\Box.F]\!]\sigma = \{p \mid p \sim \sum_{i \in I} a.p_i,\ I \neq \emptyset,\ p_i \in [\![F]\!]\sigma\}$$

Since $\lhd$ is transitive, we also have

Corollary 4.3 *Let F and G be closed and well-guarded. Then*

$$F \lhd G \Rightarrow [\![F]\!] \subseteq [\![G]\!]$$

4.2.1 Limitations of the full abstractness

A compelling problem is now if we also have the converse of Corollary 4.3, i.e.

$$[\![F]\!] \subseteq [\![G]\!] \Rightarrow F \lhd G$$

for well-guarded F, G ? Unfortunately we do not.

Example 4.1 Consider the formulae

$$\begin{aligned} F &= a_\Diamond.b_\Diamond.Nil \\ G &= a_\Diamond.b_\Box.Nil + a_\Diamond.Nil \end{aligned}$$

Then we do *not* have that $F \lhd G$, for $F \xrightarrow{a}_\Diamond b_\Diamond.Nil$ must be matched by either $G \xrightarrow{a}_\Diamond b_\Box.Nil$ or $G \xrightarrow{a}_\Diamond Nil$. But then we have, respectively, $b_\Box.Nil \xrightarrow{b}_\Box$ while $b_\Diamond.Nil \not\xrightarrow{b}_\Box$ or $b_\Diamond.Nil \xrightarrow{b}_\Diamond$ while $Nil \not\xrightarrow{b}_\Diamond$. On the other hand, since the formulae are closed we get that

$$[\![F]\!] = [\![G]\!] = \{p \mid p \sim nil \vee p \sim a.nil \vee p \sim a.b.nil \vee p \sim a.b.nil + a.nil\}$$

□

Another counterexample is

Example 4.2

$$\begin{aligned} F &= a_\Box.b_\Box.Nil + a_\Box.Nil + a_\Diamond.b_\Diamond.Nil \\ G &= a_\Box.b_\Box.Nil + a_\Box.Nil \end{aligned}$$

□

On the other hand, it turns out that we can define a relation $\ntrianglelefteq_D$ stronger than the negation of $\lhd$, $\ntrianglelefteq$, for which the desired property holds. i.e. whenever $F \ntrianglelefteq_D G$ we have $F \not\subseteq G$.

There are two ways in which $F \ntrianglelefteq G$ may happen:

- Either F or G has a transition the other does not have.
- One specification has a transition that the other can only possibly match in a way leading to specifications not related by $\lhd$.

It is easily seen that we for $\ntrianglelefteq$ have

Proposition 4.2 *The relation $\ntrianglelefteq \subseteq Sp \times Sp$ on a modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ is the least relation satisfying*

$S \ntrianglelefteq T$ if one of these two conditions holds:

1. $\exists a \exists S' : S \xrightarrow{a}_\Diamond S' \wedge (T \not\xrightarrow{a}_\Diamond \vee (\forall T' : T \xrightarrow{a}_\Diamond T' \Rightarrow S' \ntrianglelefteq T'))$
2. $\exists a \exists T' \xrightarrow{a}_\Box T' \wedge (S' \not\xrightarrow{a}_\Box \vee (\forall S' : S \xrightarrow{a}_\Box S' \Rightarrow S' \ntrianglelefteq T'))$

The definition of $\ntriangleleft_D$ is somewhat stronger in that it essentially says that the conclusion that F does not refine G is based on the information that at some point one of the specifications has at most one transition of some kind and this transition leads to a mismatch. We call such a mismatch *deterministic*. Note that the mismatches in Examples 4.1 and 4.2 were not deterministic; in Example 4.1 we thus had two $a_\Diamond$-transitions each providing a mismatch (and for two different reasons).

We define the relation, $\ntriangleleft_D$, as the union of a chain of relations, $\{{\ntriangleleft_D}^m\}$, where $F{\ntriangleleft_D}^m G$ if we observe the deterministic mismatch after at most m transitions. This is inspired by [Larsen 86] in which Kim Larsen describes parameterized bisimulation through (among other things) the negation of the simulation preorder on processes, $\leq$.

Definition 4.6

$${\ntriangleleft_D}^0 = \emptyset$$

$F{\ntriangleleft_D}^n G$ iff one of the following holds

1. $\exists a \exists F' : F \xrightarrow{a}_\Diamond F' \wedge (G \overset{a}{\not\to}_\Diamond \ \vee ((\exists! G' : G \xrightarrow{a}_\Diamond G') \ \wedge$
   $\forall G' \exists m < n : G \xrightarrow{a}_\Diamond G' \Rightarrow F'{\ntriangleleft_D}^m G'))$

2. $\exists a \exists F' : G \xrightarrow{a}_\Box G' \wedge (F \overset{a}{\not\to}_\Box \ \vee ((\exists! F' : F \xrightarrow{a}_\Box F') \ \wedge$
   $\forall F' \exists m < n : F \xrightarrow{a}_\Diamond F' \Rightarrow F'{\ntriangleleft_D}^m G'))$

$$\ntriangleleft_D = \bigcup_n^\infty {\ntriangleleft_D}^n$$

$\ntriangleleft_D$ is related to $\ntriangleleft$ in the obvious way:

Proposition 4.3

$$F \ntriangleleft_D G \Rightarrow F \ntriangleleft G$$

PROOF: The theorem says that, if for some n, $F{\ntriangleleft_D}^n G$ we have that $F \ntriangleleft G$. The proof is then a trivial induction in n. $\Box$

We now claim that we have $[\![F]\!] \not\subseteq [\![G]\!] \Rightarrow F \ntriangleleft_D G$. In the proof we need to construct two *canonical implementations* for F, $p^\Box(F)$ and $p^\Diamond(F)$, preserving the $\Box$ - and $\Diamond$ -transitions of F, respectively. Then one shows that $p^\Box(F) \notin [\![G]\!]$ or $p^\Diamond(F) \notin [\![G]\!]$, respectively. The constructions of $p^\Box$ and $p^\Diamond$ are interesting in their own right; they will also be used to show that any process satisfies some specification. We therefore describe the constructions here:

Definition 4.7 *The syntactic transformation* $p^\Box : Sp \rightarrow Pr$ *is defined structurally by*

$$\begin{array}{rcl} p^\Box(Nil) & = & nil \\ p^\Box(a_\Box.E) & = & a.p^\Box(E) \\ p^\Box(a_\Diamond.E) & = & nil \\ p^\Box(E_1 + E_2) & = & p^\Box(E_1) + p^\Box(E_2) \\ p^\Box(\mathrm{rec}x.E) & = & \mu x.p^\Box(E) \\ p^\Box(x) & = & x \end{array}$$

Thus, $p^\Box$ simply leaves out all $\Diamond$ -transitions. $p^\Box(F)$ distributes w.r.t. substitution in F in the following way:

Lemma 4.2 *For all* $S, T \in Sp$*:*

$$p^\Box(S[T/x]) = p^\Box(S)[p^\Box(T)/x]$$

Proof: Structural induction on S. See Appendix. $\Box$

Lemma 4.3 *For all closed* $E \in Sp$ *we have* $p^\Box(E) \in [\![E]\!]$

Proof: By Theorem 4.2 it is enough to show that $p^\Box(E) \lhd E$. This, in turn, follows directly from

$$p^\Box(E) \xrightarrow{a} q \Leftrightarrow E \xrightarrow{a}_\Box E' \wedge q = p^\Box(E')$$

The proof is done by induction on the structure of E. See Appendix. $\Box$

Note that the proof ensures that the process constructed has exactly the $\Box$ -transitions of F.

$p^\Diamond$ is defined almost as $p^\Box$:

Definition 4.8 *The syntactic transformation* $p^\Diamond : Sp \rightarrow Pr$ *is defined structurally by*

$$\begin{array}{rcl} p^\Diamond(Nil) & = & nil \\ p^\Diamond(a_\Box.E) & = & a.p^\Diamond(E) \\ p^\Diamond(a_\Diamond.E) & = & a.p^\Diamond(E) \\ p^\Diamond(E_1 + E_2) & = & p^\Diamond(E_1) + p^\Diamond(E_2) \\ p^\Diamond(\mathrm{rec}x.E) & = & \mu x.p^\Diamond(E) \\ p^\Diamond(x) & = & x \end{array}$$

$p^{\Diamond}$ distributes over substitution exactly as $p^{\Box}$ does:

Lemma 4.4 *For all $S, T \in Sp$:*

$$p^{\Diamond}(S[T/x]) = p^{\Diamond}(S)[p^{\Diamond}(T)/x]$$

Proof: Structural induction on S. Except for $S = a_{\Diamond}.S'$, the proof is as for $p^{\Box}$. In this case the proof is as for $S = a_{\Box}.S'$ in the proof for $p^{\Box}$. □

We also have

Lemma 4.5 *For all closed $E \in Sp$ we have $p^{\Diamond}(E) \in [\![E]\!]$*

Proof: By Theorem 4.2 it is enough to show that $p^{\Box}(E) \lhd E$. This we do by showing that

$$\{(p^{\Diamond}(E), E) \mid E \in Sp\}$$

is a refinement. This in turn amounts to showing that $p^{\Diamond}(E) \xrightarrow{a} q \Rightarrow E \xrightarrow{a}_{\Diamond} E'$ with $q = p^{\Diamond}(E')$ and $E \xrightarrow{a}_{\Diamond} E' \Rightarrow p^{\Diamond}(E) \xrightarrow{a} q$ with $q = p^{\Diamond}(E')$. The proof of this follows the same lines as the proof needed in the proof of Lemma 4.3 and is therefore omitted. □

Note how the proof also showed that $p^{\Diamond}(F)$ has exactly the $\Diamond$ -transitions of F.

Now we can finally state and prove

Theorem 4.3 *For all closed $F, G \in Sp$ we have*

$$F \not\lhd_D G \Rightarrow [\![F]\!] \not\subseteq [\![G]\!]$$

Proof: We show that

$$\forall n : F {\not\lhd_D}^n G \Rightarrow [\![F]\!] \not\subseteq [\![G]\!]$$

by induction on n. We again rely heavily on the fact that $p \in [\![F]\!]$ is the same as $p \lhd F$.

$n = 0$**:** Vacuously satisfied.

Assuming for $n = k$: By the definition of ${\not\lhd_D}^k$, there are four cases to be considered:

1. $F \xrightarrow{a}_{\Diamond} F'$ but $G \not\xrightarrow{a}_{\Diamond}$: Then we have $p^{\Diamond}(F) \lhd F$ but not $p^{\Diamond}(F) \lhd G$, since $p^{\Diamond}(F)$ has exactly the $\Diamond$ -transitions of F. Hence $p^{\Diamond}(F) \notin [\![G]\!]$.

2. $F \xrightarrow{a}_{\diamond} F'$, $(\exists! G' : G \xrightarrow{a}_{\diamond} G') \wedge (G \xrightarrow{a}_{\diamond} G' \Rightarrow \exists m < k : F' \not\triangleleft_D{}^m G')$ *:* By induction hypothesis, we have a p' such that $p' \in [\![F']\!]$ and $p' \notin [\![G']\!]$. Then $a.p' + p^{\diamond}(F) \triangleleft a_{\Box}.F' + F \triangleleft F$ but $a.p' + p^{\diamond}(F) \not\triangleleft G$ implying that a distinguishing process is $a.p' + p^{\diamond}(F)$. (Another distinguishing process is $a.p' + p^{\Box}(F)$.)

3. $G \xrightarrow{a}_{\Box} G'$ *but* $F \not\xrightarrow{a}_{\Box}$ *:* Then we have $p^{\Box}(F) \not\triangleleft G$ since $p^{\Box}(F)$ has exactly the $\Box$ -transitions of F.

4. $G \xrightarrow{a}_{\Box} G'$, $(\exists! F' : F \xrightarrow{a}_{\Box} F') \wedge (F \xrightarrow{a}_{\Box} F' \Rightarrow \exists m < k : F' \not\triangleleft_D{}^m G')$ *:* By induction hypothesis we have a p' such that $p' \in [\![F]\!]$ and $p' \notin [\![G]\!]$. Then the distinguishing process is $a.p' + p^{\Box}(F)$, for $a.p' + p^{\Box}(F) \triangleleft a_{\Box}.F' + F \triangleleft F$ but $a.p' + p^{\Box}(F) \not\triangleleft G$.

This completes the proof. □

Example 4.3 Consider again the specification S of the a-sender and the specification T of the a-transmitter from Example 2.3. It is easy to see that

$$T \not\triangleleft_D{}^2 S$$

since $T \xrightarrow{a}_{\diamond} \mathcal{U}$ but the *only* match is $S \xrightarrow{a}_{\diamond} S$ and $\mathcal{U}$ has no $\Box$ -transitions while $S \xrightarrow{a}_{\Box}$. Theorem 4.3 now states that there must be a process p such that $p \triangleleft T$ but $p \not\triangleleft S$. An example of such a process is the p_3-process from Example 2.3, given by $p_3 = \mu x.(a.x + a.nil)$. In fact, $p_3 = a.nil + p^{\Box}(T)$. □

4.2.2 Adequacy revisited

The idea of canonical implementations can also be used to show the second adequacy requirement to the semantic description of MPL, i.e. that any process is satisfied by some specification, such that we for each equivalence class have a formula representing it. For, given any process p, we can construct a specification $F_{\Box}(p)$ by simply putting $\Box$'s on the actions:

Definition 4.9 *The syntactic transformation* $F_{\Box} : Pr \rightarrow Sp$ *is defined structurally by*

$$\begin{aligned}
F_{\Box}(nil) &= Nil \\
F_{\Box}(a.p) &= a_{\Box}.F_{\Box}(p) \\
F_{\Box}(p_1 + p_2) &= F_{\Box}(p_1) + F_{\Box}(p_2) \\
F_{\Box}(\mu x.p) &= \mathrm{rec}x.F_{\Box}(p) \\
F_{\Box}(x) &= x
\end{aligned}$$

It is obvious that we have that all $\Diamond$ -transitions of $F_\Box(p)$ must be due to its $\Box$ -transitions. It is also easy to see that $F_\Box$ is the right-inverse of $p^\Box$:

Proposition 4.4 *For all $p \in Pr$ we have*

$$p^\Box(F_\Box(p)) = p$$

Proof: A trivial structural induction on p. $\Box$

$F_\Box(p)$ has exactly the transitions of p:

Proposition 4.5 $F_\Box(p) \xrightarrow{a}_m F'$ *iff* $p \xrightarrow{a} p'$ *and* $F' = F_\Box(p')$*.*

Proof: Inductions in the lengths of inferences establishing $F_\Box(p) \xrightarrow{a}_m F'$ and $p \xrightarrow{a} p'$, respectively. The proof is completely analogous to that of Lemma 4.3 and is therefore omitted. $\Box$

and thus

Proposition 4.6 $p \lhd F_\Box(p)$

$F_\Box(p)$ is the specification leading to adequacy:

Theorem 4.4 *For all $p \in Pr$ there is an $F \in Sp$ such that $p \lhd F$ and $q \lhd F$ iff $q \sim p$.*

Proof: Choose $F = F_\Box(p)$. It is clear by Theorem 4.1 that if $q \sim p$ then $q \lhd F_\Box(p)$. For the if-part note that we by the previous proposition have that

$$\{(p,q) \mid q \lhd F_\Box(p)\}$$

is a bisimulation. For suppose $p \xrightarrow{a} p'$. Then $F_\Box(p) \xrightarrow{a}_m F_\Box(p')$. But then $q \xrightarrow{a} q'$ with $q' \lhd F_\Box(p')$. Suppose $q \xrightarrow{a} q'$. Then $F_\Box(p) \xrightarrow{a}_m F'$ and $q' \lhd F'$. But then $p \xrightarrow{a} p'$ with $F' = F_\Box(p')$. $\Box$

4.3 Denotational properties of partial specifications

A predecessor to MPL is the language of partial specifications, conceived by Bent Thomsen [Thomsen 87] and elaborated in [Larsen and Thomsen 87]. With the language comes an ordering on the partial specifications, also called $\lhd$. We will here show that the partial specifications form a sublanguage of MPL, that the orderings coincide, and that one can give the language a suitable denotational description such that we have

$$[\![F]\!] \subseteq [\![G]\!] \Leftrightarrow F \lhd G$$

4.3.1 The language of partial specifications

Partial specifications are thought of as process expressions with 'holes' in them; one has an extra, unspecified process $\mathcal{U}$. The presence of $\mathcal{U}$ indicates that the behaviour of the process is not specified here. The abstract syntax of the language of partial specifications, PPr, is that of the process language in Chapter 3, augmented with $\mathcal{U}$: [1]

$$P ::= nil \mid a.P_1 \mid P_1 + P_2 \mid x \mid \mu x.P \mid \mathcal{U}$$

The operational semantics given by [Larsen and Thomsen 87] is that of the process language in Chapter 3 , augmented with a rule for $\mathcal{U}$, stating that $\mathcal{U}$ has a special, 'wild' transition $*$, which it is always able to perform:

$$\mathcal{U} \xrightarrow{*} \mathcal{U}$$

The transition system is then $\mathcal{PP} = (PPr, Act \cup \{*\}, \rightarrow)$.

The partial specification gives information about its implementations, that is, the process expressions obtained by 'filling the holes' through replacing the $\mathcal{U}$'s with proper process expressions. For one knows that the resulting implementation *must* have an a-transition at a certain place if the partial specification had an a-transition at that place. Otherwise, in the places in the partial specification where an $\mathcal{U}$ occurs, one can only say that an implementation *may* have an a-transition.

Larsen and Thomsen use this intuition to define two new transition relations, $\rightarrow_{may}, \rightarrow_{must} \subseteq PPr \times Act \times PPr$ by

$$\begin{array}{rcl} P \xrightarrow{a}_{may} P' & \Leftrightarrow^{\Delta} & P \xrightarrow{a} P' \vee P \xrightarrow{*} P' \\ P \xrightarrow{a}_{must} P' & \Leftrightarrow^{\Delta} & P \xrightarrow{a} P' \end{array}$$

It should be obvious that $\mathcal{PP} = (PPr, Act, \rightarrow_{must}, \rightarrow_{may})$ is a modal transition system with $\rightarrow_{must} = \rightarrow_{\Box}$ and $\rightarrow_{may} = \rightarrow_{\Diamond}$.

Larsen and Thomsen now define what they call *partial bisimulation*:

Definition 4.10 *A relation $R \subseteq PPr \times PPr$ on $\mathcal{PP} = (PPr, Act, \rightarrow_{must}, \rightarrow_{may})$ is a* partial bisimulation *iff whenever PRQ the following holds:*

1. $P \xrightarrow{a}_{may} P' \Rightarrow \exists Q' : Q \xrightarrow{a}_{may} Q' \wedge P'RQ'$
2. $Q \xrightarrow{a}_{must} Q' \Rightarrow \exists P' : P \xrightarrow{a}_{must} P' \wedge P'RQ'$

[1]We will denote partial specifications with capital letters P, Q etc.

They then go on to say that there exists a largest partial bisimulation, which they then call $\lhd$. But what they have defined are just the refinements on $\mathcal{PP} = (PPr, Act, \rightarrow_{must}, \rightarrow_{may})$ and the largest such, so this indeed subsumes our usual concept of $\lhd$.

We can now give an operational semantics of partial specifications using a modal transition system. The notions of free variables and substitution are as for the process language with the obvious extensions that $FV_{PPr}(\mathcal{U}) = \emptyset$ and $\mathcal{U}[Q/x] = \mathcal{U}$.

Definition 4.11 (Operational semantics for partial specifications)
In $\mathcal{S}_{\mathcal{PP}} = (PPr, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ the transition relations $\rightarrow_\Box$ and $\rightarrow_\Diamond$ are defined as the smallest (w.r.t. $\subseteq$) pair of relations $\rightarrow_m \subseteq Pr \times Act \times Pr$, $m \in \{\Box, \Diamond\}$ satisfying:

'Unspecification'

$$\mathcal{U} \xrightarrow{a}_\Diamond \mathcal{U} \quad \textit{for all } a \in Act$$

Prefixing

$$a.P \xrightarrow{a}_\Box P$$

$$a.P \xrightarrow{a}_\Diamond P$$

Nondeterminism

$$\frac{P \xrightarrow{a}_m P'}{P + Q \xrightarrow{a}_m P'} \quad \frac{Q \xrightarrow{a}_m Q'}{P + Q \xrightarrow{a}_m Q'}$$

Recursion

$$\frac{P[\mu x.P/x] \xrightarrow{a}_m P'}{\mu x.P \xrightarrow{a}_m P'}$$

where $m \in \{\Box, \Diamond\}$.

It should be clear that this operational semantics yields the same transition relations as the old one:

Proposition 4.7 $\mathcal{S}_{\mathcal{PP}} = (PPr, Act, \rightarrow_\Diamond, \rightarrow_\Box) = (PPr, Act, \rightarrow_{may}, \rightarrow_{must})$

Proof: Inductions in the length of inferences establishing the transitions using the two sets of operational rules. □

4.3.2 A denotational description of partial specifications

In the previous section we saw that the converse of Corollary 4.3 does not hold in MPL in general. We gave a condition for $F \not\sqsubseteq G$ by means of a restricting *relation*, $\ntriangleleft_D$. Now we show how one by putting restrictions on the *language* can get the converse of Corollary 4.3, namely by restricting ourselves to the language of partial specifications.

Why can we see the partial specification language as a subset of MPL ? The process constructs in the partial specification language are the same. One only notices the renamings of Nil and rec to nil and μ. We see from the operational behaviour of the action prefixing construct $a.\_$ that it corresponds closely to the prefixing $a_\Box.\_$ in MPL. Finally, we see that the $\mathcal{U}$ has exactly the behaviour of the $\mathcal{U}$ -specification we have mentioned in Chapters 2 and 3. It is therefore natural to give a denotational semantics reflecting this.

We use the same semantic algebra as before:

Definition 4.12 (Denotational semantics for partial specifications) *Define for $P \in PPr$ the function $[\![\_]\!] : PPr \to (\boldsymbol{E} \to \boldsymbol{2}^{Pr})$ inductively by:*

$$
\begin{aligned}
[\![\mathcal{U}]\!]\sigma &= Pr \\
[\![nil]\!]\sigma &= \text{nil} \\
[\![x]\!]\sigma &= \sigma(x) \\
[\![a.P]\!]\sigma &= \lceil a_\Box \rceil([\![P]\!]\sigma) \\
[\![P+Q]\!]\sigma &= [\![P]\!]\sigma \oplus [\![Q]\!]\sigma \\
[\![\mu x.P]\!]\sigma &= \text{FIX } V.([\![P]\!]\sigma\{V/x\})
\end{aligned}
$$

where $\sigma \in \boldsymbol{E}$ and FIX *is the* maximal *fixed point operator.*

4.3.3 A full abstractness result

We are now dealing with what can be thought of as simply a subset of MPL with a slightly altered syntax. The notions of guardedness extend those of MPL in the natural way by simply stating that $UG_{PPr}(\mathcal{U}) = \emptyset$. We can therefore again call upon Theorem 4.2, getting that we for closed, well-guarded P have

$$[\![P]\!] = \{p \mid p \triangleleft P\} \tag{4.1}$$

and that we for P and Q well-guarded have

$$P \lhd Q \Rightarrow [\![P]\!] \subseteq [\![Q]\!] \tag{4.2}$$

One observes that this implies

Corollary 4.4

$$S \simeq T \Rightarrow [\![S]\!] = [\![T]\!]$$

which we will be using a lot in the proof to come.

As already revealed, the *converse* of (4.2) holds with the denotational description just given. To prove this, we need some lemmae, giving normal-form-like equational characterizations of partial specifications and processes. The proof of the latter is given in [Milner 82]. In establishing the former we need these two propositions:

Proposition 4.8

$$\mu x.(P + x) \simeq \mu x.P$$

PROOF: See [Thomsen 87]. □

Proposition 4.9

$$\mu x.P \simeq P[\mu x.P/x]$$

PROOF: Apparent from the operational rule for recursion. □

Lemma 4.6 *For any $P \in PPr$ with free variables in $\{Y_1, Y_2, \ldots, Y_n\}$, there exist expressions $P_1, \ldots, P_r$ also with free variables in $\{Y_1, Y_2, \ldots, Y_n\}$ such that we have a system of r equations over $\simeq$ of the form:*

$$P_i \simeq \sum_{j=1}^{M(i)} a_{ij} P_{f(i,j)} + \sum_{j=1}^{N(i)} Y_{g(i,j)} \;\; [+\mathcal{U}]$$

with $P \simeq P_1$, f and g being index functions of type $\mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$, and $[+\mathcal{U}]$ indicating the possible presence of a $\mathcal{U}$. [2]

PROOF: Structural induction on P.

$P = \mathcal{U}$**:** $r = 1$; the only equation is $P_1 \simeq \mathcal{U}$.

[2]If $M(i) = 0$ or $N(i) = 0$, this is the same as writing a *nil*.

$P = nil$: $r = 1$; the only equation is $P_1 \simeq nil$.

$P = x$: $r = 1$; the only equation is $P_1 \simeq x$.

$P = P' + P''$: By induction hypothesis we have two systems of equations with r' and r'' equations, respectively. Set $r = r' + r'' + 1$. For the equations, take the union of the systems for P' and P'' adding the equation

$$P_1 \simeq \underline{rhs}(P'_1) + \underline{rhs}(P''_1)$$

where $\underline{rhs}(Q)$ is the right-hand side of the equation for Q with multiple $\mathcal{U}$'s reduced to one $\mathcal{U}$.

$P = a.P'$: Suppose we have r' equations for P'. Then $r = r' + 1$ and we obtain the system of equations by adding

$$P_1 \simeq a.P'_1$$

to the system of equations for P'.

$P = \mu x.P'$: Suppose we for P' have the equations

$$P'_i \simeq \sum_{j=1}^{M(i)} a_{ij}.P'_{f(i,j)} + \sum_{j=1}^{N(i)} Y_{g(i,j)}[+x][+\mathcal{U}]$$

where $1 \leq i \leq r'$. x may appear in the equations, since there may be free occurrences of x in P'. Now define

$$P^*_1 =^{\Delta} \sum_{j=1}^{M(1)} a_{1j}P'_{f(1,j)} + \sum_{j=1}^{N(1)} Y_{g(i,j)}[+\mathcal{U}]$$

By Propositions 4.8 and 4.9 we then have

$$P \simeq P^*_1[P/x]$$

since either $P' \simeq P^*_1$ or $P' \simeq P^*_1 + x$.

Now set

$$P_i =^{\Delta} P'_i[P/x]$$

$(1 \leq i \leq r)$. Substituting P for x in the right-hand sides of the P'_i-equations, we then get

$$P_i \simeq \sum_{j=1}^{M(i)} a_{ij}P_{f(i,j)} + \sum_{j=1}^{N(i)} Y_{g(i,j)}[+\mathcal{U}] \tag{4.3}$$

$$[+ \sum_{j=1}^{M(1)} a_{1j}P_{f(1,j)} + \sum_{j=1}^{M(1)} Y_{g(1,j)}] \tag{4.4}$$

which is a system of equations of the desired form.

This completes the proof. □

Lemma 4.7 ([Milner 82])

For any $q \in Pr$ with free variables in $\{Y_1, Y_2, \dots, Y_n\}$, there exist expressions $q_1, \dots, q_r$ also with free variables in $\{Y_1, Y_2, \dots, Y_n\}$ such that we have a system of r equations over $\sim$ of the form:

$$q_i \sim \sum_{j=1}^{m(i)} a_{ij} q_{f(i,j)} + \sum_{j=1}^{n(i)} Y_{g(i,j)}$$

with $p \sim q_1$, f and g being index functions of type $\mathbf{N} \times \mathbf{N} \to \mathbf{N}$.[3]

Another result in [Milner 82] states that whenever such a system of equations of processes is defined it has a *unique solution.*

Lemmae 4.6 and 4.7 can be used to describe when a process q lies in the denotation of a partial specification P:

Lemma 4.8 *Let $q \in Pr$ and $P \in PPr$ determine systems of equations as given by Lemmae 4.6 and 4.7 of the forms*

$$q_i \sim \sum_{j=1}^{m(i)} a_{ik} q_{f(i,k)} \text{ with } q \sim q_1$$

and

$$P_{i'} \simeq \sum_{k=1}^{M(i')} b_{i'k} P_{g(i',k)} [+\mathcal{U}] \text{ with } P \simeq P_1$$

Then for any h, l we have

$$q_h \in [\![P_l]\!]$$

if and only if

$$\begin{cases} \forall j \leq m(h) \exists k \leq M(l) : a_{hj} = b_{lk} \quad \wedge \quad q_{f(h,j)} \in [\![P_{g(l,k)}]\!] \\ \forall k \leq M(l) \exists j \leq m(k) : a_{hj} = b_{lk} \quad \wedge \quad q_{f(h,j)} \in [\![P_{g(l,k)}]\!] \end{cases}$$

when $\mathcal{U}$ does not appear in the equation for P_l and

$$\forall k \leq M(l) \exists j \leq m(i) : a_{hj} = b_{ik} \wedge q_{f(h,j)} \in [\![P_{g(l,k)}]\!]$$

when $\mathcal{U}$ appears in the equation for P_l.

[3]As before, if $m(i) = 0$ or $n(i) = 0$, we interpret this as *nil*

Proof: Straightforward; by Corollary 4.4 we have either

$$[\![P_l]\!] = [\![\sum_{k=1}^{M(l)} b_{lk} P_{g(l,k)}]\!] \tag{4.5}$$

or

$$[\![P_l]\!] = [\![\sum_{k=1}^{M(l)} b_{lk} P_{g(l,k)} + \mathcal{U}]\!] \tag{4.6}$$

Consider first (4.5). We then have

$$\begin{aligned} [\![P_l]\!] &= \bigoplus_{k=1}^{M(l)} \lceil b_{lk\Box} \rceil [\![P_{g(l,k)}]\!] \\ &= \{p \mid \forall k \leq M(l) \exists p_k \in \lceil b_{lk\Box} \rceil [\![P_{g(l,k)}]\!] : p \sim \sum_{k=1}^{M(l)} p_k\} \end{aligned}$$

from which the lemma follows by the definition of $\lceil \_{\Box} \rceil$. Consider then (4.6). Along the same lines, we have

$$[\![\sum_{k=1}^{M(l)} b_{1k} P_{g(l,k)} + \mathcal{U}]\!] = \{p \mid \exists p', \forall k \leq M(l) \exists p_k \in \lceil b_{lk\Box} \rceil [\![F_{g(l,k)}]\!] : p \sim \sum_{k=1}^{M(l)} p_k + p'\}$$

from which we get the second case of the lemma. □

We are now almost ready for the proof. Again the trick consists in devising a *canonical implementation* scheme.

One of our basic assumptions is that the modal transition system $\mathcal{S}_{\mathcal{PP}} = (PPr, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ and the process system are image-finite, for as seen in Chapter 2, we then have $\triangleleft = \bigcap_{n=0}^{\infty} \mathcal{RE}^n((PPr \cup Pr)^2)$.

Another basic assumption is that the set of actions used in the partial specifications does not completely exhaust the set of actions used in the processes, in that we assume the existence of an action d not demanded by any specification, i.e. $\forall P \in PPr : P \overset{d}{\not\rightarrow}_\Box$. The canonical implementation is simply the partial specification with $\mathcal{U}$'s replaced by *d.nil* everywhere. More formally, we have

Definition 4.13 *The syntactic transformation $q_P : PPr \rightarrow Pr$ is defined structurally by*

$$q_P(\mathcal{U}) \quad = \quad d.nil$$

$$\begin{aligned}
q_P(nil) &= nil \\
q_P(a.P) &= a.q_P(P) \\
q_P(P_1 + P_2) &= q_P(P_1) + q_P(P_2) \\
q_P(\mu x.P_1) &= \mu x.q_P(P_1) \\
q_P(x) &= x
\end{aligned}$$

The transitions of the canonical implementation are given by

Lemma 4.9

$$q_P(P) \xrightarrow{a} p' \Leftrightarrow \exists P' : P \xrightarrow{a}_{\Box} P' \wedge p' = q_P(P')$$

for $a \neq d$ and

$$q_P(P) \xrightarrow{d} nil \Leftrightarrow \forall a \in Act : P \xrightarrow{a}_{\Diamond} \mathcal{U}$$

Proof: Inductions in the length of the inferences establishing the transitions concerned. The proof follows the lines of that of Lemma 4.3 and is therefore omitted. □

We now have that $q_P(P)$ defines an implementation of P.

Lemma 4.10 *For all closed $P \in PPr$*

$$q_P(P) \lhd P$$

Proof: By Lemma 4.9, $\{(q_P(P), P) \mid P \in PPr\} \cup \{(nil, \mathcal{U}\}$ is a refinement. □

Now we can finally prove and state

Theorem 4.5 *If $\mathcal{S}_{\mathcal{PP}} = (PPr, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$ and $\mathcal{P} = (Pr, Act, \rightarrow)$ are image-finite, we have for all closed $P, Q \in PPr$ that*

$$[\![P]\!] \subseteq [\![Q]\!] \Rightarrow P \lhd Q$$

Proof: The proof is indirect, showing that $P \not\lhd Q \Rightarrow [\![P]\!] \not\subseteq [\![Q]\!]$. By image-finiteness, this reduces to $\forall n : (P,Q) \notin \mathcal{RE}^n((PPr \cup Pr)^2) \Rightarrow [\![P]\!] \not\subseteq [\![Q]\!]$. We prove this by showing that $\forall n : (P,Q) \notin \mathcal{RE}^n((PPr \cup Pr)^2) \Rightarrow q_P(P) \notin [\![Q]\!]$. We proceed by induction:

$n = 0$: $(P,Q) \notin \mathcal{RE}^0((PPr \cup Pr)^2)$ is impossible.

Assuming for $n = m$: $(P, Q) \notin \mathcal{RE}^{m+1}(PPr)$ can happen in 4 ways (recall Definition 4.2):

1. $\exists a : P \xrightarrow{a}_{\Diamond} \wedge Q \not\xrightarrow{a}_{\Diamond}$ *:* By Lemma 4.6 we get the equation system including the equations where $P_1 \simeq P$ and $Q_1 \simeq Q$:

$$P_1 \simeq \sum_{j=1}^{M(1)} a_{1j} P_{f(1,j)} [+\mathcal{U}]$$

$$Q_1 \simeq \sum_{k=1}^{L(1)} b_{1k} Q_{f'(1,k)}$$

(Q cannot include a $\mathcal{U}$, for then Q would possess any $\Diamond$ -transition). We get similar equations for $q_P(P)$, one of them being [4]

$$p_1 \sim \sum_{j=1}^{m(1)} {a'}_{1j} p_{h(1,j)} [+d.nil]$$

with $p_1 \sim q_P(P)$ and thus $p_1 \in [\![P]\!]$.

If $P \xrightarrow{a}_{\Diamond}$ because of the presence of $\mathcal{U}$, the term $d.nil$ assures that $q_P(P) \xrightarrow{d}$ while the equation for Q_1 shows that $Q \not\xrightarrow{d}_{\Diamond}$, implying that $q_P(P) \notin [\![Q]\!]$. Else, $P \xrightarrow{a}_{\Diamond}$ and $Q \not\xrightarrow{a}_{\Diamond}$, with $a = a_{1j}$ for some j and $b_{1k} \neq a_{1j}$ for all k. The equation for p_1 then implies that $a = a'_{1j'}$ for some j'. The 'only-if'-condition in Lemma 4.8 fails to hold, and again $q_P(P) \notin [\![Q]\!]$.

2. $\exists a \exists P' : P \xrightarrow{a}_{\Diamond} P' \wedge (\forall Q' : Q \xrightarrow{a}_{\Diamond} Q' \Rightarrow (P, Q) \notin \mathcal{RE}^m((PPr \cup Pr)^2))$*:* The aforementioned systems of equations, where $P \simeq P_1$ and $Q \simeq Q_1$, include the equations

$$P_1 \simeq \sum_{j=1}^{M(1)} a_{1j} P_{f(1,j)} [+\mathcal{U}]$$

$$Q_1 \simeq \sum_{k=1}^{L(1)} b_{1k} Q_{f'(1,k)}$$

As before, Q cannot include a $\mathcal{U}$ since it then would be able to match any of P's transitions. Suppose first that P had a transition that Q

[4]There is really no term $d.nil$ is this expression; strictly speaking the term is $d.P_k$ with an equation $P_k \simeq \sum_{j=0}^{0} a_{kj}.P_{f(k,j)}$

could not match because of the presence of a $\mathcal{U}$ in the equation for P_1. We then obtain the process equations for $q_P(P)$, one of them being

$$p_1 \sim \sum_{j=1}^{m(1)} a'_{1j} p_{h(1,j)} + d.nil$$

where $p_1 \sim q_P(P) \lhd P$. The result then follows as in the previous case.

Else, by induction hypothesis, whenever $Q \xrightarrow{a}_{\Diamond} Q'$ we always have that $q_P(P') \notin [\![Q']\!]$. But then we always have some k such that $Q_{f'(1,k)} \simeq Q'$, implying that $q_P(P') \notin [\![Q_{f'(1,k)}]\!]$. Also, for any $Q_{f'(1,k)}$ with $b_{1k} = a$, $Q_{f'(1,k)} \simeq Q'$ for some Q' where $Q \xrightarrow{a}_{\Diamond} Q'$. Now, by Lemma 4.9 we have

$$q_P(P) \sim \sum_{\{b,p' \mid q_P(P) \xrightarrow{b} p', b \neq a\}} b.p' \quad + \sum_{\{q_P(P') \mid q_P(P) \xrightarrow{a} q_P(P')\}} a.q_P(P')$$

which by Lemma 4.7 can be expanded into a system of equations. Since $q_P(P) \in [\![Q]\!]$ if and only if we for these k have $q_P(P') \in [\![Q_{f'(1,k)}]\!]$ (by Lemma 4.8) we're done.

3. $\exists a : Q \xrightarrow{a}_{\Box} \wedge P \not\xrightarrow{a}_{\Box}$ *:* We have by Lemma 4.6 systems of equations for P and Q such that

$$P_1 \simeq \sum_{j=1}^{M(1)} a_{1j} P_{f(1,j)} [+\mathcal{U}]$$

$$Q_1 \simeq \sum_{k=1}^{L(1)} b_{1k} Q_{f'(1,k)} [+\mathcal{U}]$$

with a $b_{1k'} \neq a_{1j}$ for all j. But, again constructing equations for $q_P(P)$, this implies that the 'only if'-condition in Lemma 4.8 fails to holds, and we're done.

4. $\exists a \exists Q' : Q \xrightarrow{a}_{\Box} Q' \wedge (\forall P' : P \xrightarrow{a}_{\Box} P' \Rightarrow (P', Q') \notin \mathcal{RE}^k((PPr \cup Pr)^2))$*:* Once again, Lemma 4.6 gives us systems of equations for P and Q such that

$$P_1 \simeq \sum_{j=1}^{M(1)} a_{1j} P_{f(1,j)} [+\mathcal{U}]$$

$$Q_1 \simeq \sum_{k=1}^{L(1)} b_{1k} Q_{f'(1,k)} [+\mathcal{U}]$$

This is as for subcase 2. By induction hypothesis, whenever $P \xrightarrow{a}_{\Box} P'$ we always have that $q_P(P') \notin [\![Q']\!]$. But then we have a k such that $Q_{f'(1,k)} \simeq Q'$, implying that $q_P(P') \notin [\![Q_{f'(1,k)}]\!]$. Also, for any $Q_{f'(1,k)}$ with $b_{1k} = a$, $Q_{f'(1,k)} \simeq Q'$ for some Q' where $Q \xrightarrow{a}_{\Box} Q'$. Again, we obtain the process equation

$$q_P(P) \sim \sum_{\{b,p' \mid q_P(P) \xrightarrow{b} p', b \neq a\}} b.p' \quad + \sum_{\{q_P(P') \mid q_P(P) \xrightarrow{a} q_P(P')\}} a.q_P(P')$$

and, (also by the same argument as in subcase 2), the 'if'-part in Lemma 4.8 fails to hold.

This completes the proof. □

4.3.4 Consequences of the full abstractness

The above now gives us a way of finding 'characteristic processes' similar to the 'characteristic formulae' in Hennessy-Milner logic. In [Godskesen et al 87] Godskesen, Ingolfsdottir and Zeeberg show how one, for any given process expression p can derive a logical formula F_p in Hennessy-Milner logic with recursion such that

$$q \sim p \Leftrightarrow q \models F_p$$

where $\models$ is the entailment relation in the logic. This entailment relation is really just another notation for a denotational description in our sense, and one could just as well write e.g.

$$q \sim p \Leftrightarrow q \in [\![F_p]\!]$$

The process transform q_P just defined has the property that $q_P(P) \in [\![P]\!]$ and $P \not\lhd Q \Rightarrow q_P(P) \notin [\![Q]\!]$, or equivalently, that $q_P(P) \in [\![Q]\!] \Rightarrow P \lhd Q$. We then have

Corollary 4.5 *If $\mathcal{S}_{\mathcal{PP}} = (PPr, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$ and $\mathcal{P} = (Pr, Act, \rightarrow)$ are image-finite, we have for all closed $P, Q \in PPr$ that*

$$P \lhd Q \Leftrightarrow q_P(P) \in [\![Q]\!]$$

Fortunately, we cannot use this to say that q_P determines some 'strongest implementation' in the sense that $q_P(P) \in [\![P]\!]$ and $p \in [\![P]\!] \Rightarrow p \sim q_P(P)$, for not only could we choose *any* suitable action in the definition of q_P and still get a process in $[\![P]\!]$; the mere existence of $\mathcal{U}$ shows that specifications are not limited to single equivalence classes.

Another consequence is that we can prove laws about $\lhd$ on partial specifications without establishing a refinement. For instance, we can prove that $\lhd$ is a precongruence w.r.t. action prefixing:

Proposition 4.10

$$P \lhd Q \Rightarrow a.P \lhd a.Q$$

PROOF: We have $P \lhd Q$ iff $[\![P]\!] \subseteq [\![Q]\!]$. But

$$[\![a.P]\!] = \{p \mid p \xrightarrow{b} p' \Rightarrow b = a \land p' \in [\![P]\!]\}$$

which is clearly a subset of

$$[\![a.Q]\!] = \{p \mid p \xrightarrow{b} p' \Rightarrow b = a \land p' \in [\![Q]\!]\}$$

whenever $P \subseteq Q$. □

So why would one want $\lhd$? One good reasons for having $\lhd$ around is that it is a polynomial-time decidable relation whenever we are dealing with *finite* transition systems. In [Larsen and Thomsen 87] an implementation of such a decision procedure is mentioned. On the other hand, $\subseteq$ is nicer when proving laws.

Yet another consequence of the full abstraction result is that we can give a sound and complete proof system for semantic equality of finite MPL specifications with disjunction. We will return to this in Chapter 6.

Chapter 5

Introducing static constructs

So far we have only considered a modal process logic incorporating dynamic constructs. In this chapter we introduce static constructs into MPL. The chapter starts out with definitions of static constructs and their behaviour. The most important static construct is parallel composition. The use of it motivates a new, 'observational' refinement relation on specifications, $\trianglelefteq$, which we then set out to define. The remainder of the chapter consists of three examples of the use of $\trianglelefteq$.

5.1 Operational behaviour of static constructs

The dynamic process constructs - inaction, action prefixing, nondeterminism, and recursion - yield a language which is essentially equivalent to finite automata. This means that we are only able to describe finite-state behaviours; this is not always enough. For if we consider systems with value passing or systems whose configuration changes dynamically, we no longer have a finite-state behaviour.

Of course we could use other, richer theories, viz. for instance temporal logic or Hennessy-Milner logic with recursion. However, MPL is different in purpose from these in that it is a language which directly specifies the operational behaviour of implementations. MPL can therefore be regarded as a language of 'not fully determined' processes and should therefore contain the same basic ingredients as the process language, which is here $\mathcal{CCS}$.

This calls for the introduction of all the process constructs of $\mathcal{CCS}$ into MPL. Among these are parallel composition and action restriction. It is important to assure that $\triangleleft$ is a precongruence w.r.t. the static constructs, since this will assure that a compositionality of the specifications w.r.t. their structure exists.

5.1.1 Parallel composition and action restriction

The most important static operators in $\mathcal{CCS}$ as described in [Milner 80] are parallel composition and action restriction. The introduction of these results in the need for some assumptions about the structure of the set of actions. The set of possible actions in MPL is chosen similarly:

Let our modal transition system be $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$. Then we define Act to have the structure

$$Act = A \cup \overline{A} \cup \{\tau\}$$

The set $\overline{A}$ is called the set of *co-names* of A. We introduce the co-name bijection ('bar'):

$$\overline{} : A \leftrightarrow \overline{A}$$

having the property that $\overline{\overline{a}} = a$, thus emphasizing the connection between names and co-names. The function is extended to sets of actions in the obvious way, i.e. $\overline{S} = \{a \mid \exists b \in S : a = \overline{b}\}$.

Informally one should think of $\overline{a}$-actions as representing outputs and 'unbarred' a-actions as actions representing inputs. The action τ is the analogue of τ in $\mathcal{CCS}$, representing internal communications.

The set of actions occurring in an MPL formula is called the *sort* of the formula. It can be defined structurally by

Definition 5.1 *The function* $Sort : Sp \rightarrow \mathbf{2}^{Act}$ *is defined structurally by*

$$\begin{aligned}
Sort(nil) &= \emptyset \\
Sort(x) &= \emptyset \\
Sort(a_\Box.F) &= \{a\} \cup Sort(F) \\
Sort(a_\Diamond.F) &= \{a\} \cup Sort(F) \\
Sort(F_1 + F_2) &= Sort(F_1) \cup Sort(F_2) \\
Sort(\mathrm{rec}x.F) &= Sort(F)
\end{aligned}$$

We sometimes also need to lump together names and co-names. This is done using the naming function $Names$ defined by $Names(a) = Names(\overline{a}) = a$, which is also extended to sets in the obvious way.

When defining the $\|$-construct in $\mathcal{CCS}$ this is done in terms of $|$ and using action-restriction to hide the 'internal' actions used for communication. We therefore now define parallel composition, $|$, along with action restriction, in a way similar to the definitions of parallel composition and action restriction in $\mathcal{CCS}$ [Milner 80]:

Definition 5.2 (Composition and restriction) *Let* $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ *be the modal transition system where* Sp *is the language with the abstract syntax*

$$E ::= Nil \mid a_\Box.E_1 \mid a_\Diamond.E_1 \mid E_1 + E_2 \mid x \mid \mathrm{rec}x.E \mid E_1|E_2 \mid E \setminus A$$

where A *is interpreted as a set of actions and* $\rightarrow_\Box$ *and* $\rightarrow_\Diamond$ *are the least relations satisfying the operational rules and axioms in Definition 3.6 plus*

$$\frac{S \xrightarrow{a}_m S'}{S \mid T \xrightarrow{a}_m S' \mid T} \qquad \frac{T \xrightarrow{a}_m T'}{S \mid T \xrightarrow{a}_m S \mid T'}$$

$$\frac{S \xrightarrow{a}_m S' \ , \ T \xrightarrow{\overline{a}}_m T'}{S \mid T \xrightarrow{\tau}_m S' \mid T'}$$

$$\frac{S \xrightarrow{a}_m S'}{S \setminus A \xrightarrow{a}_m S' \setminus A} \ , a \notin A$$

where $m \in \{\Box, \Diamond\}$.

Notice that the semantic description above, just as is the case in $\mathcal{CCS}$, does not reflect 'true concurrency', but resorts to an interleavings-view of concurrency. The main reason for this is that we use $\mathcal{CCS}$ as our implementation language; since we use the same process constructs in MPL the semantics should not be too different.

5.1.2 General static constructs

Parallel composition and action restriction are just special cases of the more general idea of static constructs. The static constructs are called so because they describe a fixed linkage structure between components.

All static constructs are similar in that they associate a combination of actions performed by the components in the construct with an action performed by the entire construct. One can thus regard a static construct as being defined by a function over the set of actions. We introduce a special no-action symbol, 0, not in Act, since we do not require that all components contribute to the common action (as is the case in the description of | in Definition 5.2, for instance). We define $Act_0 = Act \cup \{0\}$.

Assuming an n-ary partial function on actions, $f : Act_0^n \rightarrow Act$, any static construct involving n components $F_1, \ldots, F_n$ can now be written using the notation

$$(F_1, \ldots, F_n)[f]$$

The operational rule for static constructs basically says that the composite specification has a transition iff its components have the transitions required by f:

Definition 5.3 (General static constructs) *Let $\mathcal{S} = (Sp, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$ be the modal transition system where Sp is the language with the abstract syntax*

$$E ::= Nil \mid a_{\Box}.E_1 \mid a_{\Diamond}.E_1 \mid E_1 + E_2 \mid x \mid \mathrm{rec}x.E \mid (F_1, \ldots, F_n)[f]$$

where f is interpreted as a function on Act_0 and $\rightarrow_{\Box}$ and $\rightarrow_{\Diamond}$ are the least relations satisfying the operational rules and axioms in Definitions 3.6 plus

$$\frac{\Big[(F_i \xrightarrow{a_i}_m F_i') \ \vee \ (F_i = F_i' \wedge a_i = 0)\Big]_{i \leq n}}{(F_1 \ldots F_n)[f] \xrightarrow{a}_m (F_1' \ldots F_n')[f]} \quad a \asymp f(a_1, \ldots, a_n)$$

where $m \in \{\Box, \Diamond\}$, and $a \asymp f(a_1, \ldots, a_n)$ is true *if $f(a_1, \ldots, a_n)$ is defined and has the value a and is* false *otherwise.*

We can now define the constructs of the previous subsection by

$$S_1 \mid S_2 = (S_1, S_2)[f_{comp}]$$

where $f_{comp} : Act_0^2 \hookrightarrow Act$ is defined by

$$\begin{array}{rcl} f_{comp}(a, \overline{a}) & = & \tau \\ f_{comp}(0, a) & = & a \\ f_{comp}(a, 0) & = & a \end{array}$$

Similarly, action restriction can be defined by

$$S \setminus A = S[f_{res}]$$

where $f_{res} : Act_0 \hookrightarrow Act$ is defined by

$$f_{res}(a) = a \quad \text{if } a \in A$$

We will sometimes need the restricted parallel construct $\|$ of $\mathcal{CCS}$ [Milner 80] defined by

$$F \parallel G =^{\Delta} (F \mid G) \setminus Names(Sort(F) \cap \overline{Sort(G)})$$

where we simply remove all possible observation of the actions used in communicating.

The following theorem shows that $\lhd$ is a precongruence w.r.t. static constructs and that the introduction of these constructs preserve compositionality:

Theorem 5.1 (Compositionality) *If $F_i \lhd G_i$ for $i \leq n$, for any static construct $(\ldots)[f]$ we have*

$$(F_1, \ldots, F_n)[f] \lhd (G_1, \ldots, G_n)[f]$$

Proof: We show that the relation

$$\{\langle (F_1 \ldots, F_n)[f], (G_1, \ldots, G_n)[f] \rangle \mid \forall i. F_i \lhd G_i\}$$

is a refinement. Suppose $(F_1, \ldots, F_n)[f] \xrightarrow{a}_\Diamond (F_1', \ldots, F_n')[f]$. (These are the only transitions possible according to the operational rule). By the operational rule for static constructs we have a set $S \subseteq \{1, \ldots, n\}$ such that $F_j \xrightarrow{a_j}_\Diamond F_j'$ when $j \in S$ and $F_j \xrightarrow{0}_\Diamond F_j$ otherwise. But then we have for the same S that $G_j \xrightarrow{a_j}_\Diamond G_j'$ whenever $j \in S$ with $F_j' \lhd G_j'$ and $G_j \xrightarrow{0}_\Diamond G_j$ otherwise. But this implies that we also have $\langle (F_1', \ldots, F_n')[f], (G_1', \ldots, G_n')[f] \rangle$ in the relation. The other half of the proof, concerning $(G_1, \ldots, G_n)[f] \xrightarrow{a}_\Box (G_1', \ldots, G_n')[f]$, is similar. $\Box$

Corollary 5.1 *If $F \lhd G$, we have for any $S \in Sp$ that $F \mid S \lhd G \mid S$.*

5.1.3 Expansion theorems

In [Milner 80] an expansion theorem is given, allowing one to deduce the possible transitions of a process expression obtained using parallel composition and action restriction. A similar theorem exists for the general static constructs of MPL:

Theorem 5.2 (General expansion theorem)
For any n-ary static construct $(\ldots)[f]$ we have

$$(F_1, \ldots, F_n)[f] =_D \sum_{\pi_f((F_1, \ldots, F_n))} a_m.(F_1', \ldots, F_n')[f]$$

where $\pi_f((F_1, \ldots, F_n))$ is the set of possible derivations given by

$$\{(a_m, (F_1', \ldots, F_n')) \mid \exists (a_1, \ldots, a_n) \colon ((F_i \xrightarrow{a_i}_m F_i') \vee (a_i = 0 \wedge F_i = F_i')), 1 \leq i \leq n \wedge f(a_1, \ldots, a_n) \asymp a_m\}$$

and $=_D$ is the direct equivalence [1]*of* [Milner 80].

[1]$S =_D T$ means that S and T have the same modal transitions and that the results of matching transitions are *syntactically equal*.

Proof: Immediate from the operational rule. □

We can now use this theorem to establish a specialized expansion theorem concerning the derivations of a parallel composition of MPL specifications. This expansion theorem also shows that concurrency here is explained by nondeterministic interleavings.

Writing $(F_1 \mid \ldots \mid F_n)$ is justified by the following properties:

Proposition 5.1 $F \mid G \simeq G \mid F$.

Proof: By the operational rule. □

Proposition 5.2 $(F \mid G) \mid H \simeq F \mid (G \mid H)$

Proof: It is easy to see that $\{((R \mid S) \mid T, R \mid (S \mid T)) \mid R, S, T \in Sp\}$ is a bisimulation on $\mathcal{S} = (Sp, Act, \rightarrow_\diamond, \rightarrow_\Box)$. □

Now we can prove

Theorem 5.3 *For all $F_1, \ldots, F_n$ we have*

$$\begin{aligned}(F_1 \mid \ldots \mid F_n) \;\simeq\; & \sum_{\{(a,F_i') \mid \exists i:1 \le i \le n \wedge F_i \xrightarrow{a}_m F_i'\}} a_m.(F_1 \mid \ldots F_i' \ldots \mid F_n) \\ + & \sum_{\{(F_i',F_j') \mid \exists i,j:1 \le i < j \le n \wedge F_i \xrightarrow{a}_m F_i' \wedge F_j \xrightarrow{\overline{a}}_m F_j'\}} \tau_m.(F_1 \mid \ldots F_i' \mid \ldots F_j' \mid F_n)\end{aligned}$$

Proof: Induction in n. For $n = 2$ this is just Theorem 5.2 with $f = f_{comp}$. For the full proof, see Appendix. □

By first applying the above theorem and then Theorem 5.2 with $f = f_{res}$ we can get

Corollary 5.2 *For all $F_1, \ldots, F_n$ we have*

$$\begin{aligned}(F_1 \mid \ldots \mid F_n) \setminus A \;\simeq\; & \sum_{\{(a,F_i') \mid a \notin A \wedge \exists i:1 \le i \le n \wedge F_i \xrightarrow{a}_m F_i'\}} a_m.((F_1 \mid \ldots F_i' \ldots \mid F_n) \setminus A) \\ + & \sum_{\{(F_i',F_j') \mid a,\overline{a} \notin A \wedge \exists i,j:1 \le i < j \le n \wedge F_i \xrightarrow{a}_m F_i' \wedge F_j \xrightarrow{\overline{a}}_m F_j'\}} \tau_m.((F_1 \mid \ldots F_i' \mid \ldots F_j' \mid F_n) \setminus A)\end{aligned}$$

giving a useful characterization of the transitions of a specification constructed using $\|$.

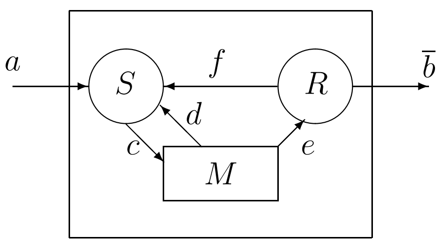

Figure 5.1: A simple data link P

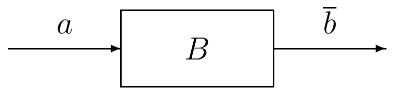

Figure 5.2: A 1-place buffer B

5.2 An observational refinement order

The introduction of static constructs also meant the introduction of the action τ. The intuition behind this action is that it should be unobservable, since it represents internal communications. $\lhd$ as defined in Chapter 3 does not reflect this, which we will now see.

5.2.1 Motivation

The simple data link P in Figure 5.1 is described in [Parrow 85]. It lets information pulses pass through one at a time. Information enters with the a-action, the sender S transmits it to the medium M through a c-communication. The medium may lose the message but then lets S know via a d-communication. The medium transmits the information to the receiver R which emits the information using a $\overline{b}$-action and sends an acknowledge to S using an f-communication. (This system may *diverge* in that the information keeps getting lost by M and retransmitted by S).

Now consider the 1-place buffer B pictured in Figure 5.2.

The buffer lets in pulses on action a and emits them using a $\overline{b}$.

One can see the buffer as an abstract description of the protocol, in that both systems *essentially* just take in pulses one at a time and release them later. We want a refinement ordering disregarding the unobservable behaviour, stating that

$$P \trianglelefteq B$$

meaning that observationally, P is a more precise description than B.

The reader should now have noticed that the motivation is analogous to the one leading to the introduction of observational equivalence ($\approx$) in [Milner 80].

5.2.2 Definition of the observational refinement order

As we shall see shortly, $\trianglelefteq$ is defined as $\triangleleft$, using two 'observational' transition relations on *strings* of actions, $\Longrightarrow_{\Box}$, $\Longrightarrow_{\Diamond} \subseteq Sp \times (Act \setminus \{\tau\})^* \times Sp$. This definition will allow us to use Park's Induction Principle when proving facts about $\trianglelefteq$.

The intuition behind $\Longrightarrow_{\Box}$, say, is that $S \overset{w}{\Longrightarrow}_{\Box} T$ if one by a sequence of $\Box$ -transitions corresponding to a string of actions, w, possibly interleaved by arbitrary $\tau_{\Box}$-transitions from S can derive T. First we need to define the concept of string derivations:

Definition 5.4 *Let* $\rightarrow^*{}_{\Box} \subseteq Sp \times Act^* \times Sp$ *be defined by* $S \overset{w}{\rightarrow}{}^*{}_{\Box}\, T$ *iff* $w = a_1a_2 \ldots a_n, n \geq 0$ *and there exist* $S_0, S_1, \ldots, S_n \in Sp$ *such that*

$$S = S_0 \xrightarrow{a_1}_{\Box} S_1 \xrightarrow{a_2}_{\Box} \cdots S_{n-1} \xrightarrow{a_n}_{\Box} S_n = T$$

$\rightarrow^*{}_{\Diamond}$ is defined similarly:

Definition 5.5 *Let* $\rightarrow^*{}_{\Diamond} \subseteq Sp \times Act^* \times Sp$ *be defined by* $S \overset{w}{\rightarrow}{}^*{}_{\Diamond}\, T$ *iff* $w = a_1a_2 \ldots a_n, n \geq 0$ *and there exist* $S_0, S_1, \ldots, S_n \in Sp$ *such that*

$$S = S_0 \xrightarrow{a_1}_{\Diamond} S_1 \xrightarrow{a_2}_{\Diamond} \cdots S_{n-1} \xrightarrow{a_n}_{\Diamond} S_n = T$$

In other words, $\rightarrow^*{}_{\Box}$ and $\rightarrow^*{}_{\Diamond}$ are the reflexive, transitive closures of $\rightarrow_{\Box}$ and $\rightarrow_{\Diamond}$. Also notice that we have $S \overset{\epsilon}{\rightarrow}{}^*{}_m\, S$, where ϵ is the empty action string.

We now define a mapping which eliminates all τ-actions:

Definition 5.6 $\tilde{\ } : Act \rightarrow (Act \setminus \{\tau\})^*$ *is defined by*

$$\tilde{a} = \begin{cases} a & \textit{if } a \neq \tau \\ \epsilon & \textit{if } a = \tau \end{cases}$$

A result in algebra [Manes and Arbib 75] states that $\tilde{\ }$ can be extended to a unique homomorphism ϕ between the corresponding free monoids:

$$\phi : (Act^*, \cdot, \epsilon) \to ((Act \setminus \{\tau\})^*, \cdot, \epsilon)$$

We can now use this homomorphism to define the observational transition relations:

Definition 5.7 *For all $S, T \in Sp$ we have*

$$S \overset{w}{\Longrightarrow}_m T, w \in (Act \setminus \{\tau\})^*$$

iff

$$\exists u \in Act^* : S \overset{u}{\longrightarrow}{}^*_m T \ \wedge \ w = \phi(u)$$

and then in turn define the observational refinements by

Definition 5.8 *A relation $R \subseteq Sp \times Sp$ on a modal transition system $\mathcal{S} = (Sp, Act, \to_\Diamond, \to_\Box)$ is an* observational refinement *iff whenever SRT the following holds for all $w \in (Act \setminus \{\tau\})^*$:*

1. $S \overset{w}{\Longrightarrow}_\Diamond S' \Rightarrow \exists T' : T \overset{w}{\Longrightarrow}_\Diamond T' \wedge S'RT'$
2. $T \overset{w}{\Longrightarrow}_\Box T' \Rightarrow \exists S' : S \overset{w}{\Longrightarrow}_\Box S' \wedge S'RT'$

The observational refinements are thus just the refinements on the 'observational' modal transition system $\mathcal{S}_\mathcal{W} = (Sp, Act, \Longrightarrow_\Diamond, \Longrightarrow_\Diamond)$. This implies that the sought largest observational refinement, $\trianglelefteq$, exists and that

Proposition 5.3 *The relation $\trianglelefteq \subseteq Sp \times Sp$ on a modal transition system $\mathcal{S} = (Sp, Act, \to_\Diamond, \to_\Box)$ satisfies*
$S \trianglelefteq T$ iff

1. $S \overset{w}{\Longrightarrow}_\Diamond S' \Rightarrow \exists T' : T \overset{w}{\Longrightarrow}_\Diamond T' \wedge S' \trianglelefteq T'$
2. $T \overset{w}{\Longrightarrow}_\Box T' \Rightarrow \exists S' : S \overset{w}{\Longrightarrow}_\Box S' \wedge S' \trianglelefteq T'$

One may worry that, since the definition of observational refinements deals with string transitions, it is not of much use in proofs concerning $\trianglelefteq$. However, it turns out that we can make do with action strings of length ≤ 1:

Proposition 5.4 *R is an observational refinement iff whenever SRT*

1. $S \overset{a}{\to}_\Diamond S' \Rightarrow \exists T' : T \overset{\phi(a)}{\Longrightarrow}_\Diamond T' \wedge S'RT'$

2. $T \xrightarrow{a}_{\Box} T' \Rightarrow \exists S' : S \overset{\phi(a)}{\Longrightarrow}_{\Box} S' \wedge S'RT'$

PROOF:

Only if: For $a \neq \tau$ this is obvious, since $\rightarrow_m \; \subseteq \; \Longrightarrow_m$. For $a = \tau$ we have that $P \xrightarrow{\tau}_m P'$ implies $P \overset{\epsilon}{\Longrightarrow}_m P'$ and we're done.

If: For string derivations, this follows from successive applications of the above. For, with $w = a_1 a_2 \ldots a_n, n \geq 0, w \in (Act \setminus \{\tau\})^*$, we have $S \overset{w}{\Longrightarrow}_m S'$ iff

$$S \overset{\tau^{k_1}}{\longrightarrow^*}_m S_1 \xrightarrow{a}_m S_2 \overset{\tau^{k_2}}{\longrightarrow^*}_m S_3 \cdots \overset{\tau^{k_n}}{\longrightarrow^*}_m S'$$

implying that

$$T \overset{\epsilon}{\Longrightarrow}_m T_1 \overset{a_1}{\Longrightarrow}_m T_2 \cdots \overset{a_n}{\Longrightarrow}_m T'$$

with $(S, T) \in R$, $(S_i, T_i) \in R$ and $(S', T') \in R$. The other clause is proved similarly.

This completes the proof. $\Box$

Besides being of great use in the correctness proofs to follow, Proposition 5.4 shows how $\trianglelefteq$ is related to $\triangleleft$:

Proposition 5.5 $S \triangleleft T \Rightarrow S \trianglelefteq T$

PROOF: By Proposition 5.4, any refinement is also an observational refinement, since $\phi(a) = a$ and $S \rightarrow_m$ implies $S \Longrightarrow_m$ for $a \neq \tau$. $\Box$

Proposition 5.6 $\trianglelefteq$ *is a preorder on Sp.*

PROOF: By Park's Induction Principle, $\trianglelefteq$ is *reflexive*, since $\{(S, S) \mid S \in Sp\}$ is an observational refinement. $\trianglelefteq$ is *transitive*, since the composition (as defined in Chapter 2) of observational refinements again results in an observational refinement. $\Box$

Finally, note that our definition of $\trianglelefteq$ subsumes the definition of $\approx$ in e.g. [Milner 83] since we in the derived $\mathcal{S}_{\mathcal{P}} = (Pr, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$ can describe the observational transition relation on processes through the fact that $\Longrightarrow_{\Diamond} = \Longrightarrow_{\Box} = \Longrightarrow$

5.3 Examples of the use of observational refinement

The rest of the chapter is devoted to three examples of the use of $\trianglelefteq$ when proving properties of MPL specifications obtained using static constructs. The examples also show how one may apply the modalities when specifying systems.

In what follows, for ease of notation we will relax the syntax of MPL a bit so that we can write a specification as a set of (possibly mutually) recursive definitions of identifiers such as

$$\begin{aligned} S &\Longrightarrow \cdots S \cdots T \cdots \\ T &\Longrightarrow \cdots S \cdots T \cdots \end{aligned}$$

as is done in [Milner 80] instead of using the somewhat cumbersome rec-notation. The '$\Longrightarrow$ ' is chosen instead of the '$\Longleftarrow$ ' to indicate that we are taking *maximal* fixed points when using recursion.

5.3.1 The 1-place buffer and data link

Consider again the buffer and data link from subsection 5.2.1. This example was considered by Joachim Parrow in [Parrow 85]. He here showed that the buffer B and a version of the data link, P_J, (incorporating a faulty medium) when described in $\mathcal{CCS}$ satisfy

$$P_J \approx B$$

Here we will show that we - as claimed in subsection 5.2.1 - have

$$P \trianglelefteq B \tag{5.1}$$

where P is an MPL specification, including among many others P_J as a valid implementation. Thus our more general correctness proof of (5.1) subsumes that of [Parrow 85]. Consider again the 1-place buffer (Figure 5.3).

The *buffer B must* be able to take in information through an a-action, whereupon it *must* release it using a $\bar{b}$-action. After that we still have a 1-place buffer. This is reflected in the specification

$$B \Longrightarrow a_{\Box}.\bar{b}_{\Box}.B$$

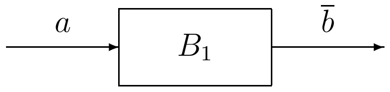

Figure 5.3: A 1-place buffer

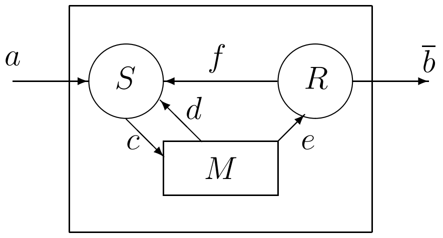

Figure 5.4: The simple data link

The data link is described through S, M, and R (see Figure 5.4).
We have

$$P = S \parallel M \parallel R$$

The *sender S must* admit information via an a-action and then emit it on a $\bar{c}$, thus communicating with M, then becoming a sender. The sender *must* also be able to receive the acknowledgment sent by R, again becoming a sender. Since the medium used may be faulty, a fault-handling capability *must* be included, for when implementing the sender one may not know whether the medium to be used is faulty or not; this calls for an exception handler. Or one may consider re-implementing the data link, replacing a (say) slow, non-faulty medium by a fast, slightly faulty one. For reasons of modularity one should not be forced to redefine the sender. Since we assume that the medium reports faults using a d-action, we get the specification

$$\begin{aligned} S &\Longrightarrow a_{\Box}.\bar{c}_{\Box}.S' \\ S' &\Longrightarrow f_{\Box}.S\,[+d_{\Box}.\bar{c}_{\Box}.S'] \end{aligned}$$

The part of the specification concerned with exception-handling is enclosed in brackets.

All implementations of the *medium* M *must* admit pulses on c and emit them again with an $\overline{e}$-action. We might thus specify a perfect medium M_p as

$$\begin{array}{rcl} M_p & \Longrightarrow & c_\Box.M'_p \\ M'_p & \Longrightarrow & \overline{e}_\Box.M_p \end{array}$$

Since we do not rule out the possibility of a faulty but honest M we introduce an *optional* exception-handling part, which responds to a failure by communicating with S by a $\overline{d}$-action. But then the medium *must* be ready for a retransmission at c. Again putting the exception-handler in brackets, we get

$$\begin{array}{rcl} M & \Longrightarrow & c_\Box.M' \\ M' & \Longrightarrow & \overline{e}_\Box.M\left[+\overline{d}_\Diamond.M\right] \end{array}$$

The perfect medium is of course a refinement (in the sense of Chapter 2) of the possibly faulty medium, i.e. $M_p \lhd M$, since $\{(M_p, M), (M'_p, M')\}$ is easily seen to be a refinement.

The *receiver* R has no exception parts and is simply defined as any process which *must* admit pulses on e and then emit them on $\overline{b}$ and send an acknowledgment on $\overline{f}$:

$$R \Longrightarrow e_\Box.\overline{b}_\Box.\overline{f}_\Box.R$$

Given the above MPL specifications, we can now show (5.1) through exhibiting an observational refinement. Using the operational rules and the expansion theorems, we see that the behaviour of the specifications is given by the transition diagram in Figure 5.5. We have here named the sets as follows:

$$\begin{array}{rcl} Q_1 & =^\Delta & (\overline{c}_\Box.S' \parallel M \parallel R) \\ Q_2 & =^\Delta & (S' \parallel M' \parallel R) \\ Q_3 & =^\Delta & (S' \parallel M \parallel \overline{b}_\Box.\overline{f}_\Box.R) \\ Q_4 & =^\Delta & (S' \parallel M \parallel \overline{f}_\Box.R) \end{array}$$

Observe that there is a τ-loop in the diagram for the data link, implying possible divergence.

From the transition diagrams one can now see by using Proposition 5.4 that

$$\{(P, B), (Q_1, B'), (Q_2, B'), (Q_3, B'), (Q_4, B)\}$$

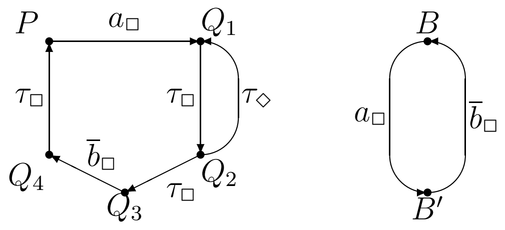

Figure 5.5: Data link and 1-place buffer

is the sought observational refinement. Take a look at (Q_1, B'), for instance. Suppose that $Q_1 \xrightarrow{\tau}_\Diamond Q_2$ (the only $\Diamond$ -transition possible here). Then $\phi(\tau) = \epsilon$, $B' \overset{\epsilon}{\Longrightarrow}_\Diamond B'$, and we have (Q_2, B') in the relation. And $B' \xrightarrow{\overline{b}}_\Box B$ can be matched by $Q_1 \overset{\tau^2\overline{b}}{\longrightarrow^*}_\Box Q_4$, implying that $Q_1 \overset{\overline{b}}{\Longrightarrow}_\Box Q_4$ with (Q_4, B) again in the relation.

Returning briefly to the 'perfect' medium, we saw that $M_p \lhd M$. Theorem 5.1 then says that $(S \parallel M_p \parallel R) \lhd (S \parallel M \parallel R)$. This again means that we have $S \parallel M_p \parallel R \unlhd B$, which is what we would expect - a data link with a perfect medium should also be an implementation of the buffer. And since both $S \parallel M_p \parallel R$ and B possess only $\Box$ -transitions, they can be regarded as process expressions and $\unlhd$ as $\approx$, thus also establishing an observational equivalence. (Along the same lines, we can prove Joachim Parrow's original result for a data link with a faulty medium.)

5.3.2 A restartable system

In his Ph.D. thesis [Prasad 87], K.V.S. Prasad motivates his theoretical work with this simple example. Consider a system consisting of an *essential system* - for simplicity we here use the 1-place buffer from the previous example - a little *demon* making the essential system crash from time to time, and an *exception handler* being evoked when the essential system has crashed. The handler sets up a new essential system having the state of the old one. The system is sketched in Figure 5.6. How can we describe the class of such systems in MPL ??

The *demon* D is simply a process which *may* interrupt the essential system through an $\overline{f}$-action and then return to its demonic behaviour. This is modelled by

$$D \Longrightarrow \overline{f}_\Diamond.D$$

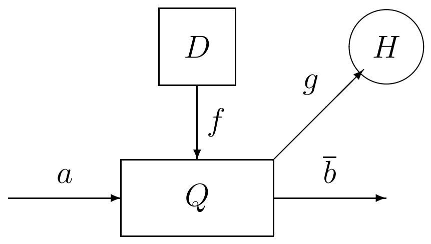

Figure 5.6: A simple restartable system

The *essential system* Q contains the 1-place-buffer and *possibly* a part that, when crashing with the demon on an f, stops the buffer behaviour, evokes the handler and then kills the crashed system. The whole essential system is thus

$$Q \Longrightarrow a_{\Box}.\overline{b}_{\Box}.Q\,[+f_{\Diamond}.\overline{g}_{\Box}.Nil]$$

(Again, the part of the specification dealing with exceptional conditions is put in brackets. Also notice that while we of course do not require that the essential system responds to the demon, *if* it does, it *must* evoke the handler.)

The *handler* H is a process that, upon receiving a g-signal from Q *must* produce a new Q together with a new handler:

$$H \Longrightarrow g_{\Box}.(Q \parallel H)$$

The entire system, $SF = (Q \parallel H \parallel D)$, should observationally refine our 1-place buffer, i.e. we want

$$SF \trianglelefteq B \tag{5.2}$$

It is here important to observe that this seemingly simple system has an *infinite* number of states, for after a crash the entire system still holds the remainders of the old essential system. For we have, apart from the 'normal' transition

$$SF \xrightarrow{a}_{\Box} (\overline{b}_{\Box}.Q \parallel D \parallel H)$$

followed by

$$(\overline{b}_\Box.Q \parallel D \parallel H) \xrightarrow{\overline{b}}_\Box (Q \parallel D \parallel H)$$

the 'crash' transition

$$SF \xrightarrow{\tau}_\Diamond (\overline{g}_\Box.Nil \parallel D \parallel H)$$

followed by

$$(g_\Box.Nil \parallel D \parallel H) \xrightarrow{\tau}_\Box (Nil \parallel D \parallel (Q \parallel H))$$

If we denote $Nil \parallel Nil \cdots \parallel Nil$ n times by Nil^n we can name the states as follows:

S_k	is the system after k crashes, ready to receive on a
S_k^b	is the system after k crashes, ready to release on $\overline{b}$
S'_k	is the system right after the $k+1$th crash

or, formally,

$$\begin{aligned}
S_k &= (Nil^k \parallel D \parallel Q \parallel H)\\
S_k^b &= (Nil^k \parallel D \parallel \overline{b}_\Box.Q \parallel H)\\
S'_k &= (Nil^k \parallel D \parallel \overline{g}_\Box.Nil \parallel g_\Box.(Q \parallel H))
\end{aligned}$$

We then get the modal transition diagram in Figure 5.7.

$$\begin{tikzcd}
S_0^b \arrow[d, bend left, "\overline{b}_\Box"] & & S_1^b \arrow[d, bend left, "\overline{b}_\Box"] & \\
S_0 \arrow[u, bend left, "a_\Box"] \arrow[r, "\tau_\Diamond"] & S'_0 \arrow[r, "\tau_\Box"] & S_1 \arrow[u, bend left, "a_\Box"] & \cdots
\end{tikzcd}$$

Figure 5.7: Modal transition diagram for restartable system

One again discovers the possibility of divergence, here not because of a τ-cycle but because of the possibility of always 'moving to the right' in the diagram. (The system repeatedly crashes and recovers!!). From Figures 5.7 and 5.5 we see that (5.2) holds, because of the observational refinement

$$\begin{tikzcd}
B_0 \arrow[r, bend right, "a_\Diamond"'] & B_1 \arrow[l, bend right, "b_\Box"'] \arrow[r, bend right, "a_\Diamond"'] & B_2 \arrow[l, bend right, "b_\Box"'] \arrow[r, bend right, "a_\Diamond"'] & \cdots\cdots \arrow[l, bend right, "b_\Box"']
\end{tikzcd}$$

Figure 5.8: Specification of pulse buffer

$$\{(S_k, B) \mid k \geq 0\} \cup \{(S_k^b, B') \mid k \geq 0\} \cup \{(S'_k, B) \mid k \geq 0\}$$

where $SF = S_0$.

Suppose $S_k \xrightarrow{a}_\Diamond S_k^b$. Then $B \overset{a}{\longrightarrow}{}^*_\Diamond B'$, implying that $B \overset{a}{\Longrightarrow}_\Diamond B'$, and (S_k^b, B') is in the relation. Or, if $S_k \xrightarrow{\tau}_\Diamond S'_k$ we have that $B \overset{\epsilon}{\longrightarrow}{}^*_\Diamond B$, implying that $B \overset{\epsilon}{\Longrightarrow}_\Diamond B$ and (S'_k, B) is in the relation. If $B \xrightarrow{a}_\Box B'$ we have $S_k \xrightarrow{a}_\Box S_k^b$ and then $S_k \overset{a}{\Longrightarrow}_\Box S_k^b$ with (S_k^b, B') in the relation.

The fact that (5.2) holds shows that any fault-tolerant system with a *nil*-demon is also an implementation of the buffer and that any fault-tolerant system where the demon is ignored also implements the buffer; not a very surprising fact.

5.3.3 Generalized data link and buffer

As the final example we will generalize the buffer and data link presented earlier in subsection 5.3.1 and show that the generalized data link observationally refines the general buffer. We here only consider the modal transition diagrams, not giving explicit MPL formulae since they in this particular case do not seem to aid in the understanding.

In [Larsen and Thomsen 88] Kim Larsen and Bent Thomsen showed how one could describe a general buffer which may accumulate a number of pulses before releasing them - see Figure 5.8. The specification says nothing about the capacity of the buffer; a buffer may even not allow any information to enter to begin with. But if a buffer has information pulses in it, it *must* be able to release them. We assume again that we let in information on an a-action and release it with a b-action. The state of a buffer is characterized by the number of pulses it is holding; call a buffer with n pulses B_n. We then get the modal transition diagram of Figure 5.8.

The actions of the data link are again as in the version considered in subsection 5.3.1. However, we now allow larger classes of implementations for the components.

The state of a *sender* S can be characterized by the number of acknowledgments still to be received, m, and the number of pulses still to be sent on to the medium, n. If a sender in state $S_{m,n}$ receives a pulse at a, it *may* let it in, becoming $S_{m,n+1}$. Hence for $m \geq 0$ we demand $S_{m,n} \xrightarrow{a}_{\Diamond} S_{m+1,n+1}$ (there is now one more acknowledgment to receive and one more message to send). When it receives a fault-report from the medium on d it *must* respond to this. Thus $S_{m,n} \xrightarrow{d}_{\Box} S_{m,n+1}$. And when the sender receives an acknowledgement at f it *must* respond by becoming $S_{m-1,n}$. Hence $S_{m,n} \xrightarrow{f}_{\Box} S_{m-1,n}$. Finally, it is essential that the sender *must* be able to send a pulse to the medium via c, wherefore we have $S_{m,n} \xrightarrow{\overline{c}}_{\Box} S_{m,n-1}$.

This altogether leads to the modal transition diagram in Figure 5.9.

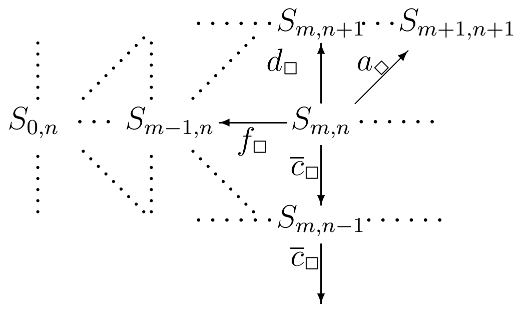

Figure 5.9: Modal transition diagram of the general sender

The *medium* M was described in Chapter 2, Example 2.4. The state of the medium can be characterized by the number of pulses, l, it is holding. A necessary requirement to a medium is that it *must* be able to take in at least one pulse and release it, for otherwise we cannot guarantee that information that has entered the data link can reach the receiver. Apart from that, the specification does not restrict the capacity of the medium. This means $M_0 \xrightarrow{c}_{\Box} M_1$ and $M_l \xrightarrow{c}_{\Diamond} M_{l+1}$ for $l > 0$. For all $l > 0$ we have $M_l \xrightarrow{\overline{e}}_{\Box} M_{l-1}$. Since it is not essential that the medium is faulty, we have $M_l \xrightarrow{\overline{d}}_{\Diamond} M_{l-1}$ for $l > 0$. Altogether this leaves us with the modal transition diagram in Figure 5.10.

The state of the general *receiver* R is characterized by the number of acknowledgments, p it has yet to send to the sender using the $\overline{f}$-action,

$$\begin{tikzcd}
M_0 & M_1 \arrow[l, "\overline{d}_\Diamond"'] \arrow[l, bend right=60, "\overline{e}_\Box"'] & M_2 \arrow[l, "\overline{d}_\Diamond"'] \arrow[l, bend right=60, "\overline{e}_\Box"'] \arrow[r, bend left=60, dash] \arrow[r, bend right=60, dash] & \cdots\cdots \\
M_0 \arrow[r, phantom] & & &
\end{tikzcd}$$

$$\begin{tikzcd}
M_0 \arrow[r, bend right=60, "c_\Box"'] & M_1 \arrow[r, bend right=60, "c_\Diamond"'] & M_2
\end{tikzcd}$$

Figure 5.10: Modal transition diagram specifying a general medium.

$$\begin{tikzcd}
& & & R_{p+1,q+1} \\
& \cdots R_{p-1,q} & R_{p,q} \arrow[l, "\overline{f}_\Box"] \arrow[ur, "e_\Diamond"] \arrow[d, "\overline{b}_\Box"] & \\
R_{1,1} \arrow[ur, dotted, no head] & & R_{p,q-1} & \\
R_{0,0} \arrow[u, "e_\Box"] & & &
\end{tikzcd}$$

Figure 5.11: Modal transition diagram of the general receiver

and by the number of pulses, q, it has yet to emit on action $\overline{b}$. As for the medium, the receiver *must* be able to take in at least one information pulse on e; otherwise we could not guarantee that information that has entered the system can leave it. So we have $R_{p,0} \xrightarrow{e}_\Box R_{p+1,1}$. For $q > 0$ we only let $R_{p,q} \xrightarrow{e}_\Diamond R_{p+1,q+1}$. Any information in the receiver must always be able to leave on $\overline{b}$, so we require $R_{p,q} \xrightarrow{\overline{b}}_\Box R_{p,q-1}$ for $q > 0$. And the receiver must be able to send the acknowledgment, so we have $R_{p,q} \xrightarrow{\overline{f}}_\Box R_{p-1,q}$.

This gives us the modal transition diagram in Figure 5.11.

Now we have described the transition capabilities for generalized components in a generalized data link. Next step is to show that a buffer B_k containing k information pulses is an abstract description of a data link containing k information pulses:

$$(S \parallel M \parallel R)_k \trianglelefteq B_k \tag{5.3}$$

What is a data link containing k pulses ? Answer: It is a system where the components altogether contain k pulses. In fact, one can see the data link as consisting of three communicating buffers. We thus only consider compositions $S_{m,n} \parallel M_l \parallel R_{p,q}$ where $k = n + l + q$.

And

$$\mathbf{R} = \{(S_{m,n} \parallel M_l \parallel R_{p,q}, B_k) \mid k = n + l + q\}$$

is indeed an observational refinement.

The operational rules (or the expansion theorems) yield information about the *data link transitions.*

One type of transitions gives information about information entering the data link. For $n > 0$ we get

$$S_{m,n} \parallel M_l \parallel R_{p,q} \xrightarrow{a}_{\Diamond} S_{m+1,n+1} \parallel M_l \parallel R_{p,q}$$

This transition says that we by performing an a may bring one more pulse into the data link by placing it in the sender component.

Other transitions state that information can always leave the system when the receiver is non-empty ($q > 0$):

$$S_{m,n} \parallel M_l \parallel R_{p,q} \xrightarrow{\bar{b}}_{\Box} S_{m,n} \parallel M_l \parallel R_{p,q-1}$$

All other transitions are unobservable. The sender and medium must be able to communicate a pulse if there are no pulses in the medium and pulses in the sender ($l = 0, n > 0$).

$$S_{m,n} \parallel M_0 \parallel R_{p,q} \xrightarrow{\tau}_{\Box} S_{m,n-1} \parallel M_1 \parallel R_{p,q}$$

For $l > 0, n > 0$ we get

$$S_{m,n} \parallel M_l \parallel R_{p,q} \xrightarrow{\tau}_{\Diamond} S_{m,n-1} \parallel M_{l+1} \parallel R_{p,q}$$

Similarly, if there are no pulses in receiver and the medium is non-empty ($l > 0, q = 0$), they must communicate a pulse:

$$S_{m,n} \parallel M_l \parallel R_{p,0} \xrightarrow{\tau}_{\Box} S_{m,n} \parallel M_{l-1} \parallel R_{p,1}$$

For $l > 0, q > 0$ we get

$$S_{m,n} \parallel M_l \parallel R_{p,q} \xrightarrow{\tau}_{\Diamond} S_{m,n} \parallel M_{l-1} \parallel R_{p,q+1}$$

If the medium reports a failure this must be because it lost a pulse ($l > 0$) and then an unobservable communication may take place:

$$S_{m,n} \parallel M_l \parallel R_{p,q} \xrightarrow{\tau}_{\Diamond} S_{m,n+1} \parallel M_{l-1} \parallel R_{p,q}$$

There must at any time be the possibility of the communication of an acknowledgment from the receiver to the sender, if there are any to send $(m > 0, p > 0)$:

$$S_{m,n} \parallel M_l \parallel R_{p,q} \xrightarrow{\tau}_{\Box} S_{m-1,n} \parallel M_l \parallel R_{p-1,q}$$

The *buffer transitions* are

$$B_k \xrightarrow{a}_{\Diamond} B_{k+1}$$

$$B_k \xrightarrow{\overline{b}}_{\Box} B_{k-1}$$

We now easily see why $\mathbf{R}$ is an observational refinement. For, with $k = n+l+q$ we have that $(S_{m,n} \parallel M_l \parallel R_{p,q}) \xrightarrow{a}_{\Diamond} (S_{m+1,n+1} \parallel M_l \parallel R_{p,q})$ can be matched directly by $B_k \xrightarrow{a}_{\Diamond} B_{k+1}$ and $((S_{m+1,n+1} \parallel M_l \parallel R_{p,q}), B_{k+1}) \in \mathbf{R}$. Else, if $(S_{m,n} \parallel M_l \parallel R_{p,q}) \xrightarrow{\tau}_{\Diamond}$, these unobservable transitions always preserve the size of $n+l+q$, as is easily seen, and can be matched by $B_k \overset{\epsilon}{\Longrightarrow}_{\Diamond} B_k$, the results still in $\mathbf{R}$.

Any $\overline{b}_{\Box}$-transition made by the buffer must be due to $k > 0$. This means that at least one of n, l and q is non-zero, and thus we can by a series of $\tau_{\Box}$-transitions (not changing $n+l+q$) and a final $\overline{b}_{\Box}$-transition match this such that $n+l+q$ also decreases by one.

Notice that we in the above never used the sizes of m and p. This shows that the acknowledgment capabilities of the sender and receiver were here not essential to the specification. We might alternatively have marked the transitions here as $\Diamond$ -transitions or even left them out; we would still get (5.3).

Actually, we also have that

$$\mathbf{R}^T = \{(B_k, (S_{m,n} \parallel M_l \parallel R_{p,q}) \mid k = n+l+q\}$$

is an observational refinement, meaning that the buffer also observationally refines the data link ! This means that any buffer can be implemented using components satisfying the specifications of the sender, medium, and receiver considered here.

Moreover, the present result not only gives an alternative proof of the simple data link and buffer introduced in subsection 5.2.1; it also shows how even the data link not allowing any information to enter and the data link with a very faulty medium (i.e. one that always loses its pulse thus performs an infinite sequence of d-transitions) observationally implement a buffer.

Chapter 6

Proof systems

Wouldn't it be nice if we could give sound and complete proof systems for $\lhd$ and semantic equality ? Indeed it would, and in this chapter we present such proof systems for certain subsets of MPL. The proof systems for $\lhd$ and the soundness thereof turn out to be simple corollaries of results obtained by Bent Thomsen in [Thomsen 87]. The proof system for semantic equality is based on the result of full abstractness in Chapter 4.

6.1 Limitations of the proof systems

Unfortunately there is no sound and complete proof system for $\lhd$ in a modal process logic with general static constructs. For, as already indicated, one can reasonably view $\mathcal{CCS}$ as a subcalculus of MPL and $\sim$ as a special case of $\lhd$, and there exist no sound and complete proof systems for $\sim$ in a $\mathcal{CCS}$ with static constructs. For, using static constructs, one can code any Turing machine M and input string w as a $\mathcal{CCS}$ -expression $p_{M,w}$ such that all moves of M are represented by internal (τ-)actions of $p_{M,w}$ and such that the possible eventual halting is represented by a special success action d. If we had a sound and complete proof system for $\sim$ the problem 'Does M diverge on input w ?' would be r.e., since this would amount to proving

$$p_{M,w} \sim \mu x.(\tau.x) \tag{6.1}$$

A partial decision procedure would then consist in dovetailing through all theorems of the hypothetical proof system until one found (6.1).

However, it turns out that one can give proof systems for subsets of $\mathcal{CCS}$, namely the classes of so-called finite and regular processes. We will here consider the corresponding classes of MPL expressions.

6.2 Proof systems for the refinement order

The proof systems for $\lhd$ are a direct consequence of results achieved by Bent Thomsen [Thomsen 87] on so-called M-bisimulations and proof systems for such relations. The application of these results also sheds light on the nature of the concept of transitions with modalities.

6.2.1 M-bisimulations

M-bisimulations formed the original basis for the language of partial specifications as described in Chapter 4. One simply imposes a preorder relation on the action set, $\sqsubseteq$, such that we for the 'wild' action $*$ have $a \sqsubseteq *$ for $a \in Act$. Based on this, Bent Thomsen gave his general characterization of bisimulation-like orderings on transition systems with a preorder on the action set. There are several intuitive reasons for imposing an ordering on the action set; for instance one could interpret $a \sqsubseteq b$ as 'whatever a can do b can do as well'. Or one could interpret $a \sqsubseteq b$ as 'a is better defined than b' - this is what we do here.

In what follows, we hence assume a transition system with an ordering on the action set: $\mathcal{S}_{\sqsubseteq} = (Sp, (Act, \sqsubseteq), \rightarrow)$. We first define the concept of downwardsclosedness:

Definition 6.1 *A set of actions $M \subseteq Act$ is* downwardsclosed *iff* $\forall a \in M : b \sqsubseteq a \Rightarrow b \in M$

M-bisimulation can now be defined relative to any downwardsclosed set M:

Definition 6.2 ([Thomsen 87]) *A relation $R \subseteq Sp \times Sp$ is an M-*bisimulation *iff whenever pRq we have*

1. $p \xrightarrow{a} p' \Rightarrow \exists q' \exists b : q \xrightarrow{b} q' \wedge ; a \sqsubseteq b \;\wedge\; p'Rq'$
2. $\forall a \in M : q \xrightarrow{a} q' \Rightarrow \exists p' \exists b : p \xrightarrow{a} p' \;\wedge\; b \sqsubseteq a \;\wedge\; p'Rq'$

As usual, it is easy to see that there exists a largest M-bisimulation $\sqsubseteq_M$ for any given downwardsclosed M.

It now turns out that $\lhd$ is an M-bisimulation for a properly chosen M. This should not surprise us, for there is nothing sacred about modal transition systems. Any such system, $\mathcal{S} = (Sp, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$, can be viewed as a transition system with an ordering on the action set.

We translate any modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_{\Diamond}, \rightarrow_{\Box})$ to a transition system with an ordered action set by simply letting an action be an ordered pair also containing a modality. Thus we get the derived transition system $\mathcal{S}_{\sqsubseteq} = (Sp, (Act \times \{\Box, \Diamond\}, \sqsubseteq), \rightarrow)$. The new actions are written with the modality suffixed, and we write $F \overset{a_m}{\rightarrow} F'$ instead of $F \xrightarrow{a}_m F'$ $(m \in \{\Box, \Diamond\})$. We reflect the requirement $\rightarrow_\Box \subseteq \rightarrow_\Diamond$ by demanding that $F \overset{a_\Box}{\rightarrow} F'$ implies $F \overset{a_\Diamond}{\rightarrow} F'$.

To determine the ordering on actions we simply say that, for any given a we have $a_\Box \sqsubseteq a_\Diamond$. Intuitively we could say that $\Box$ -actions are better defined than $\Diamond$ -actions. We then choose as M the $\Box$ -actions, i.e. $M = \{a_\Box | a \in Act\}$. M and $\sqsubseteq$ can be visualized by the lattice diagram in Figure 6.1.

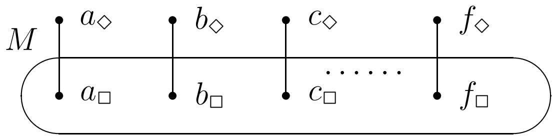

Figure 6.1: The set M and the ordering of actions in our derived transition system

It is obvious that M is downwardsclosed. We now have

Theorem 6.1 *With $M = \{a_\Box \mid a_\Box \in Act \times \{\Box, \Diamond\}\}$ we have that $\sqsubseteq_M$ in the transition system $\mathcal{S}_{\sqsubseteq} = (Sp, (Act \times \{\Box, \Diamond\}), \sqsubseteq), \rightarrow)$ is the same as $\lhd$ in the modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$.*

Proof: We simply show that

$$\{(F, G) \mid F \sqsubseteq_M G\} \tag{6.2}$$

is a refinement and that

$$\{(F, G) \mid F \lhd G\} \tag{6.3}$$

is an M-bisimulation.

For the proof of (6.2), suppose $F \sqsubseteq_M G$ and $F \xrightarrow{a}_\Diamond F'$. This is equivalent to $F \overset{a_\Diamond}{\rightarrow} F'$. By M-bisimulation, we get $G \xrightarrow{b} G'$ and $F' \sqsubseteq_M G'$ with $a_\Diamond \sqsubseteq b$. But this implies $G \overset{a_\Diamond}{\rightarrow} G'$ and thus $G \xrightarrow{a}_\Diamond G'$ with (F', G') in the relation. Conversely, suppose $G \xrightarrow{a}_\Box G'$. This is equivalent to $G \overset{a_\Box}{\rightarrow} G'$ and $a_\Box \in M$.

By M-bisimulation this means $F \xrightarrow{b} F'$ and $F' \sqsubseteq_M G'$ with $b \sqsubseteq a_\Box$. But this implies $F \xrightarrow{a}_\Box F'$ with (F', G') in the relation, and we're done. The proof of (6.3) is similar. □

6.2.2 The proof systems

A main result in [Thomsen 87] is that one can give sound and complete proof systems for $\sqsubseteq_M$. These proof systems, of course, work upon formulae of a language, and in transferring the proof systems to work for MPL, we must relate the languages. I.e., we must be sure that the languages of finite and regular processes in [Thomsen 87] correspond semantically to the languages of finite and regular MPL specifications, respectively. But this is easily seen to be the case, since the structure on actions is nowhere of any importance in the operational semantics given in [Thomsen 87] and since our operational semantics of MPL as presented in Chapter 4 is the same for both modalities, apart from the prefixing axioms (which determine the ordering!),

In the proof systems presented here, we have made a few, necessary syntactic alterations :

$\lhd$	replaces	$\sqsubseteq_M$
$\bowtie$	replaces	$=_M$
rec	replaces	μ
Nil	replaces	nil
$Nil \lhd a_\Diamond.F$	replaces	$\dfrac{b \notin M}{nil \sqsubseteq_M b.p}$

$\bowtie$ is here the equivalence induced by $\lhd$, i.e. $F \bowtie G$ iff $F \lhd G$ and $G \lhd F$.

The rule for precongruence w.r.t. prefixing has become C1:

$$\frac{F_1 \lhd F_2}{a_{m_1}.F_1 \lhd a_\Diamond.F_2} \quad , m_1 \in \{\Box, \Diamond\}$$

And finally we have omitted the rule

$$\frac{p_1 \sqsubseteq_M p_2}{p_1 \sqsubseteq_N p_2}, \quad N \subseteq M$$

since we are only interested in the M-bisimulation corresponding to $\lhd$, where M is as fixed above.

Table 6.1 presents proof systems for $\lhd$ on the finite MPL specifications (i.e. formulae constructed not using rec). Their syntax is

$$E ::= a_\Box.E \mid a_\Diamond.E \mid E_1 + E_2 \mid Nil$$

In Table 6.2 we have the proof system for $\lhd$ for the regular MPL specifications introduced in Chapter 2. Here a necessary observation is that the definition of variable substitution used by [Thomsen 87] is exactly the same as the one used in this report.

6.3 Proof systems for semantic equality

In [Graf and Sifakis 86] Graf and Sifakis present their so-called Synchronization Tree Logic (STL). STL is a hybrid between the usual regular $\mathcal{CCS}$ -processes (Pr) in this report and Hennessy-Milner logic with recursion [Larsen 87] in that it contains the process constructs of the former, the logical connectives of the latter, and a recursion operator, one main difference being that STL allows prefixing with *sets* of guards. Graf and Sifakis gave a denotational description resembling the one presented for MPL in this report (i.e. the meaning of a formula is a set of processes in a process system). They also presented a sound and complete proof system for non-recursive STL formula.

In Chapter 4 we discussed Graf and Sifakis's requirements to a specification language and found that MPL met these requirements. We therefore, tentatively, introduce the disjunction $\vee$ into our language of finite MPL specifications to see what happens. It turns out that we can give a sound and complete proof system for the set of finite (i.e. non-recursive) MPL formulae with disjunction, whose abstract syntax is

$$E ::= a_\Box.E \mid a_\Diamond.E \mid E_1 + E_2 \mid E_1 \vee E_2 \mid Nil$$

Call this language $Sp_\vee$. We abbreviate the disjunction of n components by $\bigvee_{i=1}^{n} F_i$, letting $\bigvee_{i=1}^{1} F_i = F_1$.

We extend the semantic function from Definition 4.3 by simply adding

$$[\![F \vee G]\!]\sigma = [\![F]\!]\sigma \ \cup \ [\![F]\!]\sigma \tag{6.4}$$

This is exactly as in [Graf and Sifakis 86], the intended meaning being that any process satisfying $F \vee G$ satisfies either F or G. The extension of course does not alter the facts of monotonicity and anticontinuity of $[\![\_]\!]$ established in Chapter 4.

The idea of the completeness proof is to show how one can reduce any such formula in $Sp_\vee$ to a conjunction of $\Box$ -formulae. The $\Box$ -formulae form a subset of Sp, $Sp_\Box$, given by the syntax

$$F ::= a_\Box.E \mid E_1 + E_2 \mid Nil$$

From our discussion in Chapter 4 it should be obvious that $Sp_\Box$ can be regarded as a subset of the partial specification language, thus implying

Proposition 6.1 *Whenever $F, G \in Sp_\Box$ we have $F \lhd G \Leftrightarrow [\![F]\!] \subseteq G$.*

It is also pretty obvious that the denotation of any $\Box$ -formula is exactly one equivalence class w.r.t. $\sim$:

Proposition 6.2 $F \in Sp_\Box \Rightarrow [\![F]\!] = \{q \mid q \sim p^\Box(F)\}$

By Proposition 6.1 we can now give a proof system for semantic *inclusion*, $\subseteq$, and thus also for semantic equality, $\equiv$, defined by

Definition 6.3 *We write* $\begin{array}{lll} F \equiv G & \textit{whenever} & [\![F]\!] = [\![G]\!] \\ F \subseteq G & \textit{whenever} & [\![F]\!] \subseteq [\![G]\!] \end{array}$

Essentially the proof system is obtained through taking the proof system for $\lhd$ for finite MPL specifications, replacing $\lhd$ with $\subseteq$ and $\bowtie$ with $\equiv$, extending the system with appropriate rules and axioms for $\vee$ and rules and axioms stating how the process constructs distribute over $\vee$.

The proof system is presented in Table 6.3. The rules D1 and D3 are chosen so that they correspond to the rules D1 and D3 in the proof system presented in [Graf and Sifakis 86].

The soundness of the proof system depends on the following propositions. For proofs, see Appendix.

Proposition 6.3 $(F_1 \vee F_2) + F_3 \equiv (F_1 + F_3) \vee (F_2 + F_3)$

Proposition 6.4 $a_\Diamond.F \equiv a_\Box.F \vee Nil$

Proposition 6.5 $a_\Box.(F_1 \vee F_2) \equiv a_\Box.F_1 \vee a_\Box.F_2 \vee (a_\Box.F_1 + a_\Box.F_2)$

Theorem 6.2 *The proof system in Table 6.3 is sound.*

Proof: Axioms O1-O5 are obviously true by the corresponding properties of set union. The soundness of D1-D3 follows from Propositions 6.3 to 6.5 above. P1-P4 state that set inclusion is a preorder, which is obvious. C1-C4 follow from the monotonicity of the functions $\oplus$, $\lceil -_{\diamond} \rceil$, and $\lceil -_{\Box} \rceil$ as established in Chapter 4. The soundness of S1-S4 follows from the soundness of the proof system in 6.1 by Proposition 6.1. $\Box$

In what follows, we shall write $\vdash F$ whenever the formula F in $Sp_{\vee}$ is provable in the proof system in Table 6.3.

It turns out that we can provably reduce any formula to an equivalent *disjunctive normal form*:

Theorem 6.3 (Disjunctive Normal Form) *For any formula $G \in Sp_{\vee}$, there exist $F_1, \ldots, F_n \in Sp_{\Box}$ such that*

$$\vdash G \equiv \bigvee_{i=1}^{n} F_i$$

Proof: See Appendix. $\Box$

(This should really not surprise us; we know that the $\Box$ -formulae represent single equivalence classes, and in Theorem 4.1 and Corollary 4.1 we saw that the denotation of any MPL formula is a union of equivalence classes.)

Semantic equality of such normal forms falls back on the semantic equality of relevant components therein:

Proposition 6.6

$$\bigvee_{i=1}^{m} f_i \equiv \bigvee_{j=1}^{n} g_j$$

if and only if we have

$$\forall j \leq n \exists i \leq m : f_i \equiv g_j, \forall i \leq m \exists j \leq n : g_j \equiv f_i$$

where $\forall i \leq m : f_i \in Sp_{\Box}$ and $\forall j \leq n : g_j \in Sp_{\Box}$.

Proof: See Appendix. $\Box$

Conversely, semantic equality of normal forms can be proved if semantic equality of relevant components can be proved:

Proposition 6.7 *If $\forall i \leq m \exists j \leq n : \vdash F_i \equiv G_j$ and $\forall j \leq n \exists i \leq m : \vdash F_i \equiv G_j$ we have*

$$\vdash \bigvee_{i=1}^{m} F_i \equiv \bigvee_{j=1}^{n} G_j$$

PROOF: See Appendix. $\square$

This implies the completeness of the proof system:

Theorem 6.4 *The proof system in Table 6.3 is complete.*

PROOF: Suppose $F \equiv G$. We must show that $\vdash F \equiv G$. By Theorem 6.3 we have

$$\vdash F \equiv \bigvee_{i=1}^{m} F_i \quad \text{and} \quad \vdash G \equiv \bigvee_{i=1}^{n} G_i$$

with all $F_i, G_j \in Sp_\square$. Theorem 6.2 now states that

$$F \equiv \bigvee_{i=1}^{m} F_i \quad \text{and } G \equiv \bigvee_{i=1}^{n} G_i$$

But by Proposition 6.6 this means that for all j there is an i such that $F_i \equiv G_j$ and that for all i there is a j such that $G_j \equiv F_i$. For $\square$ -formulae this is always the same as showing for any i that there is a j such that $F_i \bowtie G_j$ and for any j that there is a i such that $F_i \bowtie G_j$. The way we constructed our proof system, we have embedded a translation of the proof system for $\lhd$ and $\bowtie$ on finite MPL-formulae in it, so these equivalences over $\bowtie$ are provable. Thus for any i we have a j such that $\vdash F_i \equiv G_j$ and for any j we have an i such that $\vdash F_i \equiv G_j$. By Proposition 6.7 we then have $\vdash \bigvee_{i=1}^{m} F_i \equiv \bigvee_{j=1}^{m} G_j$, completing the proof. $\square$

Example 6.1 Remember Example 4.1 ? Now we can use our proof system in Table 6.3 to show that

$$a_\diamond.b_\diamond.Nil \equiv a_\diamond.b_\square.Nil + a_\diamond.Nil$$

With reference to the rules and axioms in the proof system we get the reductions

$$\begin{array}{rcl} F & \equiv & a_\diamond.b_\diamond.Nil \\ \text{(D2)} & \equiv & a_\square.b_\diamond.Nil \vee Nil \\ \text{(D2)} & \equiv & a_\square.(b_\square.Nil \vee Nil) \vee Nil \\ \text{(D3)} & \equiv & a_\square.b_\square.Nil \vee a_\square.Nil \vee (a_\square.b_\square.Nil + a_\square.Nil) \vee Nil \end{array}$$

and

$$\begin{array}{rcl}
G & \equiv & a_\diamond.b_\Box.Nil + a_\diamond.Nil \\
\text{(D2)} & \equiv & a_\Box.b_\Box.Nil \vee Nil + (a_\Box.Nil \vee Nil) \\
\text{(D1)} & \equiv & (a_\Box.b_\Box.Nil + (a_\Box.Nil \vee Nil)) \vee (Nil + (a_\Box.Nil \vee Nil)) \\
\text{(D1,S4)} & \equiv & (a_\Box.b_\Box.Nil + a_\Box.Nil) \vee a_\Box.b_\Box.Nil \vee Nil \vee a_\Box.Nil \vee Nil \\
\text{(O1)} & \equiv & a_\Box.b_\Box.Nil \vee a_\Box.Nil \vee (a_\Box.b_\Box.Nil + a_\Box.Nil) \vee Nil
\end{array}$$

□

Also notice that the disjunctive normal form obtained in Theorem 6.3 together with Proposition 6.6 in principle yield a *decision procedure* for $\equiv$ if one exists for $\lhd$. For we can, given any two formulae in $Sp_\vee$, F and G, obtain their normal forms and for all i check if there is a j such that $F_i \bowtie G_j$ for some pair of terms in the disjunctive normal form. This can be done through the repeated use of an algorithm for $\lhd$. In the same way we then for all j test for all pairs if there is an i such that $F_i \bowtie G_j$ for a pair of normal form terms. Unfortunately, one can see from the proof of Theorem 6.3 that the number of terms in the disjunctive normal form is roughly exponential in the number of $\vee$'s in the original formulae, so the decision algorithm sketched here is inherently exponential.

SUM	S1	$F_1 + (F_2 + F_3) \bowtie (F_1 + F_2) + F_3$
	S2	$F_1 + F_2 \bowtie F_2 + F_1$
	S3	$F_1 + F_1 \bowtie F_1$
	S4	$F_1 + Nil \bowtie F_1$
PREORD	P1	$F \lhd F$
	P2	$\dfrac{F_1 \lhd F_2, F_2 \lhd F_3}{F_1 \lhd F_3}$
	P3	$\dfrac{F_1 \lhd F_2, F_2 \lhd F_1}{F_1 \bowtie F_2}$
	P4	$\dfrac{F_1 \bowtie F_2}{F_1 \lhd F_2, F_2 \lhd F_1}$
PRECONG	C1	$\dfrac{F_1 \lhd F_2}{a_{m_1}.F_1 \lhd a_\Diamond.F_2} \quad , \ {\scriptstyle m_1 \in \{\Box, \Diamond\}}$
	C2	$\dfrac{F_1 \lhd F_2}{a_\Box.F_1 \lhd a_\Box.F_2}$
	C3	$\dfrac{F_1 \lhd F_2, F_3 \lhd F_4}{F_1 + F_3 \lhd F_2 + F_4}$
PREFIX	PF1	$Nil \lhd a_\Diamond.F$

Table 6.1: Proof system for $\lhd$ on finite MPL specifications

SUM	S1	$F_1 + (F_2 + F_3) \bowtie (F_1 + F_2) + F_3$
	S2	$F_1 + F_2 \bowtie F_2 + F_1$
	S3	$F_1 + F_1 \bowtie F_1$
	S4	$F_1 + Nil \bowtie F_1$
PREORD	P1	$F \lhd F$
	P2	$\dfrac{F_1 \lhd F_2, F_2 \lhd F_3}{F_1 \lhd F_3}$
	P3	$\dfrac{F_1 \lhd F_2, F_2 \lhd F_1}{F_1 \bowtie F_2}$
	P4	$\dfrac{F_1 \bowtie F_2}{F_1 \lhd F_2, F_2 \lhd F_1}$
PRECONG	C1	$\dfrac{F_1 \lhd F_2, F_3 \lhd F_4}{F_1[F_3/x] \lhd F_2[F_4/x]}$
	C2	$\dfrac{F_1 \lhd F_2}{\mathrm{rec}x.F_1 \lhd \mathrm{rec}x.F_2}$
REC	R1	$\mathrm{rec}x.F \bowtie F[\mathrm{rec}x.F/x]$
	R2	$\mathrm{rec}x.F \bowtie \mathrm{rec}y.F[y/x], \quad y \notin FV_{Sp}(F)$
	R3	$\mathrm{rec}x.(F + x) \bowtie \mathrm{rec}x.F$
	R4	$\dfrac{F_1[F_2/x] \lhd F_2}{\mathrm{rec}x.F_1 \lhd F_2}$, x guarded in F_1
	R5	$\dfrac{F_2 \lhd F_1[F_2/x]}{F_2 \lhd \mathrm{rec}x.F_1}$
PREFIX	PF1	$a_\Box.F \lhd a_\Diamond.F$
	PF2	$Nil \lhd a_\Diamond.F$

Table 6.2: Proof system for $\lhd$ on regular MPL specifications

OR	O1	$F \vee F \equiv F$
	O2	$F_1 \vee F_2 \equiv F_2 \vee F_1$
	O3	$F_1 \vee (F_2 \vee F_3) \equiv (F_1 \vee F_2) \vee F_3$
	O4	$F_1 \subseteq F_1 \vee F_2$
	O5	$\dfrac{F_1 \subseteq F_2, F_3 \subseteq F_4}{F_1 \vee F_3 \subseteq F_2 \vee F_4}$
DIST	D1	$(F_1 \vee F_2) + F_3 \equiv (F_1 + F_3) \vee (F_2 + F_3)$
	D2	$a_\Diamond.F \equiv a_\Box.F \vee Nil$
	D3	$a_\Box.(F_1 \vee F_2) \equiv a_\Box.F_1 \vee a_\Box.F_2 \vee (a_\Box.F_1 + a_\Box.F_2)$
SUM	S1	$F_1 + (F_2 + F_3) \equiv (F_1 + F_2) + F_3 \;\; , {\scriptstyle F_1, F_2 \in Sp_\Box}$
	S2	$F_1 + F_2 \equiv F_2 + F_1 \;\; , {\scriptstyle F_1, F_2 \in Sp_\Box}$
	S3	$F + F \equiv F \;\; {\scriptstyle F \in Sp_\Box}$
	S4	$F + Nil \equiv F \;\; {\scriptstyle F \in Sp_\Box}$
PREORD	P1	$F \subseteq F$
	P2	$\dfrac{F_1 \subseteq F_2, F_2 \subseteq F_3}{F_1 \subseteq F_3}$
	P3	$\dfrac{F_1 \subseteq F_2, F_2 \subseteq F_1}{F_1 \equiv F_2} \qquad ,$
	P4	$\dfrac{F_1 \equiv F_2}{F_1 \subseteq F_2, F_2 \subseteq F_1}$
PRECONG	C1	$\dfrac{F_1 \subseteq F_2}{a_{m_1}.F_1 \subseteq a_\Diamond.F_2} \quad , \; {\scriptstyle m_1 \in \{\Box, \Diamond\}}$
	C2	$\dfrac{F_1 \subseteq F_2}{a_\Box.F_1 \subseteq a_\Box.F_2}$
	C3	$\dfrac{F_1 \subseteq F_2, F_3 \subseteq F_4}{F_1 + F_3 \subseteq F_2 + F_4}$
PREFIX	PF1	$Nil \subseteq a_\Diamond.F$

Table 6.3: Proof system for $\equiv$ on finite MPL specifications with $\vee$

Chapter 7

Conclusion

We end the report with a summary of the main results presented in this report, discuss their consequences, and present some ideas for further work.

7.1 Summary of main results

- In Chapter 2 we introduced the notions of modal transition system and refinements, noting that a largest refinement $\lhd$ exists and that it is a preorder.

- In Chapter 3 we introduced the Modal Process Logic, giving its syntax and structural operational semantics.

- In Chapter 4 we presented a denotational description of MPL,$[\![\_]\!]$, stating that the denotation of an MPL formula is a set of $\mathcal{CCS}$ -processes. Under certain reasonable conditions we showed that the denotation of any formula was the union of some equivalence classes of processes w.r.t. $\sim$. We also gave a limiting condition stating how far the full abstractness result of [Larsen and Thomsen 88] can be extended. The original result states that

  $$F \lhd G \Rightarrow [\![F]\!] \subseteq [\![G]\!] \tag{7.1}$$

  Counterexamples showed that we could not obtain the reverse direction of (7.1). However, we could characterize a class of counterexamples through a relation $\not\lhd_D$ having the property that

  $$F \not\lhd_D G \Rightarrow [\![F]\!] \not\subseteq [\![G]\!] \tag{7.2}$$

We also showed how the partial specification language from [Thomsen 87] and [Larsen and Thomsen 87] can be viewed as a subset of MPL and how we in fact for this subset of MPL have

$$F \lhd G \Leftrightarrow [\![F]\!] \subseteq [\![G]\!] \tag{7.3}$$

- In Chapter 5 we introduced the static process constructs of $\mathcal{CCS}$ into MPL, considering both parallel composition and action restriction and static constructs in general. This gave rise to the definition of an observational refinement ordering, $\unlhd$. We gave three examples of the usefulness of this preorder.

- In Chapter 6 we used results from [Thomsen 87] in seeing how modal transition systems could be seen as just ordinary transition systems with a preorder on the action set. $\lhd$ is then just a special case of the M-bisimulation $\sqsubseteq_M$ of [Thomsen 87]. Based on this, we transferred the general proof systems of $\sqsubseteq_M$ to work for $\lhd$. Relating MPL to the STL in [Graf and Sifakis 86], we tentatively introduced disjunction ($\vee$) into MPL and showed how one, based on the proof systems for $\lhd$ and the full abstractness result for partial specifications, could obtain a sound and complete proof system for semantic equality in a subset of MPL without recursion.

7.2 Discussion

The above results should of course be put in a wider perspective.

7.2.1 Full abstractness

It would be nice if we could characterize operationally exactly when $[\![F]\!] \subseteq [\![G]\!]$ based on knowledge of F and G. As seen by the non-existence of the converse of (7.1) and the fact that the negation of $\not\lhd_D$ is *not* $\lhd$, this may be difficult. Nevertheless (7.2) sheds light on a large class of counterexamples.

7.2.2 How to use $\Box$ and $\Diamond$

The three examples in Chapter 5 not only showed the usefulness of static constructs in MPL seen in connection with $\unlhd$; they also showed a possible guidelines for using the modalities on the transitions:

One can use the modalities to distinguish between the essential parts and the exception-handling parts of a system. The transitions *raising* exceptions are marked with $\Diamond$'s , whereas the exception-*handling* transitions are marked with $\Box$'s. This idea is similar to one found in [Bækgaard et al 86], where it was claimed that, for reasons of modularity, a programming language should enforce the separation of the descriptions of exceptional and normal program states.

7.2.3 Limitations of MPL

Another thing shown through the examples in Chapter 5 is that MPL should be regarded as a modal process *algebra* rather than as a logic. The proof techniques used and the orderings $\lhd$ and $\unlhd$ are based on ideas derived from the theory behind $\mathcal{CCS}$, relying heavily on structural operational semantics.

In Chapter 5 we saw that the theory of MPL provided a specification language stronger than $\mathcal{CCS}$. One might now wonder if MPL is as strong as Hennessy-Milner logic w.r.t. the compositionality introduced with the concept of process contexts, namely viewing contexts as MPL property transformers as was done in [Larsen 86] and [Godskesen et al 87]. Can we define an MPL property transformation I_C such that we for any MPL formula F and $\mathcal{CCS}$ -context $C[\ ]$ have that $p \lhd I_C(F)$ iff $C[p] \lhd F$? Unfortunately, the answer is no.

For suppose $F = Nil$. Then there are contexts such that $C[p] \not\lhd Nil$, since a $\mathcal{CCS}$ -context may be just a process expression of $\mathcal{CCS}$. Suppose we ruled out such 'no-hole' contexts. Then we might still always have $C[p] \not\lhd Nil$ - consider $C[\ ] = a.[\ ]$. This would imply that the denotation of $I_{a.[\ ]}(Nil)$ would be the empty set of processes, something which cannot be expressed in MPL. It is well known that this is possible in Hennessy-Milner logic by the presence of *ff*. One could of course introduce a *ff*-like construct in MPL but it is rather hard to imagine what its operational interpretation (if any) might be.

In this sense MPL resembles $\mathcal{CCS}$, for the same arguments can be used to show the impossibility of such a property transform for $\mathcal{CCS}$. As a matter of fact, one sees from the canonical implementation transforms $p^{\Box}(F)$ and $p^{\Diamond}(F)$ from Chapter 4, that *any* MPL specification has an implementation. Thus it is not possible to describe specifications that cannot be implemented; some feel that a specification language should be strong enough to allow such 'absurd' specifications.

One could try extending the expressive power of MPL by adding genuine logical connectives. However, the introduction of the Boolean connectives as tried out in Chapter 6 did not come with an augmented operational description. This is due to the simple fact that neither the author nor his supervisor could come up with an operational interpretation of $\vee$. On the other hand, the introduction of process constructs into Hennessy-Milner logic as tried out by Sören Holmström in [Holmström 87] resulted in an *intensional* denotational description, putting constraints on the structure of the processes to be found in a denotation. This is not a very convincing way in which to combine process calculus and modal logic features.

Still, by the denotational description in Chapter 4 we can relate the modalities of MPL to those of Hennessy-Milner logic as follows:

$$\begin{aligned} a_{\Diamond}.F &\equiv [a]F \wedge \bigwedge_{b \neq a} [b]\mathit{ff} \\ a_{\Box}.F &\equiv \langle a \rangle \mathit{tt} \wedge [a]F \wedge \bigwedge_{b \neq a} [b]\mathit{ff} \end{aligned}$$

Here $\equiv$ is semantic equality w.r.t. sets of processes. Alternatively we could describe the Hennessy-Milner modalities in MPL as

$$\begin{aligned} \langle a \rangle F &\equiv a_{\Box}.F + \mathcal{U} \\ [a]F &\equiv a_{\Diamond}.F + \sum_{b \neq a} b_{\Diamond}.\mathcal{U} \end{aligned}$$

It seems to the author that there is a wide conceptual gap between logic and process algebra whose crossings are bound to result in certain deficiencies. The best thing is then to stick to the MPL which can be seen as an extension of $\mathcal{CCS}$ for then we get an operationally defined, intuitively simple ordering on specifications, properties of which can be proved using Park's Induction Principle. If one is to specify some system and does not find MPL to be sufficiently expressive, one should use a modal logic (e.g. Hennessy-Milner logic). But in general, one should of course choose whatever one feels is the most appropriate formalism.

7.3 Ideas for further work

There is of course still a lot of work to be done in the theory of the modal process logic (or modal process algebra). As we saw in Chapter 6, modal transition systems could be viewed as just another case of an imposed structure on the action set, a topic dealt with in greater detail in [Thomsen 87]. It seems as if the development of a theory of MPL could take advantage of this, for the idea of a structure on the action set is indeed very general. So further work on a theory of transition systems with a structure on the action set is indeed important.

It would also be of great significance if we could give a truly domain-theoretic characterization of $\lhd$ (and the M-bisimulations in general), along the lines of [Abramsky 88] in which Abramsky shows how to give a fully abstract denotational description of a so-called 'bisimulation with divergence'. Such a characterization would shed even more light on the nature of these relations.

On a more practical note, we need to examine more examples of specifications and proofs using MPL, $\lhd$ and $\unlhd$. In more realistic settings a treatment of value-passing is then inevitable.

And of great use in such verifications would be an automated decision procedure for $\lhd$ implemented along the lines of the PROLOG system in [Larsen 86].

Appendix A

Appendix

In this appendix we give detailed proofs of theorems and lemmae stated without proofs in the main part of this report.

A.1 Proofs from Chapter 2

We sometimes need to characterize $\lhd$ as the limit of a chain of approximating relations. This holds, when the modal transition system considered, $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$, is *image-finite*. This implies the anti-continuity of the functional $\mathcal{RE}$.

Definition A.1 *(Definition 2.9)*
A monotonic functional $f : A \rightarrow A$, where A is a complete lattice is anti-continuous iff we for all decreasing chains $\{B_i\}$ have that

$$f(\sqcap_{i=1}^{\infty}(B_i)) = \sqcap_{i=1}^{\infty} f(B_i)$$

Lemma A.1 *(Lemma 2.1)*
Let A be a complete lattice with a maximal element, $\top$ and let the functional $f : A \rightarrow A$ be anti-continuous. Then f has a maximal fixed point given by

$$\text{FIX } f = \sqcap_{i=0}^{\infty} f^i(\top)$$

Proof: FIX f is a *fixed point* of f, since

$$f(\text{FIX } f) = f(\sqcap_{i=0}^{\infty} f^i(\top)) = \sqcap_{i=1}^{\infty} f^i(\top) = \sqcap_{i=0}^{\infty} f^i(\top)$$

(The last equality follows from the maximality of $\top$)

FIX f is *maximal*, since any other fixed point g by monotonicity of f satisfies

$$\forall i : f^i(g) = g \sqsubseteq f^i(\text{FIX } f)$$

implying that $g \sqsubseteq \text{FIX } f$. $\square$

Theorem A.1 *(Theorem 2.1)*
If the modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ is image-finite, the refinement function $\mathcal{RE}$ is anti-continuous.

Proof: Suppose we have a decreasing chain in $(\mathbf{2}^{Sp\times Sp}, \subseteq)$: $B_0 \supseteq B_1 \supseteq B_2 \ldots$. We want to show that

$$\mathcal{RE}(\bigcap_{i=0}^{\infty} B_i) = \bigcap_{i=0}^{\infty} \mathcal{RE}(B_i)$$

Since $\mathcal{RE}$ is monotonic, the $\subseteq$-direction follows. Suppose $(F, G) \in \mathcal{RE}(\cap_{i=0}^{\infty} B_i)$. Then for all i there is a B_i such that

$$F \xrightarrow{a}_\Diamond F' \Rightarrow \exists G' : G \xrightarrow{a}_\Diamond G' \wedge (F', G') \in B_i$$

But since $\rightarrow_\Diamond$ is image-finite and $\{B_i\}$ is decreasing, some (F', G') must belong to infinitely many B_i and thus to $\cap_{i=0}^{\infty} B_i$. Conversely, we have

$$G \xrightarrow{a}_\Box G' \Rightarrow \exists F' : F \xrightarrow{a}_\Diamond F' \wedge (F', G') \in B_i$$

and by the same argument that some $(F', G') \in \cap_{i=0}^{\infty} B_i$, altogether implying the $\supseteq$-direction.
$\square$

Corollary A.1 *(Corollary 2.3)*
When the modal transition system $\mathcal{S} = (Sp, Act, \rightarrow_\Diamond, \rightarrow_\Box)$ is image-finite, we have that

$$\lhd\ = \bigcap_{i=0}^{\infty} \mathcal{RE}^i(Sp)$$

A.2 Proofs from Chapter 3

With the following lemma it is easy to infer the behaviour of an MPL expression obtained using substituion.

Lemma A.2 *(Lemma 3.1)*
Whenever x is guarded in $G \in Sp$ we have

$$G[F/x] \xrightarrow{a}_m E$$

if and only if

$$G \xrightarrow{a}_m G' \text{ and } E = G'[F/x]$$

for some G' (where $\mathcal{M} \in \{\Diamond, \Box\}$)

PROOF:
Only if: Induction in the length n of inferences establishing $G[F/x] \xrightarrow{a}_m E$ using the operational rules and axioms.

$n = 1$: We must have used one of the axioms for prefixing. We then have $G[F/x] = a_\Box.E$ or $G[F/x] = a_\Diamond.E$. Since x is assumed to be guarded in G this means that $G = a_\Box.G'$ or $G = a_\Diamond.G'$ and $E = G'[F/x]$. This implies the desired conclusion.

Assuming for $n = k$: The last rule used in the inference was either the rule for $+$ or the rule for rec.

Suppose we used the $+$-rule. Then $G[F/x] = E_1 + E_2$ and thus $G = G_1 + G_2$ with $E_1 = G_1[F/x]$ and $E_2 = G_2[F/x]$. x must be guarded in both G_1 and G_2. By induction hypothesis we then have $G[F/x] \xrightarrow{a}_m E$ if $E_1 \xrightarrow{a}_m E$ or $E_2 \xrightarrow{a}_m E$, $E = G_1[F/x]$ or $E = G_2[F/x]$ for some G'_1 or G'_2.

Suppose we used the rec-rule. Then $G[F/x] = \text{rec}z.E_1$, meaning that G must be of the form $\text{rec}y.G_1$. This can arise in three different ways, by the substitution rule.

1. $G = \text{rec}x.E_1$: Then $G[F/x] \xrightarrow{a}_m E$ implies by the rec-rule

$$E_1[\text{rec}x.E_1/x][F/x] \xrightarrow{a}_m E$$

But $E_1[\text{rec}x.E_1/x][F/x] = E_1[\text{rec}x.E_1/x]$ implying, again by the rec-rule that $\text{rec}x.E_1 \xrightarrow{a}_m E$, as desired.

2. $G = \mathrm{rec}y.G_1$ with $y \notin FV_{Sp}(F)$: Then $E_1 = G_1[F/x]$. We used the rec-rule (as hypothesized) based on the information that

$$G_1[F/x][\mathrm{rec}y.G_1[F/x]/y] \xrightarrow{a}_m E$$

This, by induction hypothesis, yields the result desired, since we in this case have

$$G_1[F/x][\mathrm{rec}y.G_1[F/x]/y] = G_1[\mathrm{rec}y.G_1/y][F/x]$$

3. $G = \mathrm{rec}y.G_1$ with $y \in FV_{Sp}(F)$: Now $G[F/x] = (\mathrm{rec}z.(G_1[z/y])[F/x])$ for a z, $z \notin FV_{Sp}(F)$. We used the rec-rule (as hypothesized) knowing that

$$((G_1[z/y])[F/x])[(\mathrm{rec}z.(G_1[z/y][F/x])/z] \xrightarrow{a}_m E$$

But we have

$$((G_1[z/y])[F/x])[(\mathrm{rec}z.(G_1[z/y][F/x])/z] = (G_1[(\mathrm{rec}z.G_1[z/y])/y])[F/x]$$

again giving the desired result, since we consider MPL formulae equal up to a renaming of bound variables.

If : Induction in the length of inferences establishing $G \xrightarrow{a}_m G'$ using the operational rules and axioms.

$n = 1$: We must have used the prefixing-rules. This means $G = a_\Box.G'$ or $G = a_\Diamond.G'$. By the substitution rules we have $a_\Box.G'[F/x] \xrightarrow{a}_\Box G'[F/x]$ or $a_\Box.G'[F/x] \xrightarrow{a}_\Diamond G'[F/x]$ or $a_\Box.G'[F/x] \xrightarrow{a}_\Diamond G'[F/x]$, in all cases resulting in the desired conclusion.

***Assuming for $n = k$*:** Again the last rule used in the inference was either the rule for $+$ or the rule for rec.

Suppose we used the rule for $+$. Then $G = G_1 + G_2$ and either $G_1 \xrightarrow{a}_m G'$ or $G_2 \xrightarrow{a}_m G'$. Substitution yields $G[F/x] = G_1[F/x] + G_2[F/x]$. By induction hypothesis we then get $G[F/x] \xrightarrow{a}_m G'$.

Suppose we used the rec-rule. This means $G = \mathrm{rec}y.G_1$. There are again three different cases to be considered.

1. $G = \mathrm{rec}x.G_1$: Then $G[F/x] = G$. The last inference was then based on the information that

$$G_1[\mathrm{rec}x.G_1/x] \xrightarrow{a}_m E$$

assuming, by hypothesis,

$$G_1[\mathrm{rec}x.G_1/x] \xrightarrow{a}_m E[F/x]$$

which by the rec-rule gives the desired result.

2. $G = \mathrm{rec}y.G_1$ *with* $y \notin FV_{Sp}(F)$*:* Then $G[F/x] = \mathrm{rec}y.G_1[F/x]$ and the information concluding the inference was then

$$(G_1[\mathrm{rec}y.G_1/y])[F/x] \xrightarrow{a}_m E[F/x]$$

giving us

$$(G_1[F/x])[\mathrm{rec}y.G_1[F/x]/y] \xrightarrow{a}_m E[F/x]$$

(since y is not free in F) which again yields the result by the rec-rule.

3. $G = \mathrm{rec}y.G_1$ *with* $y \in FV_{Sp}(F)$*:* Then $G[F/x] = \mathrm{rec}z.((G_1[z/y])[F/x])$ for a new z, $z \notin FV_{Sp}(F)$. The information yielding the inference was here

$$(G_1[\mathrm{rec}y.G_1/y])[F/x] \xrightarrow{a}_m E[F/x]$$

implying by the substitution rules that

$$(G_1[z/y])[\mathrm{rec}z.G_1[z/y]/z][F/x] \xrightarrow{a}_m E[F/x]$$

in turn leading to

$$(\mathrm{rec}y.G_1)[F/x] \xrightarrow{a}_m E[F/x]$$

since we consider MPL expressions equal up to the renaming of bound variables.

□

A.3 Proofs from Chapter 4

Definition A.2 *(Definition 4.1)*
The semantic algebra consists of the functions

$$\begin{aligned} nil:&\ \_ \rightarrow \mathbf{2}^{Pr} \\ \lceil \_{\diamond} \rceil:&\ Act \times \mathbf{2}^{Pr} \rightarrow \mathbf{2}^{Pr} \\ \lceil \_{\Box} \rceil:&\ Act \times \mathbf{2}^{Pr} \rightarrow \mathbf{2}^{Pr} \\ \_ \oplus \_:&\ \mathbf{2}^{Pr} \times \mathbf{2}^{Pr} \rightarrow \mathbf{2}^{Pr} \end{aligned}$$

defined by

$$\begin{aligned} \text{nil} &= \{p \mid p \sim nil\} \\ \lceil a_\diamond \rceil U &= \{p \mid p \xrightarrow{b} p' \Rightarrow (b = a \wedge p' \in U)\} \\ \lceil a_\Box \rceil U &= \{p \mid p \xrightarrow{a} \ \wedge \ p \in \lceil a_\diamond \rceil U\} \\ U \oplus V &= \{p \mid \exists p_1, p_2 . p_1 \in U \ \wedge \ p_2 \in V \ \wedge \ p \sim p_1 + p_2\} \end{aligned}$$

where $U, V \subseteq Pr$.

Lemma A.3 *(Lemma 4.1)*
The functions of Definition A.2 are monotonic and anticontinuous on the Boolean lattice $(\mathbf{2}^{Pr}, \subseteq)$

Proof: Monotonicity is obvious. Anticontinuity is shown below (one half is already given by monotonicity):

$\_ \oplus \_$: Since $\oplus$ clearly is commutative, we only need to show anti-continuity in one argument, i.e. we show that we for a decreasing chain $\{B_i\}$ for all $U \subseteq Sp$ have

$$U \oplus \bigcap_i B_i = \bigcap_i (U \oplus B_i)$$

But by the definition of $\oplus$, we have

$$U \oplus \bigcap_i B_i = \bigcup_{\substack{p_U \in U \\ p_B \in \cap_i B_i}} |p_U + p_B|$$

where $|p|$ denotes the equivalence class of p w.r.t. $\sim$. We also have

$$\bigcap_i (U \oplus B_i) = \bigcap_i \bigcup_{\substack{p_U \in U \\ p_{B_i} \in B_i}} |p_U + p_{B_i}|$$

Since equivalence classes are either identical or disjoint, the result follows.

$\lceil \_\diamond \rceil$: Consider again a decreasing chain $\{B_i\}$. We then have

$$\lceil a_\diamond \rceil \bigcap_i B_i = \{p \mid p \xrightarrow{b} p' \Rightarrow b = a \wedge p' \in \bigcap_i B_i\}$$

and

$$\bigcap_i \lceil a_\diamond \rceil B_i = \{p \mid \forall i : p \xrightarrow{b} p' \Rightarrow b = a \wedge p' \in B_i\}$$

which are seen to coincide.

$\lceil \_\Box \rceil$: Again the decreasing chain $\{B_i\}$ comes in, giving

$$\lceil a_\Box \rceil \bigcap_i B_i = \{p \mid p \xrightarrow{a} \wedge\, p \xrightarrow{b} p' \Rightarrow b = a \wedge p' \in \bigcap_i B_i\}$$

and

$$\bigcap_i \lceil a_\diamond \rceil B_i = \{p \mid \forall i : p \xrightarrow{a} \wedge\, p \xrightarrow{b} p' \Rightarrow b = a \wedge p' \in B_i\}$$

which coincide.

$\Box$

We need two canonical implementations in the proof of Theorem 4.3:

Definition A.3 *(Definition 4.7)*
The syntactic transformation $p^\Box : Sp \to Pr$ is defined structurally by

$$\begin{array}{rcl}
p^\Box(Nil) & = & nil \\
p^\Box(a_\Box.E) & = & a.p^\Box(E) \\
p^\Box(a_\diamond.E) & = & nil \\
p^\Box(E_1 + E_2) & = & p^\Box(E_1) + p^\Box(E_2) \\
p^\Box(\mathrm{rec}x.E) & = & \mu x.p^\Box(E) \\
p^\Box(x) & = & x
\end{array}$$

$p^\Box$ distributes w.r.t. substitution in the following way:

Lemma A.4 *(Lemma 4.2) For all $S, T \in Sp$:*

$$p^\Box(S[T/x]) = p^\Box(S)[p^\Box(T)/x]$$

Proof: Structural induction on S.

$S = Nil$**:**

$$p^{\Box}(Nil[T/x]) = p^{\Box}(Nil) = nil$$

and

$$p^{\Box}(Nil)[p^{\Box}(T)/x] \;=\; nil[p^{\Box}(T)/x] \;=\; nil$$

$S = a_{\Diamond}.S'$**:**

$$p^{\Box}((a_{\Diamond}.S')[T/x]) = nil$$

and

$$p^{\Box}(a_{\Diamond}.S')[p^{\Box}(T)/x] = nil$$

.

$S = a_{\Box}.S'$**:**

$$p^{\Box}((a_{\Box}.S')[T/x]) = a.p^{\Box}(S')[p^{\Box}(T)/x]$$

and, by hypothesis,

$$p^{\Box}(a_{\Box}.S')[p^{\Box}(T)/x] = a.p^{\Box}(S')[p^{\Box}(T)/x]$$

$S = S_1 + S_2$**:**

$$\begin{array}{rcl} p^{\Box}((S_1 + S_2)[T/x]) & = & p^{\Box}(S_1[T/x]) + p^{\Box}(S_2[T/x]) \\ & = & p^{\Box}(S_1)[p^{\Box}(T)/x] + p^{\Box}(S_2)[p^{\Box}(T)/x] \\ & = & (p^{\Box}(S_1) + p^{\Box}(S_2))[p^{\Box}(T)/x] \\ & = & p^{\Box}(S)[p^{\Box}(T)/x] \end{array}$$

$S = \mathrm{rec}y.S_1$**:** The substitution rule here gives us three subcases:

1. $G = \mathrm{rec}x.G_1$*:* Then we have

$$p^{\Box}((\mathrm{rec}x.S_1)[T/x]) = p^{\Box}(\mathrm{rec}x.S_1) = \mu x.p^{\Box}(S_1)$$

and

$$\begin{array}{rcl} p^{\Box}(\mathrm{rec}x.S_1)[p^{\Box}(T)/x] & = & (\mu x.p^{\Box}(S_1))[p^{\Box}(T)/x] \\ & = & \mu x.p^{\Box}(S_1) \end{array}$$

2. $G = \text{rec}y.G_1$ *with* $y \notin FV_{Sp}(F)$*:* We have

$$\begin{array}{rcl} p^{\Box}((\text{rec}y.S_1)[T/x]) & = & p^{\Box}(\text{rec}y.S_1[T/x]) \\ & = & \mu y.p^{\Box}(S_1[T/x]) \\ & = & \mu y.(p^{\Box}(S_1)[p^{\Box}(T)/x]) \\ & = & (\mu y.p^{\Box}(S_1))[p^{\Box}(T)/x] \\ & = & p^{\Box}(S)[p^{\Box}(T)/x] \end{array}$$

3. $G = \text{rec}y.G_1$ *with* $y \in FV_{Sp}(F)$*:* Then

$$\begin{array}{rcl} p^{\Box}((\text{rec}y.S_1)[T/x]) & = & p^{\Box}(\text{rec}z.(S_1[z/y])[T/x]) \\ & = & \mu z.p^{\Box}((S_1[z/y])[T/x]) \\ & = & \mu z.(p^{\Box}(S_1)[z/y])[p^{\Box}(T)/x] \\ & = & p^{\Box}(S)[p^{\Box}(T)/x] \end{array}$$

The last equation follows, since we consider formulae equal up to renaming of bound variables.

□

Lemma A.5 *(Lemma 4.3)*
For all closed $E \in Sp$ we have $p^{\Box}(E) \in [\![E]\!]$

PROOF: It is enough to show that $p^{\Box}(E) \lhd E$. This, in turn, follows directly from

$$p^{\Box}(E) \xrightarrow{a} \Leftrightarrow \exists E' : E \xrightarrow{a}_{\Box} E' \wedge q = p^{\Box}(E')$$

The proof is done by structural induction on E:

$E = Nil$**:** Trivial.

$E = a_{\Diamond}.E_1$**:** Trivial, since neither $p^{\Box}(E)$ nor E has any $\Box$ -transitions.

$E = a_{\Box}.E_1$**:** The only transition of $p^{\Box}(E)$ is $p^{\Box}(E) \xrightarrow{a} p^{\Box}(E_1)$. The only $\Box$ -transition of E is $E \xrightarrow{a}_{\Box} E_1$, and we're done.

$E = E_1 + E_2$**:** $p^{\Box}(E_1 + E_2) \xrightarrow{a} q'$ iff $p^{\Box}(E_1) + p^{\Box}(E_2) \xrightarrow{a} q'$, which by the operational rule for $+$ means $p^{\Box}(E_1) \xrightarrow{a} q'$ or $p^{\Box}(E_2) \xrightarrow{a} q'$. The induction hypothesis now yields $E_1 \xrightarrow{a}_{\Box} E'$ or $E_2 \xrightarrow{a}_{\Box} E'$ with $q' = p^{\Box}(E)$, giving the desired.

$E = \mathrm{rec}y.E_1$**:** We here use Lemmae 3.1 and 3.2 from Chapter 3. By the rec-rule we have that $\mu y.p^{\Box}(E) \xrightarrow{a} q'$ iff $p^{\Box}(E)[\mu y.p^{\Box}(E)/y] \xrightarrow{a} q$. By Lemma 3.2 this is equivalent to $p^{\Box}(E) \xrightarrow{a} q'$ where $q = q'[\mu y.p^{\Box}(E)/y]$. Induction hypothesis gives us $E \xrightarrow{a}_{\Box} E' \wedge q' = p^{\Box}(E')$, which by Lemma 3.1 is equivalent to $E[\mathrm{rec}y.E/y] \xrightarrow{a}_{\Box} E'[\mathrm{rec}y.e/y]$ with $q = p^{\Box}(E)[\mu x.p^{\Box}(E)/x]$. By Lemma 3.2 we then get $E[\mathrm{rec}y.E/y] \xrightarrow{a}_{\Box} E'[\mathrm{rec}y.E/y]$ while we for q have $q = p^{\Box}(E'[\mathrm{rec}y.E/y])$. We can then use the rec-rule, and this ends the proof.

$\Box$

A.4 Proofs from Chapter 5

We prove the expansion theorem for parallel composition, based on the general expansion theorem:

Theorem A.2 *(Theorem 5.2)*
For any n-ary static construct $(\ldots)[f]$ we have

$$(F_1, \ldots, F_n)[f] =_D \sum_{\pi(a_m,(F_1,\ldots,F_n))} a_m.(F'_1, \ldots, F'_n)$$

where $\pi(a_m, (F_1, \ldots, F_n))$ is the set of possible derivations given by

$$\{(a_m,(F'_1,\ldots,F'_n)) \mid \exists(a_1,\ldots,a_m)\colon ((F_i \xrightarrow{a_i}_m F'_i) \vee (a_i{=}0 \wedge F_i{=}F'_i)),1{\leq}i{\leq}n \wedge f(a_1,\ldots,a_n)\}$$

and $=_D$ is the direct equivalence of [Milner 80]

Proof: Immediate from the operational rule. $\Box$

Theorem A.3 *(Theorem 5.3) For all $F_1, \ldots, F_n$ we have*

$$\begin{aligned}(F_1 \mid \ldots \mid F_n) \;\simeq\; & \sum_{\{(a_m,F'_i) \mid \exists i:1\leq i\leq n\wedge F_i \xrightarrow{a}_m F'_i\}} a_m.(F_1 \mid \ldots F'_i \ldots \mid F_n) \\ + & \sum_{\{(F'_i,F'_j) \mid \exists i,j:1\leq i,j\leq n\wedge F_i \xrightarrow{a}_m F'_i\wedge F_j \xrightarrow{\overline{a}}_m F'_j\}} \tau_m.(F_1 \mid \ldots F'_i \mid \ldots F'_j \mid F_n)\end{aligned}$$

Proof: Induction in n.

$n = 2$ This is just Theorem A.2.

Assuming for $n = k$: We have

$$\begin{array}{rcl}(S_1 \mid S_2 \ldots \mid S_k \mid S_{k+1}) & \simeq & (S_1 \mid S_2 \ldots \mid S_k) \mid S_{k+1} \\ & \simeq & (\sum_{\{(a_m,S_i') \mid S_i \xrightarrow{a}_m S_i', 1\le i\le k\}} a_m.(S_1 \mid \ldots S_i' \mid \ldots S_k) \\ & + & \sum_{\{(S_i',S_j') \mid S_i \xrightarrow{a}_m S_i' \wedge S_j \xrightarrow{\overline{a}}_m S_j', 1\le i,j\le k\}} \tau_m.(S_1 \mid \ldots S_i' \mid \ldots S_j' \mid S_k)) \mid S_{k+1}\end{array}$$

By Theorem A.2 we then get

$$\begin{array}{rcl}(S_1 \mid S_2 \ldots \mid S_k \mid S_{k+1}) & \simeq & \sum_{\{(a_m,S_i') \mid S_i \xrightarrow{a}_m S_i', 1\le i\le k\}} (a_m.(S_1 \mid \ldots S_i' \mid \ldots S_k)) \mid S_{k+1} \\ & + & \sum_{\{(S_i',S_j') \mid S_i \xrightarrow{a}_m S_i' \wedge S_j \xrightarrow{\overline{a}}_m S_j', 1\le i,j\le k\}} (\tau_m.(S_1 \mid \ldots S_i' \mid \ldots S_j' \mid S_k)) \mid S_{k+1} \\ & \simeq & \sum_{\{(a_m,S_i') \mid S_i \xrightarrow{a}_m S_i', 1\le i\le k+1\}} a_m.(S_1 \mid \ldots S_i' \mid \ldots S_k \mid S_{k+1}) \\ & & + \\ & & \sum_{\{(S_i',S_j') \mid S_i \xrightarrow{a}_m S_i' \wedge S_j \xrightarrow{\overline{a}}_m S_j', 1\le i,j\le k+1\}} \tau_m.(S_1 \mid \ldots S_i' \mid \ldots S_j' \mid S_k \mid S_{k+1})\end{array}$$

which completes the proof. □

A.5 Proofs from Chapter 6

Proposition A.1 *(Proposition 6.3)*

$$(F_1 \vee F_2) + F_3 \equiv (F_1 + F_3) \vee (F_2 + F_3)$$

Proof: We have

$$\begin{array}{rcl}[\![(F_1 \vee F_2) + F_3]\!] & = & \{p \mid \exists p_1 \in [\![F_1 \vee F_2]\!], \exists p_2 \in [\![F_3]\!] : p \sim p_1 + p_2\} \\ & = & \{p \mid \exists p_1 \in [\![F_1]\!], \exists p_2 \in [\![F_3]\!] : p \sim p_1 + p_2\} \\ & & \cup\{p \mid \exists p_1 \in [\![F_2]\!], \exists p_2 \in [\![F_3]\!] : p \sim p_1 + p_2\} \\ & = & [\![F_1 + F_3]\!] \cup [\![F_2 + F_3]\!]\end{array}$$

establishing the proof. □

Proposition A.2 *(Proposition 6.4)*

$$a_\diamond.F \equiv a_\Box.F \vee Nil$$

Proof:

$$[\![a_\diamond.F]\!] = \{p \mid p \xrightarrow{b} p' \Rightarrow (b = a \ \wedge \ p' \in [\![F]\!])\}$$

$$[\![a_\Box.F \vee Nil]\!] = \{p \mid p \sim nil\} \cup \{p \mid p \xrightarrow{a} \ \wedge \ (p \xrightarrow{a} p' \Rightarrow b = a \wedge p' \in [\![F]\!])\}$$

are seen to coincide, since $p \overset{b}{\not\to}$ means $p \sim nil$. □

Proposition A.3 *(Proposition 6.5)*

$$a_\Box.F_1 \vee F_2 \equiv a_\Box.F_1 \vee a_\Box.F_2 \vee (a_\Box.F_1 + a_\Box.F_2)$$

Proof: We here have

$$\begin{array}{rcl}
[\![a_\Box.(F_1 \vee F_2)]\!] & = & \{p \mid p \xrightarrow{a} \ \wedge \ (p \xrightarrow{b} p' \Rightarrow (b = a \wedge p' \in [\![F_1 \vee F_2]\!]))\} \\
[\![a_\Box.F_i]\!] & = & \{p \mid p \xrightarrow{a} \ \wedge \ (p \xrightarrow{b} \ \Rightarrow (b = a \wedge p' \in [\![F_i]\!]))\}, \ 1 \leq i \leq 2 \\
[\![a_\Box.F_1 + a_\Box.F_2]\!] & = & \{p \mid \exists p_i \in [\![F_i]\!], 1 \leq i \leq 2 : (p_i \xrightarrow{a} \ \wedge \ (p \xrightarrow{b} p' \Rightarrow \\
& & \qquad (b = a \ \wedge \ p' \in [\![F_i]\!]))) \ \wedge \ p \sim p_1 + p_2\}
\end{array}$$

□

In the following we refer to the rules and axioms of the proof system in Table 6.3.

In the proof of Theorem A.4 we need these easily established lemmae concerning the provability of semantic equality in $Sp_\vee$. The lemmae generalize proof rules D1-D3. Since the proofs all consist in induction in the size of the disjunction using D1-D3 successively along with P1-P4 and C1-C3, the proofs are omitted.

Lemma A.6

$$\vdash (\bigvee_{i=1}^{n} G_i) + H \equiv \bigvee_{i=1}^{n}(G_i + H)$$

Lemma A.7

$$\vdash a_\Box. \bigvee_{i=1}^{n} G_i \equiv \bigvee_{I \in \mathcal{K}} (\sum_{j \in I} a_\Box.G_j)$$

where $\mathcal{K} = \{I \mid I \subseteq \{1, \ldots, k\}, I \neq \emptyset\}$.

Lemma A.8

$$\vdash a_\diamond. \bigvee_{i=1}^{n} G_i \equiv \bigvee_{I \in \mathcal{K}} (\sum_{j \in I} a_\Box.G_j)$$

where $\mathcal{K} = \{I \mid I \subseteq \{1, \ldots, k\}\}$

Lemma A.9

$$\vdash \bigvee_{i=1}^{n_1} G_i + \bigvee_{j=1}^{n_2} H_j \equiv \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} (G_i + H_j)$$

Theorem A.4 *(Theorem 6.3) For any $G \in Sp_{\vee}$, there exist $F_1, \ldots, F_n \in Sp_{\Box}$ such that*

$$\vdash G \equiv \bigvee_{i=1}^{n} F_i$$

Proof: A structural induction on G.

$G = Nil$**:** $n = 1$, $F_1 = Nil$.

$G = a_{\Box}.G_1$**:** By induction hypotheses we have $\vdash G_1 \equiv \bigvee_{i=1}^{k} F'_i$ with $F'_i \in Sp_{\Box}$. By Lemma A.7 and we have by use of P2, P4, and C2 that

$$\vdash a_{\Box}.\bigvee_{i=1}^{k} F'_i \equiv \bigvee_{I \in \mathcal{K}} (\sum_{j \in I} a_{\Box}.F'_j)$$

where $I \subseteq \{1, \ldots, k\}$ with $I \neq \emptyset$. This is of the desired form.

$G = a_{\Diamond}.G_1$**:** As in the previous case, just use Lemma A.8 instead.

$G = G_1 \vee G_2$**:** Assume

$$\vdash G_1 \equiv \bigvee_{i=1}^{n_1} F_{1i} \text{ and } \vdash G_2 \equiv \bigvee_{j=1}^{n_2} F_{2j}$$

Then O5 used twice and P3 yield

$$\vdash G \equiv \bigvee_{i=1}^{n_1+n_2} F_i$$

which is an expression of the desired form.

$G = G_1 + G_2$**:** Assume as before

$$\vdash G_1 \equiv \bigvee_{i=1}^{n_1} F_{1i} \text{ and } \vdash G_2 \equiv \bigvee_{j=1}^{n_2} F_{2j}$$

Then, by C3 used twice, P3, and Lemma A.9 we get the desired result.

This ends the proof. $\square$

Proposition A.4 *(Proposition 6.6)*

$$\bigvee_{i=1}^{m} F_i \equiv \bigvee_{j=1}^{n} G_j$$

if and only if

$$\forall j \exists i : F_i \equiv G_j, \forall i \exists j : G_j \equiv F_i$$

where $\forall i, j : F_i, G_j \in Sp_{\square}$.

PROOF:

If: Obvious.

Only if: By Proposition 6.2 we have for all i that $[\![F_i]\!] = \{p \mid p \sim p^{\square}(F_i)\}$ and likewise for G_j. Since equivalence classes are either identical or disjoint, this must mean that for all F_i we have a G_j such that $F_i \equiv G_j$.

$\square$

Proposition A.5 *(Proposition 6.7) If* $\forall i \leq m \exists j \leq n : \vdash F_i \equiv G_j$ *and* $\forall j \leq n \exists i \leq m : \vdash F_i \equiv G_j$ *we have*

$$\vdash \bigvee_{i=1}^{m} F_i \equiv \bigvee_{j=1}^{n} G_j$$

PROOF: By P3 we get that

$$\forall i \leq m \exists j \leq n : \vdash F_i \subseteq G_j \text{ and } \forall j \leq n \exists i \leq m : \vdash G_j \subseteq F_i$$

implying by O4 that

$$\forall i \leq m : \vdash F_i \subseteq \bigvee_{j}^{n} G_j \text{ and } \forall j \leq n : \vdash G_j \subseteq \bigvee_{i}^{m} F_i$$

which by repeated use of O5 and O1 yields

$$\vdash \bigvee_{i=1}^{m} F_i \subseteq \bigvee_{j=1}^{n} G_j \text{ and } \bigvee_{j=1}^{n} G_j \subseteq \bigvee_{i=1}^{m} F_i$$

which by P3 gives the desired result. $\square$

Bibliography

[Abramsky 88] Abramsky, Samson: *A Domain Equation for Bisimulation*, Draft, Imperial College, London 1987.

[Bækgaard et al 86] Bækgaard, Lars, Hans Hüttel, Jens Chr. Godskesen, Christian S. Jensen, Per K. Olesen, Jan Sieker and Carsten Sørensen: *UPS - Exception Handling in Programming Languages* (in Danish), Student Report, Aalborg University Centre 1986.

[Bækgaard et al 87] Bækgaard, Lars, Hans Hüttel, Christian S. Jensen, Jan Sieker and Carsten Sørensen: *DEPRI - Design Principles for Programming Tools* (in Danish), Student Report, Aalborg University Centre 1987.

[Garey and Johnson 1979] Garey, Michael R. and David S. Johnson: *Computers and Intractability - A guide to the Theory of NP-completeness*, W.H. Freeman and Co., San Francisco 1979.

[Godskesen et al 87] Godskesen, Jens Christian, Anna Ingolfsdottir and Michael Zeeberg: *From Hennessy-Milner logic to $\mathcal{CCS}$ -processes* (in Danish), Student Report, Aalborg University Centre 1988.

[Graf and Sifakis 86] Graf, S. and J. Sifakis: *A Logic for the Description of Non-deterministic Programs and Their Properties*, Information and Control, Vol. 68, Nos. 1-3, January/February/March 1986.

[Hennessy and Milner 85] Hennessy, Matthew and Robin Milner: *Algebraic Laws for Nondeterminism and Concurrency*, Jour-

	nal of The ACM, Vol. 32, No.1, January 1985 (pp. 137 - 161).
[Holmström 87]	Holmström, Sören: *Reasoning About $\mathcal{CCS}$ agents Using Hennessy-Milner Logic Extended With Fixed Points*, unpublished paper, 1987.
[Hughes and Creswell 84]	Hughes, G.E. and M.J. Creswell: *A Companion to Modal Logic*, Methuen and Co., London 1984.
[Larsen 86]	Larsen, Kim G. : *A Context-Dependent Bisimulation Between Processes*, Ph.D. Thesis, University of Edinburgh, 1986.
[Larsen 87]	Larsen, Kim G.: *Proof Systems for Hennessy-Milner Logic with Recursion*, in: *CAAP 88*, Springer Lecture Notes in Computer Science 299, 1988. (Extended version to appear in Theoretical Computer Science, North-Holland).
[Larsen and Thomsen 87]	Larsen, Kim G. and Bent Thomsen: *Compositional Proofs by Partial Specification of Processes*, Internal Report R 87-20, Aalborg University Centre, 1987. To appear in: *Proceedings of MFCS*, Carlsbad, Czekoslovakia.
[Larsen and Thomsen 88]	Larsen, Kim. G and Bent Thomsen: *A Modal Process Logic*. Internal Report R 88-5, Aalborg University Centre, 1988. To appear in: *Proceedings of LICS 88*, Edinburgh, Scotland.
[Manes and Arbib 75]	Manes, Ernest G. and Michael A. Arbib: *Arrows, Structures, and Functors*, Academic Press, New York 1975.
[Milner 80]	Milner, Robin: *A Calculus of Communicating Systems*, Springer Lecture Notes in Computer Science 92, Springer-Verlag 1980.
[Milner 82]	Milner, Robin: *A Complete Inference System for a Class of Regular Behaviours*, Report CSR-11-82, Department of Computer Science, University of Edinburgh, April 1982.

[Milner 83] Milner, Robin: *Calculi for Synchrony and Asynchrony*, Theoretical Computer Science 25 (pp. 267-310), North-Holland 1983.

[Parrow 85] Parrow, Joachim: *Fairness Properties in Process Algebra*, Ph.D. Thesis, Dept. of Computer Systems, Uppsala University, Uppsala 1985.

[Plotkin 81] Plotkin, Gordon: *A Structural Approach to Operational Semantics*, Tech. Rep., DAIMI FN-19, Computer Science Department, Aarhus University 1981.

[Pnueli 77] Pnueli, Amir: *The Temporal Logic of Programs.* In: Proceedings of the 18th Symposium on Foundations of Computer Science (Providence, R.I., November), 1977 (pp. 46-57).

[Prasad 87] Prasad, K.V.S: *Combinators and Bisimulation Proofs for Restartable Systems*, Ph.D. Thesis, Dept. of Computer Science, University of Edinburgh 1987.

[Stoy 77] Stoy, Joseph E.: *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press Series in Computer Science, 1977.

[Thomsen 87] Thomsen, Bent: *An Extended Bisimulation Induced by a Preorder on Actions*, Master's Thesis, Aalborg University Centre 1987.