Verification of Temporal Properties of Concurrent Systems

Henrik Reif Andersen

Ph.D. thesis

Department of Computer Science
Aarhus University
Denmark

June 1993

Dansk Sammenfatning

Denne afhandling behandler metoder og algoritmer til *verifikation af parallelle systemer*. Verifikationen af et system foretages ud fra en givet specifikation, der udtaler sig om visse ønskelige aspekter, og en, muligvis ufuldstændig, beskrivelse af det parallelle system.

Specifikationer vil blive angivet i en kraftig modallogik - den modale μ-kalkule - og parallelle systemer beskrives som processer fra en procesalgebra, der følger traditionen fra Milner's CCS og Hoare's CSP. Afhandlingen tager udgangspunkt i en kompositionel metode, der, givet en specifikation til en sammensat proces, dekomponerer denne til speficikationer om delprocesser, således at den oprindelige proces vi1 tilfredsstille sin specifikation, hvis og kun hvis, delprocesserne tilfredsstiller deres delspecifikationer. En sådan kompositionel metode bidrager til at kunne håndtere den store kompleksitet som parallelle systemer ofte besidder.

Selvom den modale μ-kalkule er meget udtryksfuld, har den fra et pragmatisk synspunkt nogle mangler, som vi vi1 forsøge at udbedre ved introduktionen af en *udvidet μ-kalkule*, der, udover nogle yderligere logiske konstruktioner, som tillader nemmere formulering af egenskaber i logikken, også har en mulighed for kompakt repræsentation af specifikationer ved deling af deludtryk gennem simultane fikspunkter. Fra en mindre observation i den kompositionelle metode udnyttes denne mulighed for kompakte repræsentationer til at give effektive globale og lokale algoritmer til automatisk afgørelse af om en endelig proces tilfredsstiller sin specifikation – et problem der benævnes *model-check*. Den modale μ-kalkule får sin udtrykskraft fra tilstedeværelsen af minimale og maksimale fikspunkter; effektiv beregning af fikspunkter indgår således som en vigtig del af algoritmerne til modelcheck.

De centrale ideer med deling af værdier og opfølgning af ændringer brugt i disse algoritmer er af en generel natur – en observation, der bliver brugt til at give en algoritme for beregning af fikspunkter i endelige fuldstændige partielle ordninger og gitre.

For uendelige tilstandssystemer er modelcheckproblemet generelt uafgørligt, så vi er nødt til betragte semi-automatiske eller brugerassisterede systemer. Vi præsenterer en metode baseret på angivelse af passende vel-funderede ordninger for de minimale fikspunkter. Metoden er en slags målorienteret bevissystem: Startende med målet, en proces og en specifikation som processen skal vises at tilfredsstille, konstrueres nye delmål ud fra et sæt af regler. Dette

kan gentages indtil alle delmål er trivielle. Metoden bevises at være sund og fuldstændig.

Endelig præsesenteres en ny måde at angribe det åbne problem for μ-kalkulen, der består i at finde en endelig aksiomatisering. Vi karakteriserer en klasse af kategoriske modeller for en intuitionistisk version af kalkulen og reformulerer problemet som et problem om hvordan disse kategoriske modeller kan skæres ned til mere traditionelle Kripke-agtige modeller.

I det konkluderende kapitel diskuteres kort kompleksiteten af automatisk modelcheck og det vises at for selv en simpel klasse af processer er problemet hårdt (‘PSPACE-hard’).

Abstract

This thesis is concerned with the verification of concurrent systems. It provides methods and techniques for reasoning about temporal properties as described by assertions from an expressive modal logic – the modal μ-calculus. It describes a compositional approach to verifying whether processes satisfy assertions from the logic where processes are drawn from a process language encompassing CCS, CSP and related process languages. This compositional approach is based on the notion of a reduction which transforms a satisfaction problem for a composite process into satisfaction problems for the subcomponents.

Although the modal μ-calculus is very expressive from a theoretical point of view, it leaves much to be desired in practical applications. Hence, we introduce an extended version of the modal μ-calculus which is more convenient for expressing properties. Among other things it allows for a compact representation of assertions by simultaneous fixed-points. As a side-effect it provides, using the compositional method, a means for constructing efficient local and global model checkers for automatically deciding satisfaction for finite-state processes. The central ideas of sharing values and tracing dependencies that are used in these algorithms are of a general nature; an observation which is exploited in giving a general fixed-point finding algorithm for finite cpo's and lattices.

For infinite-state systems a method based on supplying well-founded orders for the minimum fixed-points is presented. The method has the character of a goal-oriented proof system: Starting with the goal of interest new subgoals are produced by a set of rules. The method is proven sound and complete.

Finally, we begin a new attack on the fundamental problem of finding a finite axiomatization of the modal μ-calculus by giving categorical models of an intuitionistic version of the calculus.

In the concluding chapter we briefly discuss the complexity of model checking and prove the negative result that even for a simple class of finite concurrent processes the problem is intractable, in the sense that the problem is PSPACE-hard.

Acknowledgements

First of all I thank my supervisor Glynn Winskel who introduced me to the subject and provided many inspiring discussions. Many other people at DAIMI have influenced my work and I have had useful discussions with numerous people, in paxticular Uffe Engberg. In Cambridge I would like to thank Andrew Pitts for his efforts in teaching me about categorical logic; and thanks also to all the other regular guests at the coffee mornings: Christine Ernoult, Valeria de Paiva, Brian Graham, Monica Nesi, and others. At DIKU I wish to thank Neil Jones, Fritz Henglein, Lars Ole Andersen and everybody else in the TOPPS group for their kind hospitality.

Also thanks to all my office-mates: Søren Christensen, who also took good care of me during my two Edinburgh visits, Torben Amtoft, Mike Warner, Carsten Gomard, and Karoline Malmkjær.

Finally, thanks to Karen and Andrea for their support and for bearing with my occasional frustrations.

This is a slightly revised version of the thesis as submitted. It takes into account the changes suggested to me by the examiners, Mogens Nielsen, Glynn Winskel and Colin Stirling. Special thanks are due to Colin for his detailed comments and clarifying remarks on the history of the modal μ-calculus which helped me improve chapter 4.

I have been financially supported by the Danish Natural Science Research Council and by a travel grant from the Danish Research Academy.

Contents

Chapter 1

Introduction

In contrast to many other sciences, computer science has the advantage that many aspects of programming and design of systems have such a formal character that in principle it should be possible to prove – contrary to experimentally testing – that the design is correct. Hence, one of the major challenges of computer science today is to find methods and techniques for performing this formal reasoning. This thesis is a contribution in that field. More precisely, it describes methods aimed at the verification of *concurrent systems*.

1.1 Verification of Concurrent Systems

Concurrent systems have properties that are quite different from ordinary sequential programs. Whereas aspects as input-output behaviour and termination is important properties of sequential programs, the emphasis in concurrent systems is more on the *communication patterns* and *interactions* between otherwise independent components. Termination is not necessarily an essential feature; many concurrent systems are supposed to run indefinitely as in for instance embedded systems, and it is their behaviour and responses to various *actions* from their environment that is of interest. To emphasize this point such systems are often referred to as *reactive systems* (a term introduced by Pnueli).

We consider reactive systems as being described by terms in a process language, which we shall refer to as WPA, designed in the spirit of CCS and

CSP; in fact, CCS, CSP and other process algebras appear as sublanguages of WPA. Labelled transition systems capturing the idea that a system has state which changes by performing actions, are used as the basic model of concurrent systems and used in giving an operational semantics of WPA.

The general idea to verification will be that only *some aspects* of the implementation will be specified and reasoned about. Hence, we do not consider the idea of giving a *complete specification* capturing all aspects of interest and from this derive an implementation. What we present are various techniques for *partial verification*, i.e. we can verify as many aspects as we wish, but we do not require specifications to capture the implementation up to equivalence.[1]

Reactive systems often have a very complex structure which make them difficult to design and analyze. It is easy to make mistakes and it can be quite hard to ensure that the design is free for undesired properties, like deadlocks and in turn possess the desired properties. For sequential systems, Hoare logic for instance offers a structured, compositional way of verifying a program: Given a specification for the program to fulfill, it is possible to prove this fact in a structured, compositional way by proving certain derived facts about parts of the program. Hence, whereas compositional methods are convenient tools for sequential programs they seem to be *essential* for providing structured ways of attacking the complexity of reactive systems. However, no such successful method has yet been found. Recent years have shown a growing interest in this problem and various approaches have emerged; many with the idea of supplying heuristics, i.e. criterions that work in some – not always well-characterized – situations, and fail in others. Instead of contributing with yet another heuristic, we supply a general method based on the notion of *reductions* that work for a well-defined subset of our process language, and which besides being used in formal reasoning about concurrent systems, will provide algorithms for constructing characteristic formulae for behavioural relations and even efficient model-checking algorithms.

We present two specification languages: The modal μ-calculus as introduced by Kozen (building on earlier work by Pratt) which we shall refer to as the *standard calculus* and an enriched *extended calculus.* The standard calculus has the big advantage of being extremely simple in terms of the number of

[1]Although the specification language will turn out to be strong enough to express equivalences and preorders and hence allow complete verification, this is merely a benefit of the generality of the specification language, not a motivating goal.

logical connectives; thereby making it well-suited to theoretical investigations into issues as logical expressiveness, decidability, axiomatizations and so on. We will use the standard calculus as the core of the compositional method, the algorithms, and the proof system for infinite-state systems for precisely these reasons of economy. The central points of all the techniques will be illustrated from the basic constructions of the standard calculus, without much emphasize on the – from a modal logic point of view – rather trivial extensions present in the extended calculus.

However, when we turn to the practical applications of the results to even small examples the standard calculus has some shortcomings, especially as concerns the treatment of *actions*, the basic computational steps of our systems, which are simply viewed as 'simple-minded constants' which has no other properties than their different identities. To remedy this, from a pragmatic point of view, unfortunate situation we extend the standard calculus with a *first-order predicate logic* on actions and allow *simultaneous fixed-points*. The extension of the compositional method to this full, richer logic will be given without the same level of details in the proofs as for the standard calculus; this sloppiness being justified by arguments showing why such extensions to a large degree are rather immediate.

Moreover, for properly well-behaved sub-logics of the extended calculus the model checking algorithms from the standard calculus will be adapted, yielding algorithms for automatically verifying a very large class of properties of concurrent systems including equivalences and even rather exotic preorders with quite a reasonable level of efficiency.

The extended calculus is getting very close to what one reasonably could call a *realistic specification language*. An analogy with the functional programming community viewing the standard calculus as the lambda calculus and the extended calculus as a full functional programming language with built-in operators and basic constants is tempting; the lambda calculus has all the expressive power one needs still being an extremely small language making it well-suited for theoretical investigations, whereas for practical programming it leaves much to be desired. Similarly, the standard calculus is suitable for theoretical considerations, but for practical purposes the extended calculus offers a more convenient language.

The modal μ-calculus gets its expressiveness from the presence of minimum and maximum fixed-points. Besides the important implications for expressiveness the combination of modal operators and fixed-points poses a lot of interesting theoretical questions, which have received much attention during the last decade. The logic has been shown to be decidable and it has the finite model property, but it is still open whether a finite axiomatization exists and whether the hierarchy one gets from the nesting of minimum and maximum fixed-points is strict.

Even more effort has been put into the more pragmatic aspects of deciding satisfiability – called model checking – due to its immediate application as giving a means for determining that processes satisfy their specifications. This thesis is mostly concerned with the pragmatic aspects relating to the model checking problem, but we also take a little detour into the more theoretical problems.

In fact, the distinction between theoretical and pragmatic aspects is somewhat misleading. Actually, all the classical questions asked for a logic has an immediate application to program verification: Decidability of the logic corresponds to the ability to determine implications between specifications and the ability to detect trivial (always true) and inconsistent (always false) specifications. Deciding satisfiability corresponds to verifying that processes meet their specifications. The finite model property means that satisfiable assertions always have a finite implementation. An axiomatization gives a calculus for reasoning about specifications, and so on.

1.2 Organization of the Thesis

Chapter 2 introduces the process language and the logics. It provides background material for the rest of the thesis.

The first method we consider is a *compositional method* in chapter 3. It describes how properties of a compound process can be showed valid by considering derived properties of subprocesses. The method is applied on a couple of examples and partially generalized to the extended calculus.

Chapter 4 provides a useful collection of derived assertions, called 'macros', and shows how the extended calculus can be used for expressing preorders and equivalences, facilitating through the compositional method the immediate generation of characteristic formulae.

Based on an idea from the compositional method we will describe various algorithms in chapter 5 for automatically determining whether a process with a finite number of states satisfy a specification; some of these algorithms will turn out to be more efficient than previous algorithms. Central to these algorithms are efficient ways of computing fixed-points, and one of the algorithms is in chapter 6 generalized to solve fixed-point problems in arbitrary finite cpo's and lattices.

This will be followed up by a technique for reasoning about infinite-state systems in chapter 7. In general model checking is undecidable for infinite-state systems, so we have to resort to semi-automatic methods. We provide a complete proof system presented as set of rewrite rules.

In chapter 8 we approach the open problem of finding a finite axiomatization for the modal μ-calculus by supplying categorical models for an intuitionistic version of the logic. These models offer another way of attacking the problem: They can easily be shown to be complete for the given axiomatization. Hence, finding a proper way of extracting Kripke-like models from the categorical models would then solve the original problem.

Finally, in chapter 9 we draw some conclusions and point to future work. We discuss briefly the complexity of model checking and show that it is provably intractable in general.

Chapter 3 is joint work with Glynn Winskel, chapter 8 is joint work with Andrew Pitts, and the complexity analysis in chapter 9 have been inspired by discussions with Neil Jones. Various parts of the thesis have appeared elsewhere: An earlier version of chapter 3 appeared as Andersen and Winskel [7] (and an extended abstract as Andersen and Winskel [8]). Chapter 5 is a major revision of [6] and [4]. Chapter 6 is based on [5].

Chapter 2

Logic and Models

In this chapter we introduce the modal μ-calculus as the language of specifications and labelled transition systems as the underlying models of concurrent systems. We describe a language of processes, the process algebra WPA (for Winskel's Process Algebra, see Winskel [91]), which through an operational semantics gives rise to the models of the logic: labelled transition systems. The process language WPA has operators for constructing and combining processes familiar from the work on CCS and CSP and the parallel compositions from both of these languages will appear as derived forms of a more general product construction.

The modal μ-calculus will be extended in several ways for pragmatic and technical reasons allowing for easier and more concise formulations of properties. One such important extension is the ability to express *simultaneous fixed-points* allowing for sharing of subassertions and thereby more compact representations of assertions, a feature that will play a crucial role for the efficiency of the model-checking algorithms.

2.1 Labelled Transition Systems

Labelled transition systems are formally defined as follows.

Definition 2.1 A *labelled transition system* T is a triple $(S, L, \rightarrow)$, where

S is a set of *states*,
L is a set of *labels*, and

$$\rightarrow \subseteq S \times L \times S \text{ is a } \textit{transition relation}.$$

We normally write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$, and $\xrightarrow{a}$ for the relation $\{(s, s') \mid s \xrightarrow{a} s'\}$.

The set R_p of *reachable states from* p of a labelled transition system $T = (S, L, \rightarrow)$ is the least subset of S containing p and closed under $\rightarrow$. Whenever confusion might arise we use superscript T as in R_p^T to indicate the transition system under consideration. We will write $s \xrightarrow{a}$ as an abbreviation for the predicate $\exists s'.\ s \xrightarrow{a} s'$ and $s \not\xrightarrow{a}$ as an abbreviation for $\neg(s \xrightarrow{a})$.

A *pointed transition system* T is a quadruple $(S, i, L, \rightarrow)$ where $(S, L, \rightarrow)$ is a labelled transition system, $i \in S$ is a distinguished *initial state*, and all states of S are reachable from i, i.e. $R_i = S$.

The *size* of a finite transition system $T = (S, L, \rightarrow)$ is defined by $|T| = |S| + |\rightarrow|$ where $|S|$ is the number of states in S and $|\rightarrow|$ is the number of transitions in $\rightarrow$. $\square$

Sometimes we will be a bit sloppy in our use of the notions and refer to pointed transition systems as labelled transition systems, merely using the term "pointed" when needed for emphasizing the presence of an initial state. If $s \xrightarrow{a} s'$ we refer to s' as a *(direct) a-successor of* s and to s as a *(direct) a-predecessor of* s'.

When using labelled transition systems in the description of concurrent systems the labels are interpreted as *actions* which can take place in the system, and the system is considered as being in one particular state at any given time, changing states by performing actions in accordance with the transition relation. Notice, that the transition relation can be completely arbitrary, hence in general it will be non-deterministic (states can have more than one a-successor for some label a), cyclic, and partial (not every state has an a-successor for all labels a).

The non-deterministic features are crucial since in concurrent systems parallelism is in a certain – precise – sense reduced to non-deterministic interleaving of actions, as we shall see in the next section.

2.2 A Process Algebra

The language of processes we are just about to introduce is in spirit very close to Milner's CCS [59, 58] and Hoare's CSP [42], but the operators will be slightly refined such that parallel composition as known from these languages will be derived from a *product*, a *restriction*, and a *relabelling operation* much like Winskel's synchronization algebras [88] – a general scheme for defining parallel compositions. The reason for refining the operators is purely technical[1]; it makes some of the results about the compositional method easier to state and prove and it makes the results very general as many of the parallel operators considered in the process algebraic literature will be covered (actually, all that can be described by a synchronization algebra). Moreover, the product will turn out to be very useful in chapter 4 for stating some, perhaps surprising applications of the compositional method and the model-checking algorithms.

Assume we have given a set of *basic actions Act* which play the role of being the primitive (atomic) actions that a process can perform. Moreover, we assume that we have given a set of *state identifiers* (or state names) *Nam*. The process terms t of WPA are constructed from the following grammar:

$$t ::= nil \mid a.t \mid t_0 + t_1 \mid t_0 \times t_1 \mid t \restriction \Lambda \mid t\{\Xi\} \mid rec\ P.t \mid P \qquad \text{(WPA)}$$

where P ranges over *Nam*.

Nil is the *inactive process*, and $a.t$ is the *prefix operator*, where a is a basic action. We often leave out *nil* and write e.g. $a.nil + b.nil$ as $a + b$. The term $t_0 + t_1$ is the *non-deterministic choice operation* (also called *sum*) known from CCS. The product term $t_0 \times t_1$ denotes a very general kind of parallel composition which allows the components t_0 and t_1 to proceed both synchronously and asynchronously. It is neither commutative nor associative. The precise semantics will be defined below.

A state identifier P in the body of $rec\ P.t$ works as a *recursion point*, and in effect will behave as the normal recursion in CCS: A term $rec\ P.t$ has the same behaviour as the *unfolded* term $t[rec\ P.t/P]$, where by $t[rec\ P.t/P]$ we denote the result of substituting $rec\ P.t$ for all free occurrences of P in t - we are applying the usual notion of free and bound occurrences to state identifiers, so that P will be bound in $rec\ P.t$ but free in $P + nil$.[2]

From the basic actions *Act* we define a set of *composite actions* Act_* as follows. Let $*$ be a distinguished symbol not contained in *Act*. The symbol $*$

[1]In fact the operators have a categorical justification, see e.g. Winskel [91].

[2]Substitution is defined by renaming bound variables – α-conversion in lambda-calculus terms – such that unintended binding of free variables of t is avoided.

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

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \; a \neq * \qquad\qquad \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'} \; a \neq *$$

$$\frac{p \xrightarrow{a} p' \quad q \xrightarrow{b} q'}{p \times q \xrightarrow{a \times b} p' \times q'} \qquad\qquad \frac{t[\mathit{rec}\ P.t/P] \xrightarrow{a} t'}{\mathit{rec}\ P.t \xrightarrow{a} t'} \; a \neq *$$

$$\frac{p \xrightarrow{a} p'}{p\{\Xi\} \xrightarrow{b} p'\{\Xi\}} \; \Xi(a) = b \qquad\qquad \frac{p \xrightarrow{a} p'}{p \upharpoonright \Lambda \xrightarrow{a} p' \upharpoonright \Lambda} \; a \in \Lambda$$

Figure 2.1: Operational rules.

is called the *idling action* and interpreted as 'no action'. Define Act_* to be the least set including $Act \cup \{*\}$ and such that $\alpha, \beta \in Act_*$ implies $\alpha \times \beta \in Act_*$ taking $* \times * = *$. Processes constructed from the product operator will perform such composite actions.

In the *restriction* $t \upharpoonright \Lambda$, Λ is a subset of Act_* restricting the actions of t to those in Λ. In the *relabelling* $t\{\Xi\}, \Xi : Act_* \rightharpoonup Act_*$ is a partial function, such that $*$ is not in the domain of Ξ. A relabelling map is extended to a total function on Act_* by taking it to be the identity outside the domain of definition, and when referring to Ξ in for instance operational rules we are always referring to this total extension of Ξ.

The semantics of processes will be given in an operational fashion as a labelled transition system $\mathcal{T} = (Proc_{WPA}, Act_*, \longrightarrow)$ where $Proc_{WPA}$ is the set of *process terms* of WPA and $\longrightarrow$ is defined inductively as the least relation satisfying the rules of figure 2.1. There are no rules for *nil* and the state identifiers since they cannot perform any actions.

Note in particular the rule for product. One of the components in the product may *idle* by means of the idling action $*$ allowing the other component to proceed independently, as in the transition

$$p \times q \xrightarrow{a \times *} p' \times q$$

where the left component p performs an a-action and the right component idles.

The operational rules for the operators of restriction, relabelling, and product all have the same property of keeping the operators in the term after an action takes place and we use the CCS terminology of calling these *static operators*. Contrary to this the operators of prefix, sum, and recursion are removed (or for recursion also added) when an action takes place and they are called *dynamic operators*. We consider *nil* to be a dynamic operator (it is an 'empty sum') but we could just as well have classified it as being static (it is not removed by actions being performed).

The rather huge transition system $\mathcal{T}$ which we will refer to as the *universal transition system* describes the behaviour of *all* processes of our language: For a particular process p the operational behaviour is found by viewing p as a state of $\mathcal{T}$ and look at the transitions from p to other processes and so on. This suggests another way of associating a transition system to a process: Restrict attention to the states reachable from p. We will refer to this sub-system of $\mathcal{T}$, consisting of the states $R_p^{\mathcal{T}}$ and the relevant part of the transition relation, as the transition system *induced by p* and hence it is *pointed by p*. When we are going to verify properties of processes, it is this more local view of the semantics that is going to be important.

Remark 2.1 Care should be taken not to confuse the state identifiers P with process variables as used in e.g. CCS, where the states of the transition system giving the operational semantics is taken to be the *closed* process terms only and not all terms as done here for WPA . In many respects the state identifiers *do* work as process variables, but we prefer to view them as named states, since in the compositional method we will need to refer to these states by assertions $\widehat{P}$ true only at the particular state P, which seems more natural for state identifiers than for variables. □

To avoid extensive use of parentheses we assume that the operators bind with decreasing strength as follows: restriction, relabelling, prefix, product, sum, and recursion. I.e. restriction binds tightest and *rec* reaches 'as far to the right as possible.' E.g.

$$rec\ P.a.P + b.Q \restriction \Lambda \times P = rec\ P.((a.P) + ((b.(Q \restriction \Lambda)) \times P)).$$

For later reference we define:

Definition 2.2 A state identifier P is *guarded in t* if all occurrences of P are within the scope of a prefix, P is *strongly guarded in t* if P in all occurrences appears immediately under a prefix. A term t is (strongly) guarded if all state identifiers are (strongly) guarded in t. □

The process language just defined is very powerful computationally; in fact it has the full strength of a Turing machine. The easiest way to see this, is by using the embedding of CCS into the process language given in the next section, and refer to the proof of Turing strength of CCS outlined by Milner [59, sec. 6.1]. Milner's result is based on the ability to express *unboundedly evolving structures*, i.e. systems which can get arbitrarily large, storing unbounded amounts of information. Technically, this is achieved through applying recursion across parallel composition, exemplified by the process

$$p = rec\ P.b.nil \ ||| \ a.P$$

where

$$\begin{array}{rcl} q \ ||| \ r & = & (q \times r) \restriction \Lambda\{\Xi\} \\ \Lambda & = & \{a \times *, * \times a \mid a \in Act\} \\ \Xi(x) & = & \begin{cases} a & \text{if } x \equiv a \times * \text{ or } x \equiv * \times a \\ \text{undefined} & \text{otherwise} \end{cases} \end{array}$$

with the infinite transition system of figure 2.2. Notice, that we always leave out the idling actions from our diagrams. The process p mimics a simple stack; a being "push" and b being "pop."

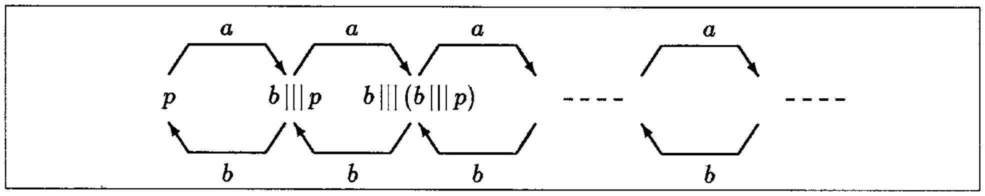


Figure 2.2: An infinite transition system.

Sometimes we will be concerned with restricting attention to processes with an associated finite labelled transition system. Determining precisely when a process gives rise to a finite state system is undecidable, so approximate criteria are needed.[3] Various syntactically or semantically based criteria

[3]Using the Turing power of the language it is not hard to code up some of the undecidable problems of Turing machines as questions of finiteness of transition systems induced by process terms. In fact we claim that the encoding can be chosen in such a way that termination of a Turing machine on the empty tape will directly correspond to finiteness of the transition system of the encoding. Alternatively, Taubner [83] could be consulted for a proof of the fact. See also the discussion in chapter 9.

could be put forward (see e.g. Taubner [83]), we, however, stick to the following simple syntactic criterion.

Definition 2.3 A process term t is said to be *finitary* if

(i) No subexpression $p \times q, p\{\Xi\}$, or $p \upharpoonright \Lambda$ of t contains a free state identifier.
(ii) All state identifiers of t are strongly guarded.

$\Box$

We can now state the following proposition:

Proposition 2.1 *Any finitary process term t induces a finite transition system.*

In order to prove the proposition we need a little lemma about guarded recursion:

Lemma 2.1 *If P is guarded in the process term q, then*

$$q[rec\ P.t/P] \xrightarrow{a} r \quad implies \quad \exists r'.q \xrightarrow{a} r' \ \&\ r'[rec\ P.t/P] \equiv r$$

Proof: Structural induction on q. $\Box$

Proof (Proposition 2.1): For all terms t define the set of terms D_t inductively as follows:

$$\begin{array}{rclrcl} D_{nil} & = & \{nil\} & D_{a.t} & = & \{a.t\} \cup D_t \\ D_{t_0+t_1} & = & \{t_0 + t_1\} \cup D_{t_0} \cup D_{t_1} & D_{t_0 \times t_1} & = & D_{t_0} \times D_{t_1} \\ D_{t \upharpoonright \Lambda} & = & D_t \upharpoonright \Lambda & D_{t\{\Xi\}} & = & D_t\{\Xi\} \\ D_{rec\ P.t} & = & \{rec\ P.t\} \cup D_t[rec\ P.t/P] & D_P & = & \{P\} \end{array}$$

where the operators of the process algebra when applied on the right-hand side denote their pointwise extensions to sets, and $D_t[rec\ P.t/P]$ is the result of substituting $rec\ P.t$ for P in all terms in D_t.

It is obvious that D_t is finite for any t. It can now be shown by structural induction on t that for any t satisfying (i) and (ii) of definition 2.3,

$$R_t \subseteq D_t \tag{2.1}$$

i.e. the set of states reachable from t is contained in D_t and hence finite.

The difficult case is the recursion operator, where the following observation is needed: Assume $t \equiv \mathit{rec}\ P.q$. If $P \notin R_q$ then it is not hard to see from the induction hypothesis that $R_{\mathit{rec}\ P.q} \subseteq R_q \cup \{\mathit{rec}\ P.q\} \subseteq D_q \cup \{\mathit{rec}\ P.q\} = D_{\mathit{rec}\ P.q}$. If $P \in R_q$ then we prove $\forall n \in \omega.Q(n)$ by mathematical induction on n, where

$$\begin{aligned} Q(n) \Leftrightarrow_{def} \forall a_1, \ldots, a_n.t \xrightarrow{a_1}\xrightarrow{a_2} \ldots \xrightarrow{a_n} t' \Rightarrow \\ \exists q'.t' \equiv q'[\mathit{rec}\ P.q/P]\ \&\ q' \in R_q. \end{aligned} \tag{2.2}$$

This means, in words, that any state reachable from t must be constructed by simply substituting $\mathit{rec}\ P.q$ for P in a state q' reachable from the subterm q. The motivation of conditions (i) and (ii) of definition 2.3 are precisely to make this true.

The base case, $Q(0)$ is immediate since $t \equiv \mathit{rec}\ P.q$; simply take $q' \equiv P$. For the inductive step, assume $Q(n)$ and suppose

$$t' \xrightarrow{a} t'',$$

where $t' \equiv q'[\mathit{rec}\ P.q/P]$ and $q' \in R_q$. If P is guarded in q' then $Q(n+1)$ follows by lemma 2.1. If P is unguarded in q' then we will argue that $q' \equiv P$, from which we conclude the following steps:

$$\begin{aligned} & t' \equiv P[\mathit{rec}\ P.q/P] \equiv \mathit{rec}\ P.q\ \&\ t' \xrightarrow{a} t'' \\ \Rightarrow\quad & q[\mathit{rec}\ P.q/P] \xrightarrow{a} t'' \\ & \qquad\qquad \text{as the only rule for } \mathit{rec} \text{ is the unfolding rule} \\ \Rightarrow\quad & \exists t'''.q \xrightarrow{a} t'''\ \&\ t'''[\mathit{rec}\ P.q/P] \equiv t'' \\ & \qquad\qquad \text{by lemma 2.1 as } P \text{ is guarded in } q \end{aligned}$$

Now, why is $q' \equiv P$ if P is unguarded in q'? We will not give a formal proof but argue intuitively. As we assume that P is strongly guarded in q, P cannot appear as $P + r$ or $\mathit{rec}\ Q.P$ in q', in fact the only way P could be unguided in a successor to q without being P is if P appeared inside one of the static operators $(\times, \lceil, \{\})$ and it should then have appeared inside this operator in q (or have entered there by some recursion) which is excluded by condition (i) of definition 2.3. $\Box$

A very simple example which does not satisfy the criteria of the proposition is

$$P =_{\text{def}} rec\ P.a.P\{\Xi\},$$

where $\Xi(a) = b$. The process p induces an infinite transition system, although it should be obvious that p is *semantically* identical to the transition system pointed by q (figure 2.3). This example would benefit greatly from using some simple equivalences between states to generate an equivalent finite-state representation (by allowing composition of relabellings), but as already noted we will not get involved with this ingenious task of finding finite representations of *essentially finite* but *syntactically infinite* transition systems. Taubner [83] and Francesco and Inverardi [40] should be consulted for a discussion of this.

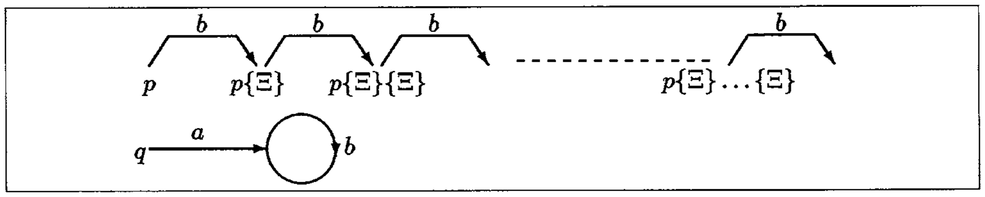


Figure 2.3: An infinite transition system with a simple finite representation.

Presently the actions that can be 'observed' from a process through the induced transition system has a certain inhomogeneous nature: They can be composite actions in all kinds of variations, e.g. $a, a \times *, (a \times b) \times *$. Later, we will bring more "order" into this by enforcing a simple type discipline, which will make all actions from one particular process be of the same homogeneous type, i.e. either only basic actions, or only products of basic actions, or products of products of basic actions etc. The next section will show two notable examples of sublanguages possessing such homogeneous types: CCS and a process algebra used by Winskel [95] which is very much like CCS except that synchronizations are visible. For ease of reference we call it OPA for 'Observable synchronizations Process Algebra.'

2.2.1 CCS and OPA

In this section we will show how two familiar process algebras can be embedded into our process algebra; no attempt is made to explain the intuitions behind the various operators that are given, for such arguments the interested reader is referred to Milner [59]. Instead we simply describe how the operators of these process algebras can be found as derived operators inside our process algebra, making all the results in the thesis applicable for at least any one of these algebras.

CCS

For CCS we assume that the set of basic actions is divided into a set of *names* $\mathcal{A}$, a set of *co-names* $\overline{\mathcal{A}}$, and a *silent action* τ not in $\mathcal{A}$ or $\overline{\mathcal{A}}$ (τ is not to be confused with $*$, τ is an action without identity, $*$ is 'no action'). Elements of $\mathcal{A}$ play the role of *input actions* and elements of $\overline{\mathcal{A}}$ play the role of *output actions*. We assume that there is a bijection $^- : \mathcal{A} \longrightarrow \overline{\mathcal{A}}$ the inverse of which we also denote by $^-$, hence $\overline{\overline{a}} = a$.

Now, CCS with finite summation is embedded into WPA by defining the CCS-operations of restriction $p\backslash L$, relabelling $p[f]$, and parallel composition $|$ as in figure 2.4 and taking as process terms $Proc_{\mathrm{CCS}}$ the set of terms which are generated from the subset of WPA given by excluding restriction, relabelling, and product, and including the derived CCS-operations of restriction, relabelling, and parallel composition. I.e. $Proc_{\mathrm{CCS}}$ is the set of terms constructed from the grammar:

Actions:	Act	$=$	$\mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$	
Restriction:	$p \backslash L$	$=_{\mathrm{def}}$	$p \upharpoonright (Act \setminus L)$	$L \subseteq \mathcal{A} \cup \overline{\mathcal{A}}$
Relabelling:	$p[f]$	$=_{\mathrm{def}}$	$p\{f\}$	$f : \mathcal{A} \cup \overline{\mathcal{A}} \rightharpoonup \mathcal{A} \cup \overline{\mathcal{A}}, f(\overline{a}) = \overline{f(a)}$
Composition:	$p \mid q$	$=_{\mathrm{def}}$	$(p \times q) \upharpoonright \Lambda_{CCS}\{\Xi_{CCS}\}$	

where

$$\Lambda_{CCS} = (Act \times \{*\}) \cup (\{*\} \times Act) \cup \{c \times \overline{c} \mid c \in \mathcal{A} \cup \overline{\mathcal{A}}\}$$

$$\Xi_{CCS}(x) = \begin{cases} a & \text{if } x \equiv a \times * \text{ or } x \equiv * \times a \\ \tau & \text{if } x \equiv c \times \overline{c} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Derived operational rules for $p \mid q$ (a ranges over Act, c over $\mathcal{A} \cup \overline{\mathcal{A}}$):

$$\frac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q} \qquad \frac{q \xrightarrow{a} q'}{p \mid q \xrightarrow{a} p \mid q'}$$

$$\frac{p \xrightarrow{c} p' \quad q \xrightarrow{\overline{c}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'}$$

Figure 2.4: CCS with finite summation embedded into WPA.

$$t ::= nil \mid a.t \mid t_0 + t_1 \mid t_0 \mid t_1 \mid t\backslash L \mid t[f] \mid rec\ P.t \mid P \qquad \text{(CCS)}$$

The semantics of CCS terms is given as the subset $\mathcal{T}_{\mathrm{CCS}}$ of the universal transition system $\mathcal{T}$ induced by the process terms $Proc_{\mathrm{CCS}}$.

OPA

To encode OPA we assume that the set of basic actions is partitioned into a set of *neutral actions* $\mathcal{A}$, a set of *input actions* $\{a? \mid a \in \mathcal{A}\}$, and a set of *output actions* $\{a! \mid a \in \mathcal{A}\}$. The neutral actions is to be considered as names of channels along which communication takes place. Contrary to CCS the communication taking place will be visible since synchronization on a channel a will give rise to the neutral action a being observed. The set of processes $Proc_{\mathrm{OPA}}$ of OPA is constructed from our process algebra by excluding product and including the derived OPA parallel composition $\|$ and restricting the restricting sets to subsets of Act and the relabelling functions to partial functions on Act. I.e. OPA processes are generated from the grammar:

$$t ::= nil \mid a.t \mid t_0 + t_1 \mid t_0 \parallel t_1 \mid t \restriction \Lambda \mid t\{\Xi\} \mid rec\ P.t \mid P \qquad \text{(OPA)}$$

The constructions are summarized in figure 2.5.

Now, CCS and OPA have the property that all actions taking place will be basic actions, hence no composite actions can be observed from such a process.

Proposition 2.2 *Let p be a process term in $Proc_{\mathrm{CCS}}$ (or $Proc_{\mathrm{OPA}}$). Then $R_p \subseteq Proc_{\mathrm{CCS}}$ (or $R_p \subseteq Proc_{\mathrm{OPA}}$) and for all $p' \in R_p$,*

$$p' \xrightarrow{a} \ \Rightarrow\ a \in Act \cup \{*\}.$$

Proof: Prove $p \xrightarrow{a} p' \Rightarrow p' \in Proc_{\mathrm{CCS}}\ \&\ a \in Act \cup \{*\}$ for all $p \in Proc_{\mathrm{CCS}}$ by a simple induction on derivations of transition steps. Similarly for OPA. $\square$

As CCS and OPA are embedded into WPA by simply providing abbreviations for the operators, we can freely mix operators from the two languages with operators from WPA without any semantic confusion. We will make use of this in the examples, although we mainly use just OPA.

$$\begin{array}{lll}
\textbf{Actions:} & Act & = \ \mathcal{A} \cup \{a? \mid a \in \mathcal{A}\} \cup \{a! \mid a \in \mathcal{A}\} \\
\textbf{Restriction:} & p \restriction \Lambda & \text{such that } \Lambda \subseteq Act \\
\textbf{Relabelling:} & p\{\Xi\} & \text{such that } \Xi : Act \rightharpoonup Act \\
\textbf{Composition:} & p \parallel q & =_{\mathrm{def}} \ (p \times q) \restriction \Lambda_{WPA}\{\Xi_{WPA}\}
\end{array}$$

where

$$\Lambda_{WPA} = (Act \times \{*\}) \cup (\{*\} \times Act) \cup \{a? \times a!, a! \times a? \mid a \in \mathcal{A}\}$$

$$\Xi_{WPA}(x) = \begin{cases} a & \text{if } x \equiv a \times * \text{ or } x \equiv * \times a \\ & \text{or } x \equiv a? \times a! \text{ or } x \equiv a! \times a? \\ \text{undefined} & \text{otherwise} \end{cases}$$

Derived operational rules for $p \parallel q$ (a ranges over Act, c over $\mathcal{A}$):

$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'}$$

$$\frac{p \xrightarrow{c?} p' \quad q \xrightarrow{c!} q'}{p \parallel q \xrightarrow{c} p' \parallel q'} \qquad \frac{p \xrightarrow{c!} p' \quad q \xrightarrow{c?} q'}{p \parallel q \xrightarrow{c} p' \parallel q'}$$

Figure 2.5: OPA embedded into our process algebra.

2.2.2 Static Processes

When considering finite-state processes, i.e. processes which induces finite transition systems, a certain canonical form which we will call *static processes* are often used:

Definition 2.4 A *regular process* is a process constructed entirely from the dynamic operators, i.e. from

$$nil, a.t, t_0 + t_1, \ rec\ P.t, \text{ and } P,$$

such that P is always strongly guarded. Hence, excluded are the static operators of restriction, relabelling, and product. A *static process* p of *order* n is a process

$$p \equiv op(p_1, p_2, \dots, p_n)$$

where for $1 \leq i \leq n$, p_i is a regular process, and *op* is any 'parallel' operator generated from the static operators. □

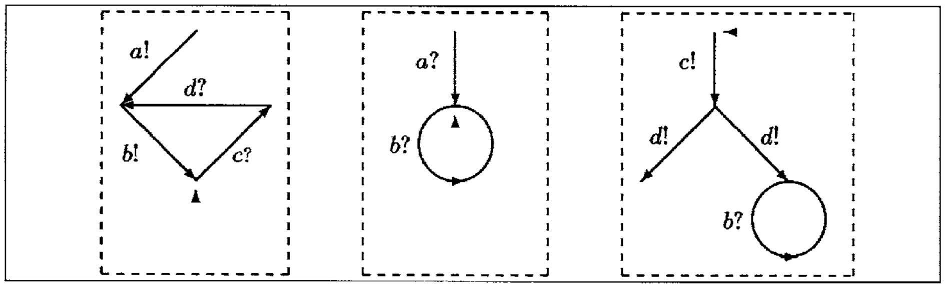


Figure 2.6: The process $S = ((a!rec\ P.b!c?d?P) \parallel (a?rec\ Q.b?Q) \parallel c!(d!+d!rec\ R.b?R)) \restriction \mathcal{A}$. A state of the static process S corresponds to a state of each of the three subprocesses. An example is indicated by the small arrow heads.

The parallel operators of CCS and OPA provide examples of operators yielding static processes, for example for CCS:

$$(p_1|p_2|\dots|p_n) \restriction \Lambda$$

We use the term *static* because of the role of the static operators and because the number of parallel processes in such a system is fixed to n. Moreover, a static process is always finite-state as can be seen by applying proposition 2.1. From a pragmatic point of view static processes have a number of nice properties that make them interesting:

- The size of the transition system induced by a static process can be exponentially bigger than the description as a process term making them well-suited for showing lower bound complexity results. We will later in chapter 9 see an example of this.
- They have a simple graphical representation resembling flow graphs, see figure 2.6.
- They provide simple, generic examples of parallel systems. Any method or algorithm hoping to perform well on all terms of the process algebra, should at least perform well on static processes, so they provide test examples.

Actually, many of the finite-state examples used in the literature fall into the category of static processes.

Remark 2.2 (Simultaneously defined processes.) Instead of the recursion operator we are using, it is common to define recursive processes by a set of simultaneous equations

$$\Sigma : \begin{array}{rcl} P_1 & = & t_1 \\ & \vdots & \\ P_n & = & t_n \end{array}$$

where the free process identifiers on the right-hand side are among $\{P_1, \ldots, P_n\}$. Now, given such a system Σ, the following operational rule is added, allowing state identifiers to be unfolded

$$\frac{t \xrightarrow{a}_{\Sigma} t'}{P \xrightarrow{a}_{\Sigma} t'} \quad (P = t) \in \Sigma$$

The same effect can be achieved with the recursion operator by constructing a process term P^{Σ} for each P only involving the recursion operator. This can be done as follows:

$$P^{\Sigma} = \begin{cases} rec\ P.(t^{\Sigma \setminus (P=t)}) & \text{if } (P = t) \in \Sigma \\ P & \text{otherwise} \end{cases}$$

$$\begin{array}{rclrcl} (a.t)^{\Sigma} & = & a.(t^{\Sigma}) & (t_0 + t_1)^{\Sigma} & = & t_0^{\Sigma} + t_1^{\Sigma} \\ (t_0 \times t_1)^{\Sigma} & = & t_0^{\Sigma} \times t_1^{\Sigma} & (t \restriction \Lambda)^{\Sigma} & = & (t^{\Sigma}) \restriction \Lambda \\ (t\{\Xi\})^{\Sigma} & = & (t^{\Sigma})\{\Xi\} & & & \end{array}$$

Now, it is not hard to see that any term t interpreted with respect to Σ will have the same behaviour as the translation t^{Σ}, although the names of the states will be different and the translated term can generate bigger transition systems. (See e.g. Milner [59, sec. 2.9].) In view of this close relationship we often use the equational formulation in examples, but base our theoretical considerations on the recursion operator. □

2.3 The Modal μ-Calculus

As mentioned in the introduction Kozen's (propositional) modal μ-calculus, often referred to as μK, has expressive power subsuming many modal and temporal logics when expressiveness is measured as the ability to express subsets of states of transition systems. To be more precise, when considering logics over labelled transition systems, we will say that a logic $\mathcal{L}$ is *more expressive than* a logic $\mathcal{M}$ if for every assertion A of $\mathcal{L}$ there exists an assertion $A_{\mathcal{M}}$ of $\mathcal{M}$ such that for all labelled transition systems T,

$$[\![A]\!]_T^{\mathcal{L}} = [\![A_{\mathcal{M}}]\!]_T^{\mathcal{M}}$$

where $[\![A]\!]_T^{\mathcal{L}}$ is the set of states of T satisfying A and similarly for $[\![A_{\mathcal{M}}]\!]_T^{\mathcal{M}}$. With this definition of expressiveness it has been shown that μK is more expressive than for instance propositional dynamic logic PDL, computation tree logic CTL, and the extensions of CTL called CTL* and ECTL.

However, the proofs of these results will in general require very big assertions in μK and says nothing about the ability to *concisely* – in terms of size – express properties. Actually, it seems that for certain properties, CTL* for instance provides much smaller assertions than μK – on the other hand some assertions of μK are not even expressible in CTL*! (We consider the relationship between CTL* and μK in more detail in section 4.4.1.)

For automatic verification methods using model checking as described in chapter 5, efficiency is a central issue and as the size of the assertions under consideration gives a significant contribution to the overall complexities of the model checking algorithms, it is of main concern to keep assertions small. We will also be faced with these problems in chapter 3 since the compositional method based on reductions has a tendency to generate large assertions.

For these reasons we define, after the description of Kozen's calculus which we will refer to as the *standard calculus*, an *extended calculus* which provides compact assertions by two means:

- by allowing simultaneous fixed-points, and
- by having a first-order predicate logic on actions.

The quantification present in the first-order predicate logic on actions adds to the expressiveness in a very convenient way: We can express infinite disjunctions over all basic actions as a finite assertion. But first we look at the standard calculus.

2.3.1 The Standard Calculus

Assertions A in the *standard calculus* referred to as μK, is generated from the grammar

$$A ::= F \mid \neg A \mid A_0 \vee A_1 \mid \langle a \rangle A \mid X \mid \mu X.A$$

where a is an action in Act. The variable X ranges over a set of *assertion variables AssnVar*, and the usual notion of free and bound variables will be used with $\mu X.A$ as a binding occurrence of X. The minimum fixed-point assertions $\mu X.A$ are formed subject to the well-formedness criterion of *syntactic monotonicity*, that is, any free occurrence of X in A is under an even number of negations. Using the negation we can dualize every operator to get some very common abbreviations:

$$\begin{array}{rclcrcl} T & = & \neg F & \quad & A_0 \wedge A_1 & = & \neg(\neg A_0 \vee \neg A_1) \\ [a]A & = & \neg\langle a \rangle \neg A & & \nu X.A & = & \neg \mu X. \neg A[\neg X/X], \end{array}$$

where A in $\nu X.A$ must be syntactic monotone in X, and $A[B/X]$ denotes substitution. Notice, that requiring syntactic monotonicity of A also ensures syntactic monotonicity of $\neg A[\neg X/X]$. The modalities $\langle a \rangle$ and $[a]$ (pronounced diamond-a respectively box-a) gives the logic its ability to express progress, $\langle a \rangle A$ will hold at states which has an a-successor satisfying A and $[a]A$ will hold at states with the property that all a-successors satisfy A.

The semantics will be given relative to a transition system T as a map $[\![\ ]\!]_T$ taking each assertion to a set of states of T, i.e. to an element of the powerset $\mathcal{P}(S)$ where S is the states of T.

However, due to the possibility of free variables the interpretation of assertions will be given relative to an environment ρ assigning a subset of S to each variable of $AssnVar$. We will use $\rho[U/X]$ to denote the environment which is like ρ except that X is mapped to U. The interpretation of A denoted $[\![A]\!]_T\rho$ is defined inductively on the structure of A as follows (we leave out subscript T for brevity):

$$\begin{array}{rcl} [\![F]\!]\rho & = & \emptyset \\ [\![A_0 \vee A_1]\!]\rho & = & [\![A_0]\!]\rho \cup [\![A_1]\!]\rho \\ [\![\langle a \rangle A]\!]\rho & = & \{s \in S \mid \exists s' \in S.\ s \xrightarrow{a} s' \ \&\ s' \in [\![A]\!]\rho\} \\ [\![X]\!]\rho & = & \rho(X) \\ [\![\mu X.A]\!]\rho & = & \mu\psi \text{ where } \psi : U \mapsto [\![A]\!]\rho[U/X] \end{array}$$

The semantics of minimum fixed points is based on Tarski's theorem:

Theorem 2.1 (Tarski [82]) *Let $(D, \leq)$ be a complete lattice and $\psi : D \to D$ a monotonic function on D. Then ψ has a minimum fixed-point $\mu\psi$ given by*

$$\mu\psi = \bigwedge\{x \in D \mid \psi(x) \leq x\}$$

and a maximum fixed-point $\nu\psi$ given by

$$\nu\psi = \bigvee\{x \in D \mid x \leq \psi(x)\}$$

Often an element x with the property $\psi(x) \leq x$ is called a *pre-fixed point of ψ*. Since in particular a fixed-point like $\mu\psi$ is a pre-fixed point, $\mu\psi$ is the *least pre-fixed point of ψ*. Similarly, an element x with $x \leq \psi(x)$ is called a *post-fixed point of ψ* and $\nu\psi$ is the *largest post-fixed point of ψ*.[4]

Now, due to the syntactic monotonicity condition it is not hard to see that the map ψ used in the semantic clause for $\mu X.A$ is monotonic on $(\mathcal{P}(S), \subseteq)$ and Tarski's theorem can be applied to give a minimum fixed-point $\mu\psi$.

$$\boxed{\begin{array}{rcl}
[\![T]\!]\rho & = & S \\
[\![A_0 \wedge A_1]\!]\rho & = & [\![A_0]\!]\rho \cap [\![A_1]\!]\rho \\
[\![[a]A]\!]\rho & = & \{s \in S \mid \forall s' \in S.\ s \xrightarrow{a} s' \ \Rightarrow\ s' \in [\![A]\!]\rho\} \\
[\![\nu X.A]\!]\rho & = & \nu\psi \text{ where } \psi : U \mapsto [\![A]\!]\rho[U/X]
\end{array}}$$

Figure 2.7: Derived semantic clauses.

For convenience of reference we give the derived semantic clauses for the dual operators in figure 2.7. In section 2.5 we will try to provide a little intuition about what can be expressed with the fixed-points by giving some simple examples, but first we turn to the extended calculus.

[4]There is no consensus in the literature to whether this is the right way around to define pre- and post-fixed points. Many authors including Milner [59, p. 104] and authors from the area of abstract interpretation use the opposite meaning of pre- and post-fixed points. Here we are following Plotkin [71], Gunter [41], and Winskel [95]. Curiously enough, in the algorithmics community, completely different notions are used; ψ is said to be *inflationary* (respectively *deflationary*) *on x* if x is a post-fixed point (respectively pre-fixed point) of ψ, cf. Cai and Paige [20, p. 202].

2.3.2 The Extended Calculus

In the extended calculus, referred to as $\mu\mathrm{K}_{ext}$, we allow *action variables* taken from an infinite set *ActVar* and ranged over by $\alpha, \beta, \ldots$. Using basic actions and action variables *composite action expressions* can now be generated from the grammar

$$\gamma ::= * \mid a \mid \alpha \mid \gamma_0 \times \gamma_1$$

where a ranges over basic actions *Act*. Given an *action variable environment*

$$\phi \;:\; \mathit{ActVar} \rightarrow \mathit{Act} \cup \{*\}$$

any composite action expression can now be taken to denote a composite action in Act_* by substituting $\phi(\alpha)$ for action variable α. Let $[\![\gamma]\!]\phi$ denote the result of performing this substitution on γ.

Assertions in the extended calculus is generated from the following grammar:

$$A ::= Q \mid \neg A \mid A_0 \vee A_1 \mid \langle\gamma\rangle A \mid \exists\alpha.A \mid \psi(\gamma) \mid X \mid \vec{P}^1$$

where $\vec{P}^{\,n}$ is an n-ary product of assertions, possibly with `where`-clauses,

$$\vec{P}^{\,n} \;::=\; \vec{A}^{\,n} \;\mid\; \vec{A}^{\,n} \;\mathtt{where}_\mu\; \vec{X}^{\,m} \;=\; \vec{P},^{\,m}$$

where we have used the notation $\vec{A}^{\,n}$ as an abbreviation for the n-tuple $(A_1, \ldots, A_n)$ and $\vec{X}^{\,m}$ for the m-tuple $(X_1, \ldots, X_m)$. When obvious from context or irrelevant we often leave out the arity and simply write $\vec{A}$ and $\vec{X}$. We again require that in a $\mathtt{where}_\mu$-clause $\vec{P}^{\,m}$ is syntactic monotone in $\vec{X}^{\,m}$.

To avoid too extensive use of parentheses we assume that the operators bind with decreasing strength as follows: $\neg, \langle\gamma\rangle, \vee, \exists\alpha, \mathtt{where}_\mu$.

Several new constructions have been added.

Constants. The assertion Q is a *constant assertion* (sometimes called *basic* or *atomic* assertion) ranging over a set of constants *Const*. These constants are supposed to denote simple properties that are not definable directly in the μ-calculus. In order to give semantics of constants, we must assume that we have given a *valuation* $V : \mathit{Const} \to \mathcal{P}(S)$ relative to the set of states S of the transition system under consideration. In particular we will assume the presence of a universal valuation $\mathcal{V}$ of $\mathcal{T}$.

Quantification on actions and action predicates. The assertion $\exists\alpha.A$ is an *existential quantification* over all basic actions and the idling action. The *action predicates* ψ will include *action tests* $\alpha = a$ and membership tests $\alpha \in \Lambda$ for finite sets Λ, but we will a priori not restrict attention to any particular language of predicates. The action quantifiers and the action predicates together with negation and disjunctions forms a *first-order predicate logic on actions.*

The dual of the existential quantifier is the universal quantifier $\forall$ defined as

$$\forall\alpha.A =_{\text{def}} \neg\exists\alpha.\neg A.$$

For the technics reasons we have chosen to include the idling action in the range of quantification. It will, however, sometimes be useful to exclude it from the quantification and we introduce the abbreviations $\tilde{\exists}$ and $\tilde{\forall}$ for this purpose. They are defined as

$$\tilde{\exists}\alpha.A =_{\text{def}} \exists\alpha.(\alpha \neq *) \wedge A \qquad \tilde{\forall}\alpha.A =_{\text{def}} \forall\alpha.(\alpha \neq *) \to A.$$

Simultaneous fixed-points. The construction adding simultaneous fixed-points is inspired by Park [69]. An assertion B $\texttt{where}_\mu$ $\vec{X}^n = \vec{A}^n$ is going to denote the same as the assertion B when considered in an environment where $\vec{X}^n$ is the simultaneous, minimum solution to the equation $\vec{X}^n = \vec{A}^n$ or analogously $\vec{X}^n$ is the minimum fixed-point of the function of $\vec{X}^n$ described by $\vec{A}^n$. The dual of $\texttt{where}_\mu$ denoted by $\texttt{where}_\nu$ is defined as the following abbreviation:

$$(\ \vec{A}^{\,n}\ \texttt{where}_\nu\ \vec{X}^{\,m} = \vec{P}^{\,m}\ ) = (\ \vec{A}^{\,n}\ [\neg\vec{X}^{\,m}\ /\ \vec{X}^{\,m}\ ]\ \texttt{where}_\mu\ \vec{X}^{\,m} = \neg\vec{P}^{\,m}\ [\neg\vec{X}^{\,m}\ /\ \vec{X}^{\,m}\ ])$$

where negation of a product and substitution on a product is coordinatewise. The presence of simultaneous fixed-points that can be nested makes it possible to write down very complicated – and unreadable expressions. Luckily in most examples, simple, unary fixed-points will suffice; the more complicated simultaneous fixed-points will only appear as the result of various transformations on assertions.

$$\begin{array}{rcl}
[\![\neg A]\!]_V\rho\,\phi & = & S \setminus [\![A]\!]_V\rho\,\phi \\
[\![A_0 \vee A_1]\!]_V\rho\,\phi & = & [\![A_0]\!]_V\rho\,\phi \cup [\![A_1]\!]_V\rho\,\phi \\
[\![[\gamma]A]\!]_V\rho\,\phi & = & \{s \in S \mid \forall s' \in S.\ s \overset{[\![\gamma]\!]\phi}{\to} s' \ \Rightarrow\ s' \in [\![A]\!]_V\rho\,\phi\} \\
[\![\exists\alpha.A]\!]_V\rho\,\phi & = & \{s \in S \mid \exists a \in \mathit{Act} \cup \{*\}.s \in [\![A]\!]_V\rho\,\phi[a/\alpha]\} \\
[\![\psi(\gamma)]\!]_V\rho\,\phi & = & \begin{cases} S & \text{if } \psi([\![\gamma]\!]\phi) \\ \emptyset & \text{otherwise} \end{cases} \\
[\![Q]\!]_V\rho\,\phi & = & V(Q) \\
[\![X]\!]_V\rho\,\phi & = & \rho(X) \\
 & & \\
[\![\vec{A}^{\,n}]\!]_V\rho\,\phi & = & ([\![A_1]\!]_V\rho\,\phi, \ldots, [\![A_n]\!]_V\rho\,\phi) \\
[\![\vec{A}^{\,n}\ \texttt{where}_\mu\ \vec{X}^{\,m} = \vec{P}^{\,m}]\!]_V\rho\,\phi & = & ([\![A_1]\!]_V\rho'\,\phi, \ldots, [\![A_n]\!]_V\rho'\,\phi) \\
 & & \quad \text{where} \quad \rho' \ = \ \rho[\mu\vec{\psi}/\vec{X}^{\,m}] \\
 & & \qquad\qquad \vec{\psi}(\vec{U}) \ = \ [\![\vec{P}^{\,m}]\!]_V\rho[\vec{U}/\vec{X}^{\,m}]\,\phi
\end{array}$$

Figure 2.8: Semantics of $\mu\mathrm{K}_{ext}$ relative to a labelled transition system $T = (S, L, \rightarrow)$ with valuation V.

To give the formal semantics we must assume given a labelled transition system T, a valuation for the constants V, an environment for assertion variables ρ, and finally an environment for action variables ϕ. Now, any assertion A of the extended calculus denotes an element $[\![A]\!]_{T,V}\rho\phi \subseteq S$, where S is the states of T. The semantics is defined inductively on A in figure 2.8.

In the extended calculus the scope of the binding of a variable in a `where`-clause has become slightly more complicated, so we give the full definition of the function FV giving the free assertion variables of an assertion. It is defined inductively as follows:

$$\begin{array}{rcl}
FV(\neg A) = FV(\langle\gamma\rangle A) = FV(\exists\alpha.A) & = & FV(A) \\
FV(A_0 \vee A_1) & = & FV(A_0) \cup FV(A_1) \\
FV(\psi(\gamma)) = FV(Q) & = & \emptyset \\
FV(X) & = & \{X\} \\
FV\ (\vec{A}^{\,n}\ ) & = & \bigcup_{1\le i\le n} FV(A_i) \\
FV(\ \vec{A}^{\,n}\ \ \mathtt{where}_\mu\ \vec{X}^{\,m}\ =\ \vec{P}^{\,m}\ ) & = & \\
\multicolumn{3}{r}{(\bigcup_{1\le i\le n} FV(A_i) \cup FV(\ \vec{P}^{\,m}\ )) \setminus \{X_1, \dots, X_m\}}
\end{array}$$

As an example we have

$$FV(X \vee Y\ \mathtt{where}_\mu\ Y = (X\ \mathtt{where}_\nu\ X = X \vee Y)) = \{X\}$$

and

$$FV((X \vee Y\ \mathtt{where}_\mu\ Y = X)\ \mathtt{where}_\nu\ X = X \vee Y) = \{Y\}$$

where in the last example the first Y is bound by the $\mathtt{where}_\mu$-clause and the Y appearing in the $\mathtt{where}_\nu$-clause is free.

By considering $\mu X.A$ from the standard calculus as merely an abbreviation

$$\mu X.A\ \equiv_{\text{def}} (X\ \mathtt{where}_\mu\ X = A)$$

it is justified to call the extended calculus an *extension* of the standard calculus. For the other way around, if we consider the fragment of the extended calculus arising from adding to the standard calculus only the where-construction, i.e. the standard calculus with simultaneous fixed-points which we refer to as $\mu\mathrm{K}_{\mathtt{where}}$, nothing has been gained in logical expressive power. This is an easy consequence of Bekič's theorem which transforms simultaneous fixed-points to simple, unary fixed-points.

Theorem 2.2 (Bekič's theorem [12])
Let D and E be complete lattices, and $f : D \times E \to D$ and $g : D \times E \to E$ monotonic functions, then

$$\mu(x,y).(f(x,y), g(x,y)) = (\mu x.f(x, \mu y.g(x,y)), \mu y.g(\mu x.f(x,y), y)).$$

In the n-ary version it is slightly more complicated to state:

Theorem 2.3 (Bekič's theorem for n-ary fixed-points) *Let $D_1, \ldots, D_n$ be complete lattices and let $f_i : D_1 \times \ldots \times D_n \to D_i, 1 \leq i \leq n$ be monotonic maps. Then*

$$\mu(x_1, \ldots, x_n).(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)) = (E_1^{\emptyset}, \ldots, E_n^{\emptyset})$$

where

$$E_i^J = \begin{cases} x_i & \textit{if } i \in J \\ \mu x_i.f_i(E_1^{J\cup\{i\}}, \ldots, E_n^{J\cup\{i\}}) & \textit{otherwise} \end{cases}$$

The idea of $E_j^{\emptyset}$ is roughly: "on any path from the top-expression $E_j^{\emptyset}$ every occurrence of a variable x_i must be within an expression $\mu x_i.f_i(\ldots)$". Hence, J collects the set of variables that have already been bound by a μ.

Proof (Theorem 2.3): We prove by induction on m that for all $m \in \underline{n} = \{1, 2, \ldots, n\}$, $Q(m)$ holds, where $Q(m) \Leftrightarrow_{\text{def}}$

$$\forall\{i_1, \ldots, i_m\} \subseteq \underline{n}.\ \mu(x_{i_1}, \ldots, x_{i_m}).(f_{i_1}(\vec{x}), \ldots, f_{i_m}(\vec{x})) = (E_{i_1}^J, \ldots, E_{i_m}^J)$$
$$\text{where } J = \underline{n} \setminus \{i_1, \ldots, i_m\}.$$

For the base case, $Q(1)$ we must argue

$$\forall i \in \underline{n}.\ \mu x_i.f_i(\vec{x}) = E_i^{\underline{n}\setminus i}$$

but by definition of E_i^J we have

$$\begin{aligned} E_i^{\underline{n}\setminus i} &= \mu x_i.f_i(x_1, \ldots, E_i^{\underline{n}}, \ldots, x_n) \\ &= \mu x_i.f_i(x_1, \ldots, x_i, \ldots, x_n) \\ &\qquad \text{as } i \in \underline{n}. \end{aligned}$$

For the inductive step assume $Q(m)$ holds for $m < n$, and assume given a set of indices $\{i_1, \ldots, i_{m+1}\} \subseteq \underline{n}$. By Bekič's theorem for binary fixed-points we get

$$\mu(x_{i_1}, \ldots, x_{i_{m+1}}).(f_{i_1}(\vec{x}), \ldots, f_{i_{m+1}}(\vec{x})) \tag{2.3}$$

$$= (\mu x_{i_1}.f_{i_1}(\vec{e}), \mu(x_{i_2}, \dots, x_{i_{m+1}}).(f_{i_2}(\vec{x'}), \dots, f_{i_{m+1}}(\vec{x'})))$$

where

$$\vec{x'} = (x_1, \dots, \mu x_{i_1}.f_{i_1}(\vec{x}), \dots, x_n)$$

$$e_i = \begin{cases} \pi_j(\mu(x_{i_2}, \dots, x_{i_{m+1}}).(f_{i_2}(\vec{x}), \dots, f_{i_m}(\vec{x})) & \text{if } i \in \{i_2, \dots, i_{m+1}\} \\ & \text{and } i \text{ has } j'\text{th position in } x_{i_2}, \dots, x_{i_{m+1}} \\ x_i & \text{otherwise} \end{cases}$$

By the induction hypothesis we have that if $i \in \{i_2, \dots, i_{m+1}\}$, then

$$e_i = E_i^{J \cup \{i_1\}}$$

where $J = \underline{n} \backslash \{i_1, \dots, i_{m+1}\}$. Moreover, for $i \notin \{i_1, \dots, i_{m+1}\}$ i.e. $i \in J \cup \{i_1\}$, then $e_i = x_i = E_i^{J \cup \{i_1\}}$, hence the first component of (2.3) equals

$$\mu x_{i_1}.f_{i_1}(E_1^{J \cup \{i_1\}}, \dots, E_n^{J \cup \{i_1\}}) = E_{i_1}^J$$

by definition of $E_{i_1}^J$. By symmetry, similar arguments hold for the other components, thereby proving $Q(m+1)$. □

Corollary 2.1 *For any ussertion A in $\mu K_{\texttt{where}}$ there exists a logical equivalent assertion A' in μK.*

Proof: (Sketch) Use Bekič's theorem to transform any `where`-clause into unary fixed-points ($A \; \texttt{where}_\mu \; X = B$) and replace this by $A[\mu X.B/X]$. □

However, the resulting assertion in μK can be exponentially bigger than the original assertion in $\mu\mathrm{K}_{\texttt{where}}$ due to the use of Bekič's theorem and the elimination of the sharing provided by the `where`-clauses, so this result is mainly of theoretical interest. When it comes to actually writing down and working with assertions the simultaneous fixed-points provide "exponentially more compact" representations.

As concerns the other extensions, the constants add to the expressive power in a rather trivial – but useful – way parameterized by the valuations; but the quantification on actions adds expressiveness in a significant way. They allow for *infinite disjunctions* indexed by actions, something that cannot be expressed in the standard calculus. In the rather pathological case where *Act* is chosen to be *finite*, quantification on actions can be replaced by a finite disjunction in the standard calculus and at least for closed assertions the action predicates can be resolved and removed altogether. We will in general assume that *Act* is *infinite*, hence the quantification on actions *does* add to the expressiveness. We return to this issue in section 2.7.

Figure 2.9 summarizes a set of simple abbreviations used throughout the text.

For future reference we define:

$$\begin{array}{lcl lcl}
A \wedge B & = & \neg(\neg A \vee \neg B) & A \rightarrow B & = & \neg A \vee B \\
A \leftrightarrow B & = & (A \rightarrow B) \wedge (B \rightarrow A) & [\gamma]A & = & \neg\langle\gamma\rangle\neg A \\
\nu X.A & = & \neg\mu X.\neg A[\neg X/X] & \forall\alpha.A & = & \neg\exists\alpha.\neg A \\
\tilde{\exists}\alpha.A & = & \exists\alpha.(\alpha \neq *) \wedge A & \tilde{\forall}\alpha.A & = & \forall\alpha.(\alpha = *) \vee A \\
\langle.\rangle A & = & \tilde{\exists}\alpha.\langle\alpha\rangle A, \quad \textit{for some } \alpha \notin fv(A) & & & \\
\langle-\gamma\rangle A & = & \tilde{\exists}\alpha.(\alpha \neq \gamma) \wedge \langle\alpha\rangle A, \quad \textit{for some } \alpha \notin fv(A) \cup fv(\gamma) & & &
\end{array}$$

Figure 2.9: Some simple abbreviations in $\mu\mathrm{K}_{ext}$.

Definition 2.5 The *size* of an assertion A denoted $|A|$ is the number of operators, variables and constants in A, i.e.

$$\begin{array}{rcl rcl}
|*| = |a| = |\alpha| & = & 1 & |\gamma_0 \times \gamma_1| & = & 1 + |\gamma_0| + |\gamma_1| \\
|Q| = |X| & = & 1 & |\neg A| & = & 1 + |A| \\
|A_0 \vee A_1| & = & 1 + |A_0| + |A_1| & |\langle\gamma\rangle A| & = & 1 + |\gamma| + |A| \\
|\exists\alpha.A| & = & 2 + |A| & |\psi(\gamma)| & = & 1 + |\gamma| \\
|\vec{A}^n \ \mathtt{where}_\mu \ \vec{X}^m = \vec{P}^m| & = & 1 + m + |\vec{A}^n| + |\vec{P}^m| & |\vec{A}^n| & = & \sum_{i=1}^n |A_i|
\end{array}$$

□

2.4 Simple Properties of the Logics and Models

2.4.1 Types of Actions

The set of composite actions Act_* consists of actions of all kinds of products of basic and idling actions. We will classify these actions according to a very simple notion of *types*.

Definition 2.6 Let *Types* be the free $\cdot$, $\times$-algebra, i.e. the least set containing $\cdot$ and closed under the rule that if $\sigma, \sigma' \in Types$ then $\sigma \times \sigma' \in Types$. Define the type of action of an action a as follows:

$$\begin{array}{lcll} type(a) & = & \cdot & \text{if } a \in Act \cup \{*\} \\ type(a) & = & type(b) \times type(c) & \text{if } a \equiv b \times c \end{array}$$

□

Hence *type* determines the "tree-structure of actions." For example $type(* \times (a \times b)) = \cdot \times (\cdot \times \cdot)$ if a and b are basic actions. This definition is trivially extended to action expressions by taking the of action variables to be $\cdot$. Now, the type of an assertion A is the set of types of modalities appearing inside A, i.e. $type(A) \subseteq Types$. We will say that an assertion A is of *monotype* if there exists a $\sigma \in Types$ s.t. $type(A) \subseteq \{\sigma\}$, and *has empty type* if $type(A) = \emptyset$. The notion of types can also be extended to type of processes:

Definition 2.7 Let the sort $sort(p)$ of a process p be defined as the set of actions $a \neq *$, s.t. there exists $p' \in R_p$ with $p' \xrightarrow{a}$. We will say that a *process p has monotype σ* if

$$type(sort(p)) \subseteq \{\sigma\}$$

□

Not all processes of WPA are monotyped, for example

$$type(sort(a + b \times c)) = type(\{a, b \times c\}) = \{\cdot, \cdot \times \cdot\}.$$

However, proposition 2.2 can be now be restated as *all processes of CCS and OPA have monotype $\cdot$*.

An assertion of type $\cdot \times \cdot$. will, when interpreted over processes $p \times q$ really denote a *relation* between the states R_p and R_q of the transition systems pointed by p and q. This observation will be used in chapter 4 to express relations like equivalences and preorders familiar from the work on especially CCS and other process algebras as assertions in the modal μ-calculus.

2.4.2 Satisfaction

It is time to introduce notation for expressing that a process satisfies an assertion.

Definition 2.8 A *correctness assertion* $(p : A)$ is *well-formed* if p and A are both monotypable with the same type. We define *satisfaction* of well-formed correctness assertions as follows:

$$\begin{array}{rcll}
\models_{T,V,\rho,\phi} p : A & \Leftrightarrow_{def} & p \in [\![A]\!]_{T,V}\rho\ \phi & \\
\models_{T,V} p : A & \Leftrightarrow_{def} & \models_{T,V,\rho,\phi} p : A & \text{for all environments } \rho \text{ and } \phi \\
\models_{T} p : A & \Leftrightarrow_{def} & \models_{T,V} p : A & \text{for all valuations } V \text{ on } T \\
\models p : A & \Leftrightarrow_{def} & \models_{\mathcal{T},\mathcal{V}} p : A & \\
\models_{\rho} p : A & \Leftrightarrow_{def} & \models_{\mathcal{T},\mathcal{V},\rho,\phi} p : A & \text{for all } \phi
\end{array}$$

□

We have defined the semantics of the two calculi relative to a particular transition system. One choice of transition system is the universal transition system $\mathcal{T}$, representing the global operational behaviour of WPA, hence, given a universal valuation $\mathcal{V} : Const \to \mathcal{P}(Proc_{WPA})$ then for a closed assertion A of $\mu\mathrm{K}_{ext}$ and arbitrary ρ and ϕ, $[\![A]\!]_{\mathcal{T},\mathcal{V}}\rho\ \phi$ is the set of *all processes satisfying A*. This is certainly a big set and for verification purposes a more local view considering only 'small' transition systems pointed by processes is more relevant. The soundness of viewing things locally in this way is ensured by the following lemma and a corollary.

Lemma 2.2 (Locality lemma) *Let $T = (S, L, \to)$ be a transition system. Given an assertion A, an assertion variable environment ρ, an action variable environment ϕ, a valuation V, and a subset U of S. Suppose U is closed under $\to$. Let*

$$T_U = (U, L, \to \cap(U \times L \times U)).$$

Then the following equality holds

$$[\![A]\!]_{T_U,V_U}\rho_U\ \phi = ([\![A]\!]_{T,V}\rho\ \phi) \cap U,$$

where $V_U(Q) = V(Q) \cap U$ and $\rho_U(X) = \rho(X) \cap U$.

Proof: Straightforward using structural induction on A (for the fixed-points something relating fixed-points in different lattices, like the reduction lemma to be introduced in chapter 3, is needed). $\square$

Corollary 2.2 *Let p be a process term, and A a closed assertion. Define $\mathcal{V}_p(Q) = \mathcal{V}(Q) \cap R_p$ for all constants Q. Then*

$$\models_{\mathcal{T},\mathcal{V}} p : A \Leftrightarrow \models_{\mathcal{T}_p,\mathcal{V}_p} p : A$$

Proof: Take $U = R_p$ in locality lemma. $\square$

Hence corollary 2.2 ensures that the local and global views give the same notion of satisfaction.

$$\boxed{\begin{array}{rclrcl}
W(nil) & = & \mathbf{true} & W(a.t) & = & \mathbf{false} \\
W(t_0 + t_1) & = & W(t_0) \wedge W(t_1) & W(t_0 \times t_1) & = & W(t_0) \wedge W(t_1) \\
W(t \upharpoonright \Lambda) & = & W(t) & W(t\{\Xi\}) & = & W(t) \\
W(rec\ P.t) & = & W(t) & W(P) & = & \mathbf{true} \\
\multicolumn{6}{c}{\mathcal{V}(\mathit{WellTerm}) = \{p \mid W(p)\}}
\end{array}}$$

Figure 2.10: Well-terminated processes in WPA.

2.5 Example: Deadlock

Having spent some time on defining the logic and our language of processes we now turn to some simple examples. The classical example is that of the absence of *deadlock* in a concurrent system. Assume we have given a constant *WellTerm* with denotation the set of *well-terminated states* of a given transition system, for WPA it could be defined as in figure 2.10. Then a *deadlock* is characterized as a state which cannot perform any actions, yet not being well-terminated. This is expressed by the assertion

$$\mathit{DeadLock} =_{\mathrm{def}} [.]F \wedge \neg\, \mathit{WellTerm}$$

and absence of deadlock is expressed by

$$\mathit{DeadLockFree} =_{\mathrm{def}} \mathit{Never(DeadLock)}$$

where

$$\mathit{Never}(X) = \neg \mathit{EEven}(X)$$

and *EEven* abbreviating *eventually* (the prefix E will be explained later) is defined as

$$\mathit{EEven}(X) = \mu Y.X \vee \langle . \rangle Y.$$

Notice, that an alternative definition of $\mathit{Never}(X)$ could be as $\mathit{AAlways}(\neg X)$ (again just ignore the prefix A for the moment) using the maximum fixed-point assertion

$$\mathit{AAlways}(X) = \nu Y.X \wedge [.]Y$$

which is easily shown to be equivalent to the above definition (by 'pushing the negation inwards'). Deadlock-freeness is often referred to as a *safety property* because it has the flavour of 'something bad never happens'.

Example 2.1 (A simple message handling system in OPA.) Consider a very simple system consisting of a *sender* S which sends messages and awaits acknowledgements, a *receiver* R which receives messages and awaits acknowledgements, and a *medium* M through which messages and acknowledgements are sent. The three processes could be defined by

$$\begin{array}{rcl} S & = & \mathit{send!ack_s?S} \\ M & = & \mathit{send?rec!M} + \mathit{ack_r?ack_s!M} \\ R & = & \mathit{rec?ack_r!R} \end{array}$$

and the complete system is

$$\mathit{Sys} = (S \,\|\, M \,\|\, R) \upharpoonright \{\mathit{send}, \mathit{rec}, \mathit{ack}_r, \mathit{ack}_s\}$$

We use the restriction to prevent the system from communicating with the environment – the only observable actions outside the system are the neutral actions arising from communication. Let us also consider a slightly different system with a faulty medium that sometimes breaks down. This is modelled by the *silent action* τ from CCS as follows:

$$M' = \mathit{send}?(\mathit{rec}!M' + \tau) + \mathit{ack}_r?\mathit{ack}_s!M'$$

Hence, after having received a message, M' can decide to break down. The system is now

$$\mathit{Sys}' = (S \,\|\, M' \,\|\, R) \upharpoonright \mathcal{A}$$

where for convenience we simple restrict the visible behaviour to the neutral actions $\mathcal{A}$. Now, alternatively we could assume that the medium could get into a state where instead of breaking down it will be occupied with internal "chatter" and prevent itself from ever delivering the message. This could be modelled by the medium M'':

$$\begin{array}{rcl} M'' & = & \mathit{send}?(\mathit{rec}!M'' + \tau L) + \mathit{ack}_r?\mathit{ack}_s!M'' \\ L & = & \tau L \end{array}$$

and the system

$$\mathit{Sys}'' = (S \parallel M'' \parallel R) \upharpoonright \mathcal{A}$$

Are these systems deadlock-free? Or formulated as correctness assertions, do we have *Sys : DeadLockFree, Sys′ : DeadLockFree, Sys″ : DeadLockFree* where $\mathcal{V}(\mathit{WellTerm}) = \{p \mid W(p)\}$ from figure 2.10? We leave it to the reader to verify that the transition systems for *Sys* (which has four states) and for *Sys′* and *Sys″* (which has five states) are as shown in figure 2.11, and compute $[\![\mathit{DeadLockFree}]\!]$, for *Sys*, *Sys′* and *Sys″*, to find out that indeed

$$\models \mathit{Sys} : \mathit{DeadLockFree}, \models \mathit{Sys}'' : \mathit{DeadLockFree}$$

but

$$\not\models \mathit{Sys}' : \mathit{DeadLockFree}.$$

□

2.6 Alternation Depth

Crucial for the expressive power of the modal μ-calculus is the ability to nest minimum and maximum fixed-points in a non-trivial manner, where by non-trivial we mean for instance a situation where if the outermost fixed-point

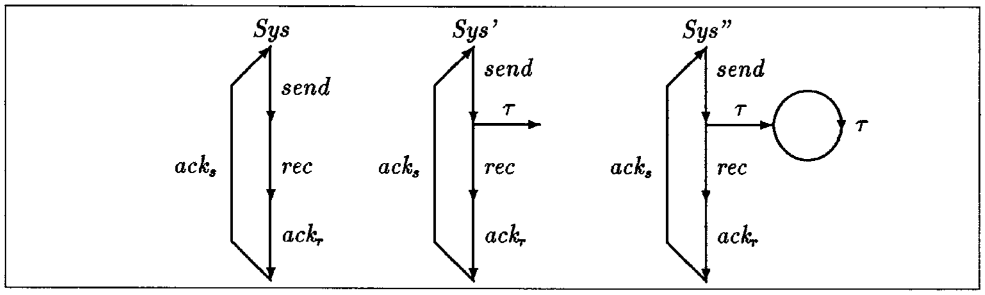


Figure 2.11: Transition systems pointed by *Sys*, *Sys*', and *Sys*".

operator is μ and some inner ν-fixed-point assertions contain free variables bound by the μ. The level of non-trivial nesting in an assertion is a somewhat tricky notion to capture and we have chosen to give the definition of this measure of *alternation depth* in an 'operational' fashion since the original motivation of defining it was to capture the complexity of a model-checking algorithm and the operational flavour fits well with our later complexity analyses.

But before applying the measure we assume that the assertions are transformed into *positive normal form* by pushing negations 'inwards' such that the assertion is built entirely from $F, T, \wedge, \vee, \langle a \rangle, [a], X, \mu$, and ν (and $Q, \neg Q, \exists\alpha, \forall\alpha, \langle\gamma\rangle, [\gamma], \texttt{where}_\mu$, and $\texttt{where}_\nu$ for the alternation depth):

Proposition 2.3 *Any assertion A in the standard (respectively extended) calculus has a logical equivalent A′ in the standard (respectively extended) calculus which is in* positive normal form.

Proof: Easy. □

We first define the notion of alternation depth for the alternation depth for the standard calculus.

2.6.1 The Standard Calculus

We call an assertion with top-most operator μ (respectively ν) *a μ-assertion* (respectively *a ν-assertion*).

Definition 2.9 For a closed assertion A, let $cps(A)$ be the set of *closed,* *proper, μ- and ν-subassertions* of A. Let $mcps(A) \subseteq cps(A)$ be the set of *maximal* such assertions, i.e. assertions in $cps(A)$ that are not subassertions of other assertions in $cps(A)$. Moreover, let the *top-level* μ-subassertions, $tl_\mu(A)$, of a closed assertions A, be defined by structure induction as follows:

$$\begin{array}{rcl} tl_\mu(F) = tl_\mu(T) & = & \emptyset \\ tl_\mu(A_0 \wedge A_1) = tl_\mu(A_0 \vee A_1) & = & tl_\mu(A_0) \cup tl_\mu(A_1) \\ tl_\mu(\langle a \rangle A) = tl_\mu([a]A) & = & tl_\mu(A) \\ tl_\mu(\mu X.A) & = & \{\mu X.A\} \\ tl_\mu(\nu X.A) & = & tl_\mu(A[F/X]). \end{array}$$

The function tl_ν is defined dually. □

We have chosen to replace, somewhat arbitrarily, F for X in the body of the maximum fixed-point to keep assertions closed.

Definition 2.10 The *alternation depth ad(A)* of a closed assertion A, is defined by induction on A as follows (we take $\max \emptyset = 0$):

if $mcps(A) = \{B_1, \ldots, B_k\} \neq \emptyset$ then
$ad(A) = \max\{ad(A[F/B_1, \ldots, F/B_k]), ad(B_1), \ldots, ad(B_k)\}$
if $mcps(A) = \emptyset$ then

$$ad(A) = \begin{cases} 0 & \text{if } A \equiv F, A \equiv T \\ \max\{ad(A_0, ad(A_1)\} & \text{if } A \equiv A_0 \vee A_1, A \equiv A_0 \wedge A_1 \\ ad(A') & \text{if } A \equiv \langle a \rangle A', A \equiv [a]A' \\ 1 + \max\{ad(B) \mid B \in tl_\nu(A')\} & \text{if } A \equiv \mu X.A' \\ 1 + \max\{ad(B) \mid B \in tl_\mu(A')\} & \text{if } A \equiv \nu X.A' \end{cases}$$

□

The purpose of the measure *ad* is to capture to what extent minimum and maximum fixed-points are nested in an *essential* way. Hence, closed assertions appearing inside μ- and ν-assertions do not increase the alternation depth, nor does sequences of fixed-points of the same kind, only when for instance a ν-assertion appears inside some μ-assertion with a free variable bound by the μ-assertion, will the alternation depth increase.

Example 2.2 Some simple examples:

$$\begin{array}{l}
ad(\langle a\rangle T \vee [b]F) = 0\\
ad(\mu X.\nu Y.X \wedge \langle a\rangle Y) = 2\\
ad(\mu X.X \wedge \nu Y.\langle a\rangle Y) = 1\\
ad(\nu X.\mu Y.\nu Z.[a]X \wedge (Y \vee Z)) = 3\\
ad(\mu Y.(\nu Z.[a]F \vee \langle b\rangle Z) \vee [c]Y) = 1\\
ad(\nu X.\mu Y.(\nu Z.[a]X \vee \langle b\rangle Z) \vee [c]Y) = 2
\end{array}$$

□

Remark 2.3 The notion of alternation depth was introduced by Emerson and Lei [35]. However, their definition contains a minor error, and our notion is slightly stronger in the sense that we attach a lower measure to some assertions.

Consider for a moment the assertion $\mu X.\nu Y.X \wedge Y$ with

$$ad(\mu X.\nu Y.X \wedge Y) = 2.$$

Although the examples of Emerson and Lei correctly suggests that their measure yields two in this particular case their definition wrongly yields one, because they require 'top-level formulae' to be *proper* subexpressions of the *bodies* of fixed-points [35, p.270] hence they consider only *proper* ν-subformulae of $\nu Y.X \wedge Y$ in case AD9 [35, p.271] of which there are none - it is a minor error: eliminating the word 'proper' corrects it.

There is a more subtle difference in the case of higher alternation depths. An example is

$$\mu X.A(X, \nu Z.B(Z, \mu U.C(U, \nu Y.D(Y, X)))) \tag{2.4}$$

where A, B, C, and D contain no fixed-points, and the notation $A(X, \nu Z. \dots)$ indicates that the only free variables of A are X and the free variables of $\nu Z. \dots$. The Emerson and Lei measure yields 4, whereas the measure introduced here yields 2. The reason is that in our measure we consider X to be a constant inside the body of the outermost minimum fixed-point, hence the alternation depth is one plus the alternation depth of

$$\nu Z.B(Z, \mu U.C(U, \nu Y.D(Y, F))). \tag{2.5}$$

Now, $\nu Y.D(Y, F)$ is a constant expression and so is $\mu U.C(U, \nu Y.D(Y, F))$ hence the alternation depth of (2.5) is one, and the alternation depth of (2.4) is two.

We believe that this was the definition originally intended by Emerson and Lei since it correctly captures the complexity of their algorithm, which is not the case with their definition – it is too pessimistic. In this last respect our definition also differs from that of Bradfield's [15, p.23].

We return to the relationship between complexities of model checking algorithms and alternation depth in chapter 5. □

2.6.2 The Extended Calculus

In order to generalize the definition of alternation depth to the extended calculus we must generalize *mcps* and tl_μ. First, the notion of a μ-assertion is extended to $\mu\mathrm{K}_{ext}$ in the obvious way by considering an assertion with top-most operator $\mathtt{where}_\mu$ to be a μ-assertion and dually for a ν-assertion. Now, $mcps(A)$ is again the set of maximal, closed, proper μ- and ν-subassertions of A. The function tl_μ (and dually tl_ν) is extended by

$$\begin{array}{rcl}
tl_\mu(Q) = tl_\mu(\neg Q) & = & \emptyset \\
tl_\mu(\psi(\gamma)) = tl_\mu(\neg\psi(\gamma)) & = & \emptyset \\
tl_\mu(\exists\alpha.A) & = & tl_\mu(A) \\
tl_\mu((A_1,\ldots,A_n)) & = & \bigcup_{1\le i\le n} tl_\mu(A_i) \\
tl_\mu(\vec{A}^{\,n}\ \mathtt{where}_\mu\ \vec{X}^{\,m} = \vec{P}^{\,m}) & = & \{\ \vec{A}^{\,n}\ \mathtt{where}_\mu\ \vec{X}^{\,m} = \vec{P}^{\,m}\ \} \\
tl_\mu(\vec{A}^{\,n}\ \mathtt{where}_\nu\ \vec{X}^{\,m} = \vec{P}^{\,m}) & = & tl_\mu(\vec{A}^{\,n}[F/X_1,\ldots,F/X_m])\cup \\
 & & tl_\mu(\vec{P}^{\,m}[F/X_1,\ldots,F/X_m])
\end{array}$$

If $mcps(A) = \emptyset$ then the defining clauses for *ad* is extended to products by:

$$ad((A_1,\ldots,A_n)) = \max\{ad(A_1),\ldots,ad(A_n)\}$$

and to `where`-clauses by:

$$ad(A) = \begin{cases}
\max\{ad(\ \vec{A}^{\,n}\ [F/X_1,\ldots,F/X_m], 1+\max\{ad(B) \mid B\in tl_\nu(\ \vec{P}^{\,m}\ )\}\} \\
\qquad\qquad\qquad\qquad \text{if } A \equiv\ \vec{A}^{\,n}\ \mathtt{where}_\mu\ \vec{X}^{\,m} = \vec{P}^{\,m} \\
\max\{ad(\ \vec{A}^{\,n}\ [F/X_1,\ldots,F/X_m], 1+\max\{ad(B) \mid B\in tl_\mu(\ \vec{P}^{\,m}\ )\}\} \\
\qquad\qquad\qquad\qquad \text{if } A \equiv\ \vec{A}^{\,n}\ \mathtt{where}_\nu\ \vec{X}^{\,m} = \vec{P}^{\,m}
\end{cases}$$

and for the constants, action predicates and quantifiers the extension is trivial.

2.7 Relating the Standard and the Extended Caculus

An interesting question about the extended calculus is under which circumstances the predicate action-logic adds to the expressive power in an essential manner, i.e. when the existential quantifier and the action predicates is more than just a compact way of representing assertions in the standard calculus. If we have an infinite number of basic actions and our predicates are powerful enough, it is not hard to see that for the pure predicate action-logic fragment – that is, even ignoring modalities and fixed-points – satisfaction can be undecidable.

Example 2.3 Assume we have given an isomorphism $\iota : \omega \cong \mathit{Act}$, i.e. we have a countable set of basic actions. Moreover, assume we have predicates *equals, less, mult* and *sum* expressing equality, multiplication and addition, i.e. they satisfy for all n, m, and l:

$$\begin{array}{rcl} equals(\iota(n) \times \iota(m)) & \Leftrightarrow & n = m \\ less(\iota(n) \times \iota(m)) & \Leftrightarrow & n < m \\ mult(\iota(n) \times (\iota(m) \times \iota(l))) & \Leftrightarrow & n = ml \\ add(\iota(n) \times (\iota(m) \times \iota(l))) & \Leftrightarrow & n = m + l \\ & . & \end{array}$$

Then the predicate action-logic given by $(\tilde{\exists}, \neg, \vee, \iota(0), \iota(1),\ equals,\ less,\ mult,\ add)$ is a version of *Number Theory*, for which the validity problem according to Gödel's incompleteness theorem is undecidable. □

To get some interesting results we first restrict attention to the situation where the only allowed predicates are

$$\alpha = a$$

for an action a. We call predicates like this action tests (see p. 27). This will also include membership tests for finite sets $\alpha \in \Lambda$ as this can be seen as an abbreviation for the finite disjunction $\bigvee_{a\in\Lambda}(\alpha = a)$. For these restricted set of predicates, we can show that the extended calculus is no more expressive than the standad calculus in logical terms, but only provides compact representations of otherwise equivalent formulas.

First, however, a simple lemma which is based on the notion of *basic label-set.*

Definition 2.11 The *basic label-set* of a labelled transition system T with label-set L is the set of basic actions appearing in actions of L. I.e. the image of the function $bls : Act_* \rightarrow \mathcal{P}(Act \cup \{*\})$ on L where bls is defined by

$$\begin{array}{rcll} bls(a \times b) & = & bls(a) \cup bls(b) & \\ bls(a) & = & \{a\} & \text{if } a \in Act \cup \{*\}. \end{array}$$

□

Lemma 2.3 *Let T be a labelled transition system with finite basic label-set L and let V be a valuation. Assume A is an assertion which only contains predicates that are action tests involving basic actions from the finite set M. Assume # is a symbol not in L or M. Then for all basic actions a,*

$$a \notin L \cup M \Rightarrow \forall \rho, \phi.\ [\![A]\!]_{T,V}\rho\ \phi[a/\alpha] = [\![A]\!]_{T,V}\rho\ \phi[\#/\alpha]$$

Proof: Simple structural induction on A. The non-trivial steps are:

$A \equiv \langle\gamma\rangle A'$. If α appears in γ, then as $a \notin L$,

$$[\![\langle\gamma\rangle A']\!]\rho\ \phi[a/\alpha] = \emptyset = [\![\langle\gamma\rangle A']\!]\rho\ \phi[\#/\alpha]$$

as also $\# \notin L$. If α does not appear in γ then the result follows from the induction hypothesis.

$A \equiv (\alpha = b)$. As $a \notin M$ then $a \neq b$ and as $\# \notin M$ we have $[\![\alpha = b]\!]\rho\ \phi[a/\alpha] = \emptyset = [\alpha = b]\!]\rho\ \phi[\#/\alpha]$.

□

Using this lemma we can, not surprisingly, remove all action quantifiers:

Lemma 2.4 *Let T be a labelled transition system with finite basic label-set L and let V be a valuation. Assume A is an action-variable closed assertion* *which only contains predicates that are action tests. Then there exists an assertion A' without action quantifiers and predicates such that*

$$\models_{T,V} A \leftrightarrow A'$$

Proof: Replace each subexpression $\exists\alpha.B$ of A by the finite disjunction

$$\bigvee_{a\in L\cup M\cup\{\#\}} B[a/\alpha]$$

where M is the set of constants appearing in matches and $\#$ is an action not in L or M. Call this new assertion C. As A is action-variable closed, C contains no action variables and all the predicates can be replaced with truth-values, yielding the assertion A'. Proving that $\models_{T,V} A \leftrightarrow C$ from which the result follows, is by straightforward structural induction, where the only non-trivial case is as follows:

$$\begin{aligned}
[\![\exists\alpha.B]\!]_{T,V}\rho\ \phi &= \bigcup_{a\in Act\cup\{*\}} [\![B]\!]_{T,V}\rho\ \phi[a/\alpha] \\
&= \bigcup_{a\in L\cup M\cup\{\#\}} [\![B]\!]_{T,V}\rho\ \phi[a/\alpha] \\
&\qquad \text{by lemma 2.3} \\
&= [\![\bigvee_{a\in L\cup M\cup\{\#\}} B[a/\alpha]]\!]_{T,V}\rho\ \phi
\end{aligned}$$

□

Although this lemma implies that we can always find quantifier-free representations of assertions, this depends on the label set of the transition system under consideration and a general translation giving a generic assertion that work in all contexts is *not* possible. In other words, we want to keep the quantifiers in order to construct general assertions, but when it comes to verify satisfaction if needed the quantifiers can be removed.

Returning to the translation we can actually do much better if just all predicates appear below modalities containing their variables. First, we formally define:

Definition 2.12 A modality $\langle\gamma\rangle A$ *binds the variable* α *in* A if α is a subexpression of γ. A predicate ψ *is guarded in* A *with respect to* α if in all occurrences of the predicate applied to an expression γ' containing α, α is bound by a modality. An assertion A *is guarded* if all predicates in A are guarded with respect to all variables. □

Lemma 2.5 *Let A be a guarded, closed assertion and assume that T is a transition system with finite basic label-set L. Then there exists an assertion A' without action quantifiers or predicates, such that*

$$\models_T A \leftrightarrow A'$$

If the predicates appearing in A are decidable, then A' is computable from A.

Proof: For assertions guarded with respect to α, lemma 2.3 can easily be strengthened to

$$a \notin L \Rightarrow \forall \rho, \phi.\ [\![A]\!]_{T,V}\rho\ \phi[a/\alpha] = [\![A]\!]_{T,V}\rho\ \phi[\#/\alpha]$$

for all basic actions a.

As in lemma 2.4 now each subexpression $\exists\alpha.B$ can be replaced by

$$\bigvee_{a \in L \cup \{\#\}} B[a/\alpha].$$

If the predicates are computable, all predicates now being applied to variable-free action expressions can be algorithmically replaced by truth-values, making A' computable from A. □

Example 2.4 Many apparently unguarded assertions can be transformed to guarded assertions. Consider for instance the unguarded assertion $\exists\alpha.(a \in \Lambda) \wedge \langle\alpha\rangle A$ abbreviating $\langle\Lambda\rangle A$ for any (possibly infinite) set of actions Λ. It is equivalent to the guarded assertion $\exists\alpha.\langle\alpha\rangle((\alpha \in \Lambda) \wedge A)$. Hence, assuming that Λ is a computable set, $\langle\Lambda\rangle A$ always has a quantifier-free version on transition systems with finite basic label-set by lemma 2.5. □

2.8 Bibliographic Notes

The generic process algebra WPA introduced here appears in work of Winskel [91]. Discussions about the applicability of process algebras as models of concurrent systems and intuitions behind the various operators can be found in various textbooks like Milner's book [59] and Hoare's book [42]. The process algebra OPA which only differs from CCS in the respect that synchronized actions are visible seems to have the benefit of making it possible to verify even closed systems (i.e. without external synchronization being allowed) using the μ-calculus because it is possible to investigate the behaviour of synchronized actions – which is impossible for CCS where any closed system is (weakly) equivalent to *nil*! Hence OPA offers a solution to the "probe problem" reported by Walker in [87].

In the modal logic community labelled transition systems are often called Kripke models and are defined slightly differently but the differences are inessential. They are also sometimes – especially when only one action is present – called "possible world semantics" indicating the more philosophical approach to modal logics as explained in Hughes and Cresswell [44].

The modal μ-calculus – here called the standard calculus – is due to Kozen [49] based upon earlier work by Park [68], Hitchcock, Scott, de Bakker, de Roever, and Pratt. Kozen, who refers to the logic as the *propositional μ-calculus*, has proven a range of results about the logic which we will return to in chapter 9. The term '*modal* μ-calculus' and the name μK is due to Stirling – who observed that the logic is really the minimal modal logic K extended with fixed-points (see for example [79]).

The idea of extending the μ-calculus with simultaneous fixed-points has been recognized by other authors as being important. This is the case in for instance the work of Larsen and Xinxin [57], Arnold and Crubille [9], and Cleaveland and Steffen [27]. However, even though Larsen and Xinxin introduces a notation for simultaneous fixed-points they do not allow for nested fixed-points to 'share across products' (i.e. the ability for several components of a simultaneous fixed-point to refer to variables of another simultaneous fixed-point without duplicating it), which is going to be a key step in making the reductions for the compositional method and the model-checking algorithms efficient. A similar construction achieving the effect of 'sharing across products' is implicitly present in Cleaveland, Dreimüller and Steffen [24].

Bekič's theorem dates back to the late sixties and appears in unpublished manuscripts by Scott and de Bakker, Park, and Bekič. The earliest publically available version the present author has been able to find is in a collection of Hans Bekič's papers [12] from 1984, where it appears in a paper dated 8. December 1969. The theorem is there called the 'bisection lemma.'

Thanks are due to Dirk Taubner for a useful discussion on the subtleties in generating finite-state systems from CCS terms and for drawing my attention to his thesis [83].

Chapter 3

Compositional Checking of Satisfaction

In the modal logic approach to verification the central problem of verifying that systems meet their specifications correspond to determining satisfaction, i.e. deciding whether a given model satisfies a given assertion. In this chapter we present a compositional method for deciding satisfaction, centered around the standard calculus but also with generalizations to the extended calculus.

The method is *compositional in the structure of the processes* and works purely on the syntax of processes. It consists of applying a sequence of *reductions*, each of which only take into account the top-level operator of the process (except the reduction for product which inspects one of the components). Such a reduction transforms a satisfaction problem for a composite process into equivalent satisfaction problems for the immediate subcomponents - without inspecting the internal structure of these.

Using process variables, systems with undefined subcomponents can be defined, and given an overall requirement to the system, *necessary and sufficient conditions* on these subcomponents can be found. Hence the process variables make it possible to specify and reason about what are often referred to as *contexts, environments,* and *partial implementations.*

Compositionality is important for at least the following reasons. Firstly, it makes the verification *modular,* so that when changing a part of a system only the verification concerning that particular part must be redone. Secondly, when designing a system or *synthesizing* a process the compositionality makes it possible to have undefined parts of a process and still be able to reason about it. For instance, it might be possible to reveal inconsistencies in the specification or prove that with the choices already taken in the design no component supplied for the missing parts will ever be able to make the overall system satisfy the original specification. Thirdly, it makes it possible to *decompose* the verification task into potentially simpler tasks. Finally, it can make possible the *reuse* of verified components; their previous verification can be used to show that they meet the requirements on the components of a larger system.

We first describe the reductions for the standard calculus then look at some examples and finally discuss how to generalize the reductions to the extended calculus.

3.1 Introduction

In the process of deriving the reductions we will temporarily have to internalize the correctness assertions $(t : A)$ into the logic which is then an extension of the standard calculus which we will refer to as $\mu K_{(:)}$. It has the following syntax:

$$A ::= F \mid \neg A \mid A_0 \vee A_1 \mid \langle a \rangle A \mid X \mid \mu X.A \mid (t : A)$$

We still require the correctness assertions $(t : A)$ to be well-formed, i.e. t must be mono-typed and $\mathit{type}(A) \subseteq \mathit{type}(t)$. We will say that an assertion A is *pure* if it does not contain any correctness assertions, hence a pure assertion in $\mu K_{(:)}$ is really an assertion in the standard calculus. A correctness assertion $(t : A)$ is pure if A is pure, and we shall later show that all correctness assertions can be made pure.

A correctness assertion $(t : A)$ is to denote true or false depending on whether t satisfies A or not. This suggests the following simple extension of the semantics using superscript (:) to indicate that assertions are from $\mu K_{(:)}$:

$$[\![(t : A)]\!]_T^{(:)}\rho = \begin{cases} S_T & \text{if } t \in [\![A]\!]_T^{(:)}\rho \\ \emptyset & \text{otherwise} \end{cases}$$

Instead of using $t \in [\![A]\!]_T^{(:)}\rho$ we could just as well, according to the locality lemma, take $t \in [\![A]\!]_T^{(:)}\phi$. However, there is a small problem with the environment ρ. On the left-hand side ρ is a map $\mathit{AssnVar} \to \mathcal{P}(S_T)$ but on the right-hand side we need an environment $\rho : AssnVar \rightarrow \mathcal{P}(Proc_{WPA})$ or $\rho : AssnVar \rightarrow \mathcal{P}(R_t)$ depending on whether we use the global or local view. If A is closed we could take any environment ρ' of the right kind and get a well-defined semantics. Unfortunately, during the reductions assertions containing correctness assertions over various different terms with free variables will appear, and therefore the environment used in the semantics must map these variables to subsets of states of the corresponding transition systems. Sticking to a global semantics using the universal transition system $\mathcal{T}$ and only consider environments mapping all variables to subsets of $Proc_{\mathrm{WPA}}$ is not a useful solution because the reductions will require a local view.

The solution we take is to give the semantics of assertions in $\mu K_{(:)}$ with respect to environments ρ which respect the 'types' of the free assertion variables. We annotate variables with types (i.e. transition systems) according to the following rules: Let T be a transition system. Then A^T is the type-annotated assertion constructed from A by annotating all variables X not within correctness assertions as X^T. If X appears inside a correctness assertion we annotate X with the process term of the nearest enclosing correctness assertion. For example $(X \vee (t : Y \vee (t' : Z)))^T = X^T \vee (t : Y^t \vee (t' : Z^{t'}))$.

An environment $\rho : AssnVar \rightarrow \mathcal{P}(Proc_{\mathrm{WPA}})$ *respects the types* of the assertion A^T if for all free variables $X^{T'}$,

$$\rho(X^{T'}) \subseteq S_{T'} \subseteq Proc_{\mathrm{WPA}} \tag{3.1}$$

This is trivially fulfilled by a closed assertion and the open assertions appearing during[-5mm] the construction of the reductions will turn out to satisfy (3.1).

Given an environment ρ satisfying (3.1) for all free variables of A^T we now define $[\![A^T]\!]^{(:)}_T \rho$ by structural induction on A.

$$\begin{array}{rcl}
[\![F]\!]^{(:)}_T \rho & = & \emptyset \\
[\![\neg A]\!]^{(:)}_T \rho & = & S_T \setminus [\![A]\!]^{(:)}_T \\
[\![A_0 \vee A_1]\!]^{(:)}_T \rho & = & [\![A_0]\!]^{(:)}_T \rho \cup [\![A_1]\!]^{(:)}_T \rho \\
[\![\langle a \rangle A]\!]^{(:)}_T \rho & = & \{s \in S \mid \exists s' \in S.\ s \xrightarrow{a} s' \ \&\ s' \in [\![A]\!]^{(:)}_T \rho\} \\
[\![X^T]\!]^{(:)}_T \rho & = & \rho(X^T) \\
[\![\mu X^T.A]\!]^{(:)}_T \rho & = & \mu\psi \text{ where } \psi : U \subseteq S_T \mapsto [\![A]\!]^{(:)}_T \rho[U/X^T] \\
[\![(t : A)]\!]^{(:)}_T \rho & = & \begin{cases} S_T & \text{if } t \in [\![A]\!]^{(:)}_T \rho \\ \emptyset & \text{otherwise} \end{cases}
\end{array}$$

Notice, that if A is pure then if ρ maps all free variables to $\mathcal{P}(S_T)$, then ρ respects the types of A^T and for any environment $AssnVar \to \mathcal{P}(S_T)$ we have $[\![A^T]\!]_T^{(:)}\rho = [\![A]\!]_T\rho$, hence for pure assertions we could just use the standard semantics.

The technical difficulties we have just experienced will not be fully rewarded until the reduction for product, which is the only reduction where correctness assertions will have to be nested.

The correctness assertions $(t : A)$ will also be atoms in a propositional logic $\mathcal{L}$ which will be used to express the reductions.[1] A grammar for the logic $\mathcal{L}$ is:

$$L ::= T \mid \neg L \mid L_0 \vee L_1 \mid (t : A)$$

where A is an assertion in $\mu K_{(:)}$. Satisfaction in $\mathcal{L}$ of a formula L is defined, relative to an environment ρ which respects the type of the correctness assertions of L, as follows:

$$\begin{array}{ll}
\models_\rho T & \text{always} \\
\models_\rho \neg L & \text{iff not} \models_\rho L \\
\models_\rho L_0 \vee L_1 & \text{iff} \models_\rho L_0 \mathit{\ or\ } L_1 \\
\models_\rho t : A & \text{iff } t \in [\![A^t]\!]_t^{(:)}\rho
\end{array}$$

Furthermore we define the derived predicate $\models$ as:

$$\models L \text{ iff for all } \rho \models_\rho L.$$

Taking $\bullet$ to be the trivial transition system with one state (denoted $\bullet$) and no transitions, we observe that the set of assertions built from correctness assertions, negations, and conjunctions when interpreted over $\bullet$ is essentially a copy of the logic L, i.e. for such an assertion A we have $[\![A]\!]_\bullet\rho = \{\bullet\}$ if and only if $\models_\rho A$ where A is interpreted as a formula in the propositional logic.

In $\mathcal{L}$ we are able to express complex relationships between properties of different processes. For example

$$(p + q : \langle a\rangle A) \leftrightarrow (p : \langle a\rangle A) \vee (q : \langle a\rangle A),$$

[1]This propositional logic is of course a sublogic of the above defined μ-calculus, but it will be beneficial to keep the two levels separate. The correctness assertions appearing in the assertions only have a temporary existence and will ultimately be removed whereas the correctness assertions in the propositional logic are crucial for expressing the reductions.

expresses a very simple example of a reduction. It states that the process $p+q$ can perform an a-action and get into a state that satisfies A if and only if p or q can do an a-action and get into a state that satisfies A. It is a reduction because the formula is valid for all p's and q's, and the validity of $(p+q : \langle a \rangle A)$ is reduced to validity of correctness assertions over the subterms p and q. Although this reduction is almost trivial, in general, it might be quite difficult to get reductions. Consider for example the problem of choosing a B such that

$$(rec\ P.t : \mu X.A) \leftrightarrow (t : B)$$

holds. The aim of this chapter is to describe a method for generating such a B and analogous assertions for all the other operators.

In constructing these reductions we will be involved with the *rooting* of transition systems which is defined as follows.

Definition 3.1 Given a pointed transition system $T = (S, i, L, \rightarrow)$ the *rooting of* T is a pointed transition system $\underline{T} = (S \cup \{\underline{i}\}, \underline{i}, L, \rightarrow')$ where $\underline{i}$ is a new state assumed not to be in S, and the transition relation $\rightarrow' \subseteq (S \cup \{\underline{i}\}) \times L \times (S \cup \{\underline{i}\})$ is defined by:

$$\rightarrow' = \rightarrow \cup \{(\underline{i}, a, q) \mid i \xrightarrow{a} q\}.$$

□

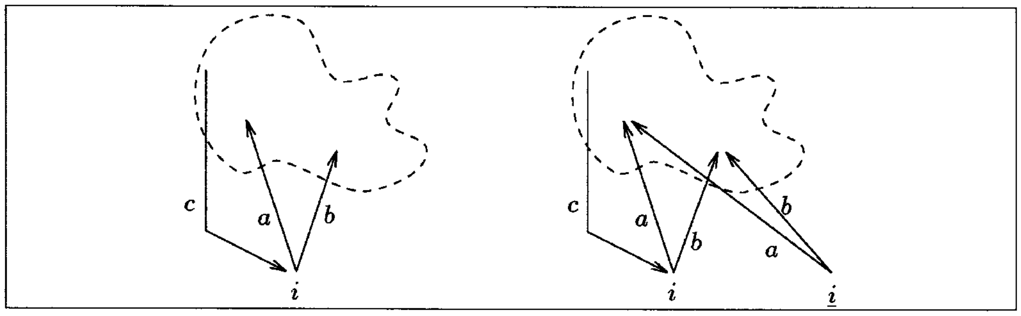


Figure 3.1: The rooting of a transition system. Notice, that no transitions enter $\underline{i}$.

Pictorially the rooting of a pointed transition system is constructed by adjoining a new initial state with the same out-going transitions as the old initial state (see figure 3.1). The rooting of a transition system T is "just as good as T" with respect to satisfaction in our logic. A claim made precise by the rooting lemma below.

Lemma 3.1 (Rooting lemma.)
Given a pointed transition system, $T = (S_T, i, L, \rightarrow_T)$, where S_T is countable and with the rooting $\underline{T}$. Let $r : \mathcal{P}(S_T) \rightarrow \mathcal{P}(S_T \cup \{\underline{i}\})$ be the map on properties that take the initial state of T to the two copies of it in $\underline{T}$ and take all other states to their obvious counterparts. Assume A is a pure assertion. Let $\rho : AssnVar \rightarrow \mathcal{P}(S_T)$ be an environment of assertions which since A is pure, respects the types of A. Then

$$r([\![A]\!]_T\ \rho) = [\![A]\!]_{\underline{T}}(r \circ \rho).$$

Proof: See appendix A.1. $\square$

The connection given by the rooting lemma between pointed transition systems T and their rootings $\underline{T}$ could be summarized as: The set of states satisfying an assertion will be the same in both interpretations up to application of the map r. In particular the initial state of T will satisfy A if and only if the initial state of $\underline{T}$ satisfies A; an observation central to our development of reductions in section 3.2.

3.2 Reductions

Our method for compositional checking of satisfaction is based on the notion of a *reduction*, which we explain in terms of the prefix operator.

Given a pure and closed assertion A and a prefix at we would like to find a propositional expression B over atoms $(t : B_i)$ such that the following holds:

$$\models (at : A) \leftrightarrow B$$

Having found such a B the validity of $(at : A)$ has been *reduced* to validity of a propositional expression containing only atoms on the subterm t. In other words: B is a *necessary* and *sufficient* condition on the subterm t ensuring that *at* satisfies A. By the word *reduction* we will henceforth understand *a description of how to find B given A and at.*

It is not obvious that such a B exists. Although we can easily express the set of processes that will make the correctness assertion valid as

$$\{t \in \mathcal{S} \mid \models at : A\},$$

it is not necessarily the case that this set can be expressed *within the logic* $\mathcal{L}$ as an assertion B over atoms $(t : B_i)$ such that

$$\{t \in \mathcal{S} \mid \models B\} = \{t \in \mathcal{S} \mid \models at : A\}.$$

In general, the ability to do so, will depend on the expressive power of the logic, and the kind of operation for which we are trying to find a reduction. We will show that for our modal logic and all operators of our process algebra, such a B does indeed exist, and furthermore we give for each operator an algorithm that computes one particular choice of B.

In providing this B the most difficult part concerns – not surprisingly – the fixed-points and the single most important property of fixed-points around which all the reductions are centered, is expressed by the reduction lemma. Recall that a map on a complete join semilattice is ω-continuous if it preserves joins of all increasing ω-chains.

Lemma 3.2 (Reduction lemma)
Suppose D and E are powersets over countable sets, and $in : D \to E$ an ω-continuous function with $in\,(\bot_D) = \bot_E$. Suppose $\psi : E \to E$ and $\theta : D \to D$ are both monotonic and have the property

$$\psi \circ in = in \circ \theta$$

We can then conclude that

$$\mu\psi = in(\mu\theta)$$

Proof: See appendix A.2. □

To understand the role of the reduction lemma, take E to be the lattice of properties of a compound process and D to be a lattice built from properties of immediate subprocesses. The lemma allows us to express a fixed-point property of the original compound process in terms of fixed-points of functions over properties of its immediate subcomponents via the transformation *in*.

For example, the properties of a process at can be identified with certain subsets of the states $R_{\underline{at}}$ in the rooting of the transition system pointed by at, and the properties of t with subsets of the states R_t of the transition system pointed by t. Now we take the transformation to be

$$in : \mathcal{P}(R_t) \times \mathcal{P}(\{\bullet\}) \to \mathcal{P}(R_{\underline{at}})$$

where

$$in(V_0, V_1) = V_0 \cup \{\underline{at} \mid \bullet \in V_1\}$$

The role of the extra product component is to record whether or not the property holds at the initial state $\underline{at}$ of $R_{\underline{at}}$. (The rooting is required to ensure that the initial state at is not confused with later occurrences .)[2]

An assertion with a free variable occurring positively essentially denotes a monotonic function $\psi : \mathcal{P}(R_{\underline{at}}) \to \mathcal{P}(R_{\underline{at}})$. The definition of the reduction is given by structural induction on assertions ensuring that assertions denoting such functions ψ, and their reductions denoting monotonic functions $\theta : \mathcal{P}(R_t) \times \mathcal{P}(\{\bullet\}) \to \mathcal{P}(R_t) \times \mathcal{P}(\{\bullet\})$ are related by *in* in the manner demanded by the reduction lemma. The lemma then allows the reduction to proceed for fixed-points. As this case of prefixing makes clear, reductions of fixed-points can be simultaneous fixed-points. Bekič's theorem (theorem 2.2) could be used to replace the simultaneous fixed-points by fixed-points in the individual components to get assertions in $\mu K_{(:)}$ but we keep the simultaneous fixed-points in order to keep assertions small. We later discuss how the reductions are extended to apply on assertions which themselves contain simultaneous fixed-points.

In the course of this definition by structural induction we will be faced with the problem of giving a reduction for assertion variables. One solution to this problem can be found by introducing a syntactic counterpart of *in* called *IN* and define a *change of variables* σ to be a map taking all variables X of type $\underline{a.t}$ to $IN(X_0, X_1)$ where the types of X_0 and X_1 are t respectively

[2]Because of the isomorphism $\mathcal{P}(A_0) \times \cdots \times \mathcal{P}(A_n) \times \cdots \cong \mathcal{P}(A_0 + \cdots + A_n + \cdots)$ we can still meet the conditions of the reduction lemma when D is a countable product of powersets of countable sets.

•. An application of such a substitution σ to an assertion A has to satisfy certain technical requirements: It should be *fresh* i.e. for an assertion A when

(i) for all variables X the free variables in $\sigma(X)$ are disjoint from those in A, and
(ii) for distinct variables X and X', the free variables in $\sigma(X)$ and $\sigma(X')$ are disjoint.

We will use the notation $A[\sigma]$ to denote the assertion resulting from performing the substitution σ and we use $\sigma\backslash X$ to denote the substitution which is like σ except that X is left unchanged. The meaning of *IN* can be summarized by the equation

$$[\![IN(X_0, X_1)]\!]_{\underline{at}}\ \rho = in(\rho(X_0), \rho(X_1)),$$

justifying that *IN* is the "syntactic counterpart of *in*." It is emphasized that while the syntactic counterparts *IN* of the transformations play the important part in expressing relationships between variables and in showing the correctness of reductions, they do *not* appear in the reductions themselves.

Reductions for all operators can be established along the lines sketched. Each operator involves a judicious choice of *in*, which *IN* is to denote. In the following sections we present this choice and the accompanying reductions.

3.2.1 Prefix

The reduction for prefix is defined inductively on the structure of assertions and shown in figure 3.2. Note that $\mathrm{red}^0(at : A; \sigma)$ just renames the variables of A from X to X_0 when $\sigma(X) = IN(X_0, X_1)$. The transformation in was explained in the previous section.[3]

The reduction is constructed in such a way that the two components are related to A through *in* by

$$[\![A^{\underline{at}}[\sigma]]\!]^{(:)}_{\underline{at}}\rho = in([\![\mathrm{red}^0(at : A; \sigma)^t]\!]^{(:)}_t\ \rho, [\![\mathrm{red}^1(at : A; \sigma)^\bullet]\!]^{(:)}_\bullet\rho), \qquad (3.2)$$

[3]For this and the following reductions we have that $\mathrm{red}(at : \langle *\rangle A; \sigma) = \mathrm{red}(at : A; \sigma)$ and henceforth we omit this trivial case from the presentation.

$$
\begin{array}{lcl}
\mathbf{red}^0(at : X; \sigma) & = & X_0 \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^0(at : \mu X.A; \sigma) & = & \mu X_0.\mathbf{red}^0(at : A; \sigma) \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^0(at : \langle b\rangle A; \sigma) & = & \langle b\rangle \mathbf{red}^0(at : A; \sigma) \\
\mathbf{red}^0(at : \neg A; \sigma) & = & \neg \mathbf{red}^0(at : A; \sigma) \\
\mathbf{red}^0(at : A \vee B; \sigma) & = & \mathbf{red}^0(at : A; \sigma) \vee \mathbf{red}^0(at : B; \sigma) \\
 & & \\
\mathbf{red}^1(at : X; \sigma) & = & X_1 \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^1(at : \mu X.A; \sigma) & = & \mathbf{red}^1(at : A; \sigma)[\mathbf{red}^0(at : \mu X.A; \sigma)/X_0][F/X_1] \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^1(at : \langle b\rangle A; \sigma) & = & \begin{cases} (t : \mathbf{red}^0(at : A; \sigma)) & \text{if } b = a \\ F & \text{if } b \neq a \end{cases} \\
\mathbf{red}^1(at : \neg A; \sigma) & = & \neg \mathbf{red}^1(at : A; \sigma) \\
\mathbf{red}^1(at : A \vee B; \sigma) & = & \mathbf{red}^1(at : A; \sigma) \vee \mathbf{red}^1(at : B; \sigma)
\end{array}
$$

Figure 3.2: Reduction for prefix defined inductively on the structure of assertions.

where σ is a change of variables for A and ρ is an environment respecting the types of A. From now on we leave out the type annotations - they are easily reconstructed.

From the rooting lemma we know that

$$at \in [\![A]\!]_{at}\ \rho \text{ iff } \underline{at} \in [\![A]\!]_{\underline{at}}(r \circ \rho)$$

where r is the map from R_{at} to $R_{\underline{at}}$, and from the definition of *in* and (3.2) we get

$$at \in [\![A]\!]_{at}\ \rho \text{ iff } \bullet \in [\![\mathrm{red}^1(at : A; \sigma]\!]_{\bullet}\ \rho.$$

As $\mathrm{red}^1(at : A; \sigma)$ consists of correctness assertions, negations, and conjunctions only, we can consider it to be a formula in our propositional logic, yielding our reduction

$$\models (at : A) \leftrightarrow \mathrm{red}^1(at : A; \sigma).$$

Theorem 3.1 (Reduction for prefix) *Given a closed, pure assertion A, a change of variables σ which is fresh for A, and an arbitray process term t, then*

$$\models (at : A) \leftrightarrow \text{red}^1(at : A; \sigma).$$

Proof: See appendix A.3. □

3.2.2 Nil

The reduction for nil is defined inductively on the structure of assertions and shown in figure 3.3. The transformation $in : \mathcal{P}(\{\bullet\}) \to \mathcal{P}(\{nil\})$ is just the direct image of the obvious isomorphism between $\{\bullet\}$ and $\{nil\}$. Note that the reduction for nil is quite trivial and just gives true (T) or false (F).

$$\begin{array}{lcl}
\mathbf{red}(nil : X; \sigma) & = & Y \textbf{ where } \sigma(X) = IN(Y) \\
\mathbf{red}(nil : \mu X.A; \sigma) & = & \mathbf{red}(nil : A; \sigma)[F/Y] \textbf{ where } \sigma(X) = IN(Y) \\
\mathbf{red}(nil : \langle a \rangle A; \sigma) & = & F \\
\mathbf{red}(nil : \neg A; \sigma) & = & \neg\mathbf{red}(nil : A; \sigma) \\
\mathbf{red}(nil : A \vee B; \sigma) & = & \mathbf{red}(nil : A; \sigma) \vee \mathbf{red}(nil : B; \sigma)
\end{array}$$

Figure 3.3: Reduction for nil.

Theorem 3.2 (Reduction for nil) *Given a closed, pure assertion A and a change of variables σ which is fresh for A, then $\models (nil : A) \leftrightarrow \text{red}(nil : A; \sigma)$.*

Proof: Like the previous, see appendix A.3. □

3.2.3 Sum

The reduction for sum is presented in figure 3.4.

To understand the transformation first note that we have a map $j : R_{t_0} + R_{t_1} \to R_{\underline{t_0+t_1}}$ taking the initial states of t_0 and t_1 to the state $t_0 + t_1$ in $R_{\underline{t_0+t_1}}$ and taking all other states to their obvious counterparts.

We take the transformation to be

$$in : \mathcal{P}(R_{t_0} + R_{t_1}) \times \mathcal{P}(\{\bullet\}) \to \mathcal{P}(R_{\underline{t_0+t_1}})$$

where $in(V_0, V_1) = \{j(s) \mid s \in V_0\} \cup \{\underline{t_0 + t_1} \mid \bullet \in V_1\}$.

$$
\begin{array}{lcl}
\mathbf{red}^0(t_0+t_1 : X;\sigma) & = & X_0 \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^0(t_0+t_1 : \mu X.A;\sigma) & = & \mu X_0.\mathbf{red}^0(t_0+t_1 : A;\sigma) \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^0(t_0+t_1 : \langle a\rangle A;\sigma) & = & \langle a\rangle\mathbf{red}^0(t_0+t_1 : A;\sigma) \\
\mathbf{red}^0(t_0+t_1 : \neg A;\sigma) & = & \neg\mathbf{red}^0(t_0+t_1 : A;\sigma) \\
\mathbf{red}^0(t_0+t_1 : A\vee B;\sigma) & = & \mathbf{red}^0(t_0+t_1 : A;\sigma)\vee\mathbf{red}^0(t_0+t_1 : B;\sigma) \\
 & & \\
\mathbf{red}^1(t_0+t_1 : X;\sigma) & = & X_1 \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^1(t_0+t_1 : \mu X.A;\sigma) & = & \mathbf{red}^1(t_0+t_1 : A;\sigma)[\mathbf{red}^0(t_0+t_1 : \mu X.A;\sigma)/X_0][F/X_1] \\
 & & \text{where } \sigma(X) = \mathit{IN}(X_0, X_1) \\
\mathbf{red}^1(t_0+t_1 : \langle a\rangle A;\sigma) & = & (t_0 : \langle a\rangle A^0)\vee(t_1 : \langle a\rangle A^0) \\
 & & \text{where } A^0 = \mathbf{red}^0(t_0+t_1 : A;\sigma) \\
\mathbf{red}^1(t_0+t_1 : \neg A;\sigma) & = & \neg\mathbf{red}^1(t_0+t_1 : A;\sigma) \\
\mathbf{red}^1(t_0+t_1 : A\vee B;\sigma) & = & \mathbf{red}^1(t_0+t_1 : A;\sigma)\vee\mathbf{red}^1(t_0+t_1 : B;\sigma)
\end{array}
$$

Figure 3.4: Reduction for sum.

Theorem 3.3 (Reduction for sum) *Given a closed, pure assertion A, a change of variables σ which is fresh for A, and arbitrary process terms t_0 and t_1, then*

$$\models (t_0 + t_1 : A) \leftrightarrow \mathrm{red}^1(t_0 + t_1 : A; \sigma).$$

Proof: Very similar to the proof of correctness for the reduction of prefix, see appendix A.3. □

3.2.4 Relabelling

For relabelling we take the transformation to be $in : \mathcal{P}(R_t) \to \mathcal{P}(R_{t\{\Xi\}})$ where $in(V) = \{p\{\Xi\} \mid p \in V\}$.

Theorem 3.4 (Reduction for relabelling) *Let $\Xi : \eta \to \eta'$ be a relabelling map which has finite preimages, i.e. for all $a \in Act, \Xi^{-1}(a)$ is finite. Assume A is closed and pure, σ a change of variables which is fresh for A, and t any process term, then*

$$\models (t\{\Xi\} : A) \leftrightarrow (t : \mathrm{red}_{\{\Xi\}}(A; \sigma)).$$

Proof: Like the proof for restriction, see appendix A.4. □

$$\begin{array}{lcl}
\mathbf{red}_{\{\Xi\}}(X;\sigma) & = & Y \ \ \mathbf{where}\ \sigma(X) = \mathit{IN}(Y) \\
\mathbf{red}_{\{\Xi\}}(\mu X.A;\sigma) & = & \mu Y.\mathbf{red}_{\{\Xi\}}(A;\sigma)\ \ \mathbf{where}\ \sigma(X) = \mathit{IN}(Y) \\
\mathbf{red}_{\{\Xi\}}(\langle a\rangle A;\sigma) & = & \bigvee_{b\in\Xi^{-1}(a)} \langle b\rangle \mathbf{red}_{\{\Xi\}}(A;\sigma) \\
\mathbf{red}_{\{\Xi\}}(\neg A;\sigma) & = & \neg\mathbf{red}_{\{\Xi\}}(A;\sigma) \\
\mathbf{red}_{\{\Xi\}}(A\vee B;\sigma) & = & \mathbf{red}_{\{\Xi\}}(A;\sigma)\vee\mathbf{red}_{\{\Xi\}}(B;\sigma)
\end{array}$$

Figure 3.5: Reduction for relabelling.

3.2.5 Restriction

For restriction we take the transformation to be $in : \mathcal{P}(R_t) \to \mathcal{P}(R_{t\restriction\Lambda})$ where $in(V) = \{p \restriction \Lambda \mid p \in V\} \cap R_{t\restriction\Lambda}$.

$$\begin{array}{lcl}
\mathbf{red}_{\restriction\Lambda}(X;\sigma) & = & Y \ \ \mathbf{where}\ \sigma(X) = \mathit{IN}(Y) \\
\mathbf{red}_{\restriction\Lambda}(\mu X.A;\sigma) & = & \mu Y.\mathbf{red}_{\restriction\Lambda}(A;\sigma)\ \ \mathbf{where}\ \sigma(X) = \mathit{IN}(Y) \\
\mathbf{red}_{\restriction\Lambda}(\langle a\rangle A;\sigma) & = & \begin{cases} \langle a\rangle\mathbf{red}_{\restriction\Lambda}(A;\sigma) & \mathbf{if}\ a\in\Lambda \\ F & \mathbf{if}\ a\notin\Lambda \end{cases} \\
\mathbf{red}_{\restriction\Lambda}(\neg A;\sigma) & = & \neg\mathbf{red}_{\restriction\Lambda}(A;\sigma) \\
\mathbf{red}_{\restriction\Lambda}(A\vee B;\sigma) & = & \mathbf{red}_{\restriction\Lambda}(A;\sigma)\vee\mathbf{red}_{\restriction\Lambda}(B;\sigma)
\end{array}$$

Figure 3.6: Reduction for restriction.

Theorem 3.5 (Reduction for restriction) *Assume A closed and pure, a change of variables σ which is fresh for A, and an arbitrary process term t, then*

$$\models (t \restriction \Lambda : A) \leftrightarrow (t : \mathrm{red}_{\restriction\Lambda}(A;\sigma)).$$

Proof: See appendix A.4. □

3.2.6 Recursion

In order to define the reduction for recursion, we will need to extend our assertion language with an assertion $\widehat{P}$ to identify recursion points. The semantics of $\widehat{P}$ is simply:[4]

[4]The general semantics should be $[\![\widehat{P}]\!]_T\ \rho = \{P, \underline{P}\} \cap R_T$, but due to our requirement of guardedness, we will never be involved with rooting a state identifier, so the stated

$$[\![\widehat{P}]\!]_T\ \rho = \{P\} \cap R_T$$

We can consider $\widehat{P}$ to be a constant with the universal valuation $\mathcal{V}(\widehat{P}) = \{P\}$.

It can be verified that the locality and the rooting lemma still hold. All the reductions mentioned in the previous sections should be extended to take care of the assertions $\widehat{P}$ and this is easily done – they should all give F. Furthermore, we add a reduction for P, which is like the one for *nil*, except that it gives T on $\widehat{P}$.

For the first time we will need to put in extra correctness assertions in our reductions, which furthermore might contain free assertion variables. These correctness assertions can however be closed by a *closure lemma* and then 'pulled out' by a *purifying lemma* yielding an expression in L which only has correctness assertion containing pure assertions, hence being applicable for further reductions.

Lemma 3.3 (Closure lemma, Winskel [94]) *Let $C[Y]$ be an assertion in which Y occurs free and positively. Let B be any assertion in $\mu K_{(:)}$. Assume that for all environments ρ respecting the types of an assertion B with respect to T, $[\![B]\!]_T\rho = S_T$ or $[\![B]\!]_T\rho = \emptyset$, then*

$$[\![\mu X.C[B]]\!]^{(:)}\rho = [\![\mu X.C[B[\mu X.C[F]/X]]]\!]^{(:)}\rho.$$

Notice, that if the *only* occurrence of X is within B, then we can remove the fixed-point altogether:

$$[\![\mu X.C[B]]\!]^{(:)}\rho = [\![C[B[C[F]/X]]]\!]^{(:)}\rho.$$

Lemma 3.4 (Purifying lemma)
Let A be an assertion with all correctness assertions closed and let t be a process term. Then there exists an expression B over unnested correctness assertions such that

$$\models (t : A) \leftrightarrow B.$$

The proof also gives a simple algorithm for computing such a B:

semantics is sufficient.

Proof: For an assertion A let $s(A)$ denote the number of correctness assertions appearing in A. We show by mathematical induction that for all n, the theorem holds for all assertions A with $s(A) = n$.

For $n = 0$: Trivial, take $B \equiv (t : A)$. For $n > 0$: Assume A is a closed assertion in which all correctness assertions are closed. Pick a correctness assertion (e.g. the leftmost) in $A, (t' : A')$ say, writing $A[(t' : A')]$ to identify the occurrence. Define

$$C \equiv ((t' : A') \wedge A[T]) \vee (\neg(t' : A') \wedge A[F]),$$

where $A[T]$ denotes the resulting of replacing T for $(t' : A')$ in A, and similar for $A[F]$. Obviously $\models (t : A) \leftrightarrow (t : C)$. Now, as $s(A[T]) < n$ and $s(A[F]) < n$ we have by the induction hypothesis that there exists B_0 and B_1 with no nested correctness assertions, such that

$$\models (t : A[T]) \leftrightarrow B_0 \qquad \text{and} \qquad \models (t : A[F]) \leftrightarrow B_1,$$

and as

$$\models (t : C) \leftrightarrow ((t' : A') \wedge (t : A[T])) \vee (\neg(t' : A') \wedge (t : A[F])),$$

we get

$$\models (t : C) \leftrightarrow ((t' : A') \wedge B_0) \vee (\neg(t' : A') \wedge B_1),$$

which proves the result by taking $B \equiv ((t' : A') \wedge B_0) \vee (\neg(t' : A') \wedge B_1)$. □

Take $j : R_{\underline{t}} \to R_{rec\ P.t}$ to be the map that takes $\underline{t}$ to $rec\ P.t$ and all other states s to $s[rec\ P.t/P]$. The transformation for recursion $in : \mathcal{P}(R_{\underline{t}}) \to \mathcal{P}(R_{rec\ P.t})$ is defined to be the direct image of j.

Theorem 3.6 (Reduction for recursion) *Given a closed, pure assertion A, a change of variables σ which is fresh for A, and a regular process term t in which P is strongly guarded then*

$$\models (rec\ P.t : A) \leftrightarrow (t : \mathrm{red}(rec\ P.t : A; \sigma)).$$

Proof: See appendix A.5. □

$$\begin{aligned}
\mathrm{red}(\mathit{rec}\ P.t : X;\sigma) &= Y \ \text{ where } \sigma(X) = \mathit{IN}(Y)\\
\mathrm{red}(\mathit{rec}\ P.t : \mu X.A;\sigma) &= \mu Y.\mathrm{red}(\mathit{rec}\ P.t : A;\sigma) \ \text{ where } \sigma(X) = \mathit{IN}(Y)\\
\mathrm{red}(\mathit{rec}\ P.t : \langle a\rangle A;\sigma) &= \langle a\rangle A' \vee (\hat{P} \wedge (t : \langle a\rangle A'))\\
& \text{ where } A' = \mathrm{red}(\mathit{rec}\ P.t : A;\sigma)\\
\mathrm{red}(\mathit{rec}\ P.t : \neg A;\sigma) &= \neg\mathrm{red}(\mathit{rec}\ P.t : A;\sigma)\\
\mathrm{red}(\mathit{rec}\ P.t : A \vee B;\sigma) &= \mathrm{red}(\mathit{rec}\ P.t : A;\sigma) \vee \mathrm{red}(\mathit{rec}\ P.t : B;\sigma)
\end{aligned}$$

Figure 3.7: Reduction for recursion.

3.3 Reduction for Product

A reduction for a product $q \times p$ should be an assertion B over atoms $(q : B_i)$ and $(p : C_j)$ such that

$$\models q \times p : A \quad \text{iff} \quad \models B.$$

Unfortunately, if we insist on finding such a B without inspecting either p or q, we can get a very complex expression which, in the case of fixed-points will even become infinite unless assumptions on the possible sizes of p and q are made (cf. the remarks at the end of Winskel [94]). In Winskel [94] it is shown how a reasonable sized B can be found, when the assertion language is restricted rather severely, excluding disjunctions, negations, minimal fixed-points, and general box formulas, but still having maximal fixed-points, diamond formulas, a strong version of box formulas, and conjunctions.

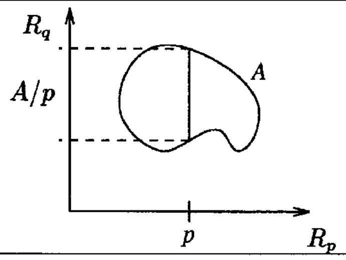


Figure 3.8: Graphical view of the reduction for product.

Here we present another approach. We give a reduction when p is a process term without restrictions and relabellings, i.e. we find a B (depending on p) s.t.

$$
\begin{array}{lcl}
\neg A/p & = & \neg(A/p) \\
A_0 \vee A_1/p & = & (A_0/p) \vee (A_1/p) \\
X/p & = & X_p \\
\mu X.A/p & = & (X_p \ \mathtt{where}_\mu \ (X_{p_1},\dots,X_{p_n}) = (A/p_1,\dots,A/p_n)) \\
 & & \text{where } R_p = \{p_1,\dots,p_n\} \\
A/q \times r & = & (A/r)/q \\
 & & \text{with the actions in the modalities of } A \text{ re-associated} \\
 & & \text{changing the type of } A \text{ from } \eta_1{\times}(\eta_2{\times}\eta_3) \text{ to } (\eta_1{\times}\eta_2){\times}\eta_3 \\
\langle a{\times}b\rangle A/nil & = & \begin{cases} \langle a\rangle(A/nil) & \text{if } b = * \\ F & \text{if } b \neq * \end{cases} \\
\langle a{\times}b\rangle A/cq & = & \begin{cases} \langle a\rangle(A/cq) & \text{if } b = * \\ \langle a\rangle(A/q) & \text{if } b = c \\ F & \text{otherwise} \end{cases} \\
\langle a{\times}b\rangle A/q + r & = & ((\langle a{\times}b\rangle A/q) \vee ((\langle a{\times}b\rangle A/r) \\
\langle a{\times}b\rangle A/rec\ P.t & = & \langle a{\times}b\rangle A/t[rec\ P.t/P]
\end{array}
$$

Figure 3.9: Reduction for product. Syntactic version. We use A/p as shorthand for $\mathrm{red}_{\times p}(A;\sigma)$.

$$
\begin{array}{lcl}
\neg A/p & = & \neg(A/p) \\
A_0 \vee A_1/p & = & (A_0/p) \vee (A_1/p) \\
X/p & = & X_p \\
\mu X.A/p & = & (X_p \ \mathtt{where}_\mu \ (X_{p_1},\dots,X_{p_n}) = (A/p_1,\dots,A/p_n)) \\
 & & \text{where } R_p = \{p_1,\dots,p_m\} \\
\langle a{\times}b\rangle A/p & = & \langle a\rangle \bigvee\{A/p' \mid p \xrightarrow{b} p'\}
\end{array}
$$

Figure 3.10: Reduction for product. Operational version.

$$\models q \times p : A \quad \text{iff} \quad \models q : B.$$

Let $R_p = \{p_1, \dots, p_n\}$ be the finite set of reachable states of p in some fixed enumeration. We define the map $in : \underbrace{\mathcal{P}(R_q \times \dots \times \mathcal{P}(R_q)}_{n} \to \mathcal{P}(R_{q\times p})$ as

$$in(U_{p_1}, \dots, U_{p_n}) = (U_{p_1} \times p_1) \cup \dots \cup (U_{p_n} \times p_n),$$

where $U \times p = \{u \times p \mid u \in U\}$. As usual we have a change of variables σ with $\sigma(X) = IN(X_{p_1}, \dots, X_{p_n})$. We construct $\mathrm{red}_{\times p_i}(A;\sigma)$ such that for a change of variables σ respecting the types of $A[\sigma]$, we have

$$\llbracket A[\sigma] \rrbracket^{(:)}_{q \times p} \rho = in(\llbracket \mathrm{red}_{\times p_1}(A;\sigma) \rrbracket^{(:)}_{q} \rho, \ldots , \llbracket \mathrm{red}_{\times p_n}(A;\sigma) \rrbracket^{(:)}_{q} \rho).$$

As a notational convenience we write A/p for $\mathrm{red}_{\times p}(A;\sigma)$ omitting the σ which is always assumed to map an X into $X_{p_1}, \ldots , X_{p_n}$. The reduction is shown in figure 3.9 in a syntactic version, and in a more operational version in figure 3.10. Figure 3.8 gives a graphical interpretation of A/p.

Theorem 3.7 (Reduction for product)
Assume given a pure and closed assertion A of type $\eta_1 \times \eta_2$, a change of variables σ, and a finitary term p of type η_2 with no restrictions and relabellings. We then have for an arbitrary term q of type η_1:

$$\models (q \times p : A) \leftrightarrow (q : \mathrm{red}_{\times p}(A;\sigma)).$$

Proof: See appendix A.6. □

Notice that termination is ensured by the well-founded order consisting of the number of products in the process term combined lexicographically with the structure of assertions again combined lexicographically with the maximal depth to a prefix in the process term.

The minimum fixed-point gives rise to an assertion with a `where`-clause which could be removed using Bekič's theorem (theorem 2.3) at the expense of potentially increasing the assertion size exponentially. In the next section we consider an example in which the application of Bekič's theorem causes no problems, whereas in the later example in section 3.6 applying Bekič's theorem would yield a rather unpleasant assertion, pointing out precisely why we decided to enrich the calculus with simultaneous fixed-points: To keep assertions small.

The reduction for product is a very powerful construction, which has some striking applications we will consider in great detail in chapter 4. But first a simple example.

3.4 Example: A Researcher and a Coffee Vending Machine

It is an important property of all our reductions (except product) that they only depend on the top-most operator of the process term, hence we can leave part of a process unspecified and still apply the reductions. Technically this can be done by adding *process variables* to our language of processes. Given an assertion and a process with variables, we can then compute a propositional expression with correctness assertions over the variables, expressing what relationship there should be between them in order to make the process satisfy the assertion. In this way the reductions compute what corresponds to weakest preconditions in Hoare logic.

As pointed out in the previous section, the reductions for product has the potential of becoming rather complex if applying Bekič's theorem. In this section we show two examples for which this is not the case.

First we define a binary parallel operator $\|_{K,L}$ which allows its left and right components to independently perform the actions indicated by the sets K and L, except that they are required to synchronize on common actions of K and L. The precise definition is

$$p \|_{K,L} q \overset{\text{def}}{=} (p \times q) \restriction \Lambda\{\Xi\}$$

where $\Lambda = \{a \times a \mid a \in K \cap L\} \cup \{a \times * \mid a \in K \setminus L\} \cup \{* \times a \mid a \in L \setminus K\}$ and

$$\begin{array}{ll} \Xi(a \times a) & = a, \text{ for all } a \in K \cap L \\ \Xi(a \times *) & = a, \text{ for all } a \in K \setminus L \\ \Xi(* \times a) & = a, \text{ for all } a \in L \setminus K \\ \Xi(a) & \text{undefined otherwise.} \end{array}$$

Now assume that we want to construct a small system consisting of a coffee vending machine and a researcher. The coffee machine should be able to accept money and then supply a cup of coffee. The researcher should be able to pay out money, drink coffee, and publish papers. Suppose we know how the researcher behaves, specified by a process term r, but would like to find out what kind of coffee machine x to put into the system, such that eventually the researcher has no other choice than to publish a paper.

In general a property of the form 'eventually only the action a can happen' can be expressed by the assertion

$$\mu X.\langle . \rangle T \wedge [-a]X.$$

Our problem can now be restated.

> Assume the actions to be p for publish, c for taking/giving coffee, m for taking/giving money, and define $K = \{m, c\}, L = \{m, c, p\}$. Which values of x make the following correctness assertion valid
>
> $$x \parallel_{K,L} r : \mu X.\langle . \rangle T \wedge [-p]X?$$

Suppose the researcher r behaves as *rec* $P.m.c.(m.c.P + p.P)$. Then expanding the definition of $\parallel_{K,L}$ and applying the reduction for restriction and relabelling, we get the equivalent correctness assertion

$$x \times r : \mu X.\langle m \times m, c \times c, * \times p \rangle T \wedge [m \times m, c \times c]X$$

and then, by applying the reduction for product, the equivalent

$$x : \mu X.\langle m \rangle T \wedge [m](\langle c \rangle T \wedge [c][m](\langle c \rangle T \wedge [c]X)), \tag{3.3}$$

which using the abbreviation $[a]'A = [a]A \wedge \langle a \rangle T$ gives the more compact

$$x : \mu X.[m]'[c]'[m][c]'X, \tag{3.4}$$

where only the third box-modality is "strong." One can now use (3.4) to verify different proposals for coffee machines, without redoing the first two steps. This might be done by the compositional method applying reductions repeatedly until simple correctness assertions involving only *nil* and state identifiers are met and the overall satisfaction problem will reduce to true or false, or for closed terms by other model checking algorithms as the algorithms to be presented in chapter 5.

An interesting point to note about the assertion in (3.4) is that, although the researcher r had *four* reachable states, and then potentially four fixed-points could appear, only *one* fixed-point appears in the resulting assertion.

Returning to the example, we can verify that a successful choice of x is $m.c.nil$ i.e. a coffee vending machine that accepts money and give coffee once, and then breaks down, whereas *rec* $P.m.c.P$ is an unsuccessful choice. Reading the assertion in (3.4) carefully, we can express the requirement to the machine as 'after having offered a finite and odd number of m's followed by c's, no m should be offered.'

Changing the behaviour of the researcher slightly and taking $r =$ *rec* $P.m.c.P + m.c.p.P$ and performing the reductions for restriction, relabelling, and product, we arrive at the correctness assertion $x : F$, i.e. there are no coffee vending machines that will make the system fulfill the requirement.

$$\begin{array}{rcl}
(A_1,\ldots,A_l)/p_i & = & (A_1/p_i,\ldots,A_l/p_i) \\
(\vec{A}^l\ \mathtt{where}_\mu\ \vec{Y}^m = \vec{P}^m)/p_i & = & (\vec{A}^l/p_i)\ \mathtt{where}_\mu\ \vec{Z}^{mn} = (\vec{P}^m/\vec{p}^n) \\
 & & \text{where } \vec{Z}^{mn} = \sigma(\vec{Y}^m) = (\sigma(Y_1),\ldots,\sigma(Y_m)) \\
\langle\gamma_0\times\gamma_1\rangle A/p_i & = & \langle\gamma_0\rangle \bigvee_{a,s;\,p_i\xrightarrow{a}s} (\gamma_1 = a)\wedge(A/s) \\
Q/p_i & = & Q_{\mathrm{red}_{\times p_i}} \\
\exists\alpha.A/p_i & = & \exists\alpha.(A/p_i) \\
\psi(\gamma)/p_i & = & \psi(\gamma) \\
 & & \\
(\vec{A}^l\ \mathtt{where}_\mu\ \vec{Y}^m = \vec{P}^m)/\vec{p}^n & = & (\vec{A}^l/\vec{p}^n)\ \mathtt{where}_\mu\ \vec{Z}^{mn} = (\vec{P}^m/\vec{p}^n) \\
 & & \text{where } \vec{Z}^{mn} = \sigma(\vec{Y}^n) \\
(A_1,\ldots,A_l)/\vec{p}^n & = & (A_1/\vec{p}^n,\ldots,A_l/\vec{p}^n) \\
A/\vec{p}^n & = & (A/p_1,\ldots,A/p_n) \\
 & & \text{if } A \text{ is not a product assertion}
\end{array}$$

Figure 3.11: Reduction for product in the extended calculus. The missing cases are as for the standard calculus (see figure 3.9). Again A/p_i abbreviates $\mathrm{red}_{\times p_i}(A;\sigma)$ and $A/\stackrel{\rightarrow n}{p}$ abbreviates $\mathrm{red}_{\times \stackrel{\rightarrow n}{p}}(A;\sigma)$

3.5 Reductions for the Extended Calculus

In generalizing the reductions to the extended calculus, we only consider the static operators. We consider each new construction in $\mu\mathrm{K}_{ext}$ in turn. The reductions for the three operators are given in figures 3.11, 3.12 and 3.13.

Simultaneous fixed-points. The reduction for product already gave rise to simultaneous fixed-points and it is not difficult to extend the reductions to work on assertions that already contain simultaneous fixed-points. For restriction and relabelling this is straightforward. For the product we define a reduction $\mathrm{red}_{\times \stackrel{\rightarrow n}{p}}(A;\sigma)$ with $R_p = \{p_1,\ldots,p_n\}$ giving an assertion of arity n with the property

$$[\![A[\sigma]]\!]_{q\times p}\rho\ \phi = in([\![\mathrm{red}_{\times \stackrel{\rightarrow n}{s}}(A;\sigma)]\!]_q)$$

where as before $in : \mathcal{P}(R_q)^n \to \mathcal{P}(R_{q\times p})$ is defined by

$$in(U_1,\ldots,U_n) = (U_1\times p_1)\cup\ldots\cup(U_n\times p_n).$$

For keeping the size of the resulting assertions small in reducing the simultaneous fixed-points, it is important that we define the reduction $\mathrm{red}_{\times \vec{p}^n}(A;\sigma)$ directly instead of taking it to be the n-tuple of reduced assertions $(\mathrm{red}_{\times p_1}(A;\sigma), \ldots, \mathrm{red}_{\times p_n}(A;\sigma))$. This point is discussed later.

Constants. For the constants we must for each reduction assume the presence of other constants which in the universal valuation is related through the respective *in*-maps. For instance for the reduction for restriction $\mathrm{red}_{\restriction\Lambda}$ we assume for each constant Q the presence of another constant $Q_{\mathrm{red}_{\restriction\Lambda}}$ with the property that[5]

$$\mathcal{V}(Q_{\mathrm{red}_{\restriction\Lambda}} \restriction \Lambda = \mathcal{V}(Q)$$

and therefore

$$\models (t \restriction \Lambda : Q) \leftrightarrow (t : Q_{\mathrm{red}_{\restriction\Lambda}}).$$

Action quantifiers and action predicates. For restriction and product the existential action-quantifier is easily handled, intuitively because it is an (infinite) disjunction and the reductions commutes with disjunction. Similarly, the action predicates cause no problems for restriction and product.

For relabelling a little extra is needed. To see why, assume that the relabelling map 'has type $\eta \to \cdot$' for some type η. I.e. Ξ has domain a subset of $[\![\eta]\!]$ and image a subset of $[\![\ \cdot\ ]\!]$. Then a modality $\langle\alpha\rangle A$ should now, inspired by the reductions for the standard calculus, be replaced by a modality $\langle\gamma\rangle A'$ of type η such that "when applying Ξ to γ we get α". This suggests that we must extend the notion of change of variables to action variables. Hence we say that σ *is an extended change of variables for the assertion A and the relabelling* Ξ, if it is a change of assertion variables for A as before, and if it maps action variables α to an expression $\Xi(\eta(\beta_1, \ldots, \beta_k))$ where Ξ plays a role similar to that of *IN* and $\eta(\beta_1, \ldots, \beta_k)$ is constructed from the type η by filling in the variables $\beta_1, \ldots, \beta_k$ for the 'holes' of η indicated by $\cdot$. The semantics of $\Xi(\eta(\beta_1, \ldots, \beta_k))$ is simply

[5]Recall that for a set $U \subseteq Proc_{WPA}$ the set $U \restriction \Lambda$ is the result of syntactically operating by restriction on each element of U.

$$[\![\Xi(\eta(\beta_1,\dots,\beta_k))]\!]\phi = \Xi(\eta(\phi(\beta_1),\dots,\phi(\beta_k)))$$

where $\eta(\phi(\beta_1),\dots,\phi(\beta_k))$ is now the composite action constructed by filling in $\phi(\beta_1),\dots,\phi(\beta_k)$ for the 'holes' of η.

We also require that σ is fresh with respect to the action variables, i.e. for two variables α_0 and α_1, the free variables of $\sigma(\alpha_0)$ and $\sigma(\alpha_1)$ are disjoint, and also disjoint from any other variables of A.

Hence the variables β_i are also variables ranging over $[\![\cdot]\!] = Act \cup \{*\}$, and a quantification $\exists\alpha$ will be replaced by the quantification $\exists\beta_1\dots\exists\beta_k$ when $\sigma(\alpha) = \eta(\beta_1,\dots,\beta_k)$.

Theorem 3.8 (Reduction for product in μK$_{\text{ext}}$) *Assume given a closed assertion A in μK$_{ext}$ of type $\eta_1 \times \eta_2$, a change of variables σ, and a term p of type η_2. We then have for an arbitrary term q of type η_1:*

$$\models (q \times p : A) \leftrightarrow (q : \mathrm{red}_{\times p}(a;\sigma))$$

Proof: A straightforward extension of the case of the standard calculus as given in appendix A.6. □

$$\boxed{\begin{array}{rcl}
\mathbf{red}_{\restriction\Lambda}(\vec{A}^{n}\ \mathtt{where}_\mu\ \vec{Y}^{m} = \vec{P}^{m};\sigma) & = & \mathbf{red}_{\restriction\Lambda}(\vec{A}^{n};\sigma)\ \mathtt{where}_\mu\ \sigma(\vec{Y}^{m}) = \mathbf{red}_{\restriction\Lambda}(\vec{P}^{m};\sigma) \\
\mathbf{red}_{\restriction\Lambda}((A_1,\dots,A_l);\sigma) & = & (\mathbf{red}_{\restriction\Lambda}(A_1;\sigma),\dots,\mathbf{red}_{\restriction\Lambda}(A_l;\sigma)) \\
\mathbf{red}_{\restriction\Lambda}(\langle\gamma\rangle A;\sigma) & = & (\gamma \in \Lambda) \wedge \langle\gamma\rangle\mathbf{red}_{\restriction\Lambda}(A) \\
\mathbf{red}_{\restriction\Lambda}(\exists\alpha.A;\sigma) & = & \exists\alpha.\mathbf{red}_{\restriction\Lambda}(A;\sigma) \\
\mathbf{red}_{\restriction\Lambda}(\psi(\gamma);\sigma) & = & \psi(\gamma) \\
\mathbf{red}_{\restriction\Lambda}(Q;\sigma) & = & Q_{\mathbf{red}_{\restriction\Lambda}}
\end{array}}$$

Figure 3.12: Reduction for restriction in the extended calculus. An assertion $\langle a\rangle A$ for $a \in Act \cup \{*\}$ is considered an abbreviation for $\exists\alpha.(\alpha = a) \wedge \langle\alpha\rangle A$ for some *$\alpha \notin fv(A)$.*

Theorem 3.9 (Reduction for restriction in μK$_{\text{ext}}$) *Let η be a type and $\Lambda \subseteq [\![\eta]\!]$ a restriction set. Assume A is a closed assertion in μK$_{ext}$ of type η, and t is any process term, then*

$$\models (t\{\Xi\} : A) \leftrightarrow (t : \mathrm{red}_{\{\Xi\}}(A;\sigma))$$

$$\boxed{\begin{array}{rcl}
\mathbf{red}_{\{\Xi\}}(\vec{A}^n\ \mathtt{where}_\mu\ \vec{Y}^m = \vec{P}^m;\sigma) & = & \mathbf{red}_{\{\Xi\}}(\vec{A}^n;\sigma)\ \mathtt{where}_\mu\ \sigma(\vec{Y}^m) = \mathbf{red}_{\{\Xi\}}(\vec{P}^m;\sigma) \\
\mathbf{red}_{\{\Xi\}}((A_1,\ldots,A_l);\sigma) & = & (\mathbf{red}_{\{\Xi\}}(A_1;\sigma),\ldots,\mathbf{red}_{\{\Xi\}}(A_l;\sigma)) \\
\mathbf{red}_{\{\Xi\}}(\langle\alpha\rangle A;\sigma) & = & \langle\eta(\beta_1,\ldots,\beta_k)\rangle\mathbf{red}_{\{\Xi\}}(A) \\
 & & \qquad \text{where } \sigma(\alpha) = \Xi(\eta(\beta_1,\ldots,\beta_k)) \\
\mathbf{red}_{\{\Xi\}}(\exists\alpha.A;\sigma) & = & \exists\beta_1\ldots\beta_k.\mathbf{red}_{\{\Xi\}}(A;\sigma) \\
 & & \qquad \text{where } \sigma(\alpha) = \Xi(\eta(\beta_1,\ldots,\beta_k)) \\
\mathbf{red}_{\{\Xi\}}(\psi(\alpha);\sigma) & = & \psi\circ\Xi(\alpha) \\
\mathbf{red}_{\{\Xi\}}(Q;\sigma) & = & Q_{\mathbf{red}_{\{\Xi\}}}
\end{array}}$$

Figure 3.13: Reduction for relabelling in the extended calculus. Like in figure 3.12 an assertion $\langle a\rangle A$ for $a \in Act \cup \{*\}$ is considered an abbreviation for $\exists\alpha.(\alpha = a) \wedge \langle\alpha\rangle A$ for some $\alpha \notin fv(A)$.

for a change of variables σ which is fresh for A.

Proof: Also a straightforward extension of the case of the standard calculus as given in appendix A.4, the only non-trivial, yet simple case being the modality. □

Theorem 3.10 (Reduction for relabelling in $\mu\mathrm{K}_{\mathbf{ext}}$) *Let η be a type and Ξ be relabelling map with domain a subset of $[\![\eta]\!]$ and image the whole of $Act \subseteq [\![\cdot]\!]$. Assume A is a closed assertion in $\mu\mathrm{K}_{ext}$, and t is any process term, then*

$$\models (t\{\Xi\} : A) \leftrightarrow (t : \mathrm{red}_{\{\Xi\}}(A;\sigma))$$

for an extended change of variables σ which is fresh for A.

Proof: We sketch the proof for three cases of the existential quantifier, the action predicate and the diamond modality, the others are simple. We use the induction hypothesis:
For all ρ, ϕ

$$[\![A[\sigma]]\!]^{:}_{t\{\Xi\}}\ \rho\ \phi = ([\![\mathrm{red}_{\restriction\Xi}(A;\sigma)]\!]^{(:)}_{t}\rho\ \phi)\{\Xi\}. \qquad (3.5)$$

For the existential quantifier, we proceed as follows:

$[\![(\exists\alpha.A)[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi$

$$
\begin{aligned}
&= \{s \in R_{t\{\Xi\}} \mid \exists a \in \mathit{Act} \cup \{*\}.\ s \in [\![A[\sigma\backslash\alpha]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi[a/\alpha]\} \\
&= \{s \in R_t \mid \exists a \in \mathit{Act} \cup \{*\}.\ s\{\Xi\} \in [\![A[\sigma\backslash\alpha]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi[a/\alpha]\}\{\Xi\} \\
&= \{s \in R_t \mid \exists b_1, \ldots, b_k \in \mathit{Act} \cup \{*\}. \\
&\qquad \Xi(\eta(b_1, \ldots, b_k)) = a\ \&\ s\{\Xi\} \in [\![A[\sigma\backslash\alpha]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi[a/\alpha]\}\{\Xi\} \\
&\qquad \text{as } \Xi \text{ is surjective} \\
&= \{s \in R_t \mid \exists b_1, \ldots, b_k \in \mathit{Act} \cup \{*\}.\ s\{\Xi\} \in \\
&\qquad\quad [\![A[\sigma]]\!]^{:}_{t\{\Xi\}}\ \rho\ \phi[b_1/\beta_1, \ldots, b_k/\beta_k]\}\{\Xi\} \\
&\qquad \text{assuming } \sigma(\alpha) = \eta(\beta_1, \ldots, \beta_k) \\
&= \{s \in R_t \mid \exists b_1, \ldots, b_k \in \mathit{Act} \cup \{*\}.\ s \in \\
&\qquad\quad [\![\mathrm{red}_{\{\Xi\}}(A;\sigma)]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi[b_1/\beta_1, \ldots, b_k/\beta_k]\}\{\Xi\} \\
&\qquad \text{by the induction hypothesis} \\
&= ([\![\exists\beta_1, \ldots, \beta_k.\ \mathrm{red}_{\{\Xi\}}(A;\sigma)]\!]^{(:)}_t\ \rho\ \phi)\{\Xi\}
\end{aligned}
$$

For the action predicate:

$$
\begin{aligned}
[\![\psi(\alpha)[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi &= [\![\psi(\alpha)[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi \\
&\qquad \text{by definition of substitution} \\
&= [\![\psi(\alpha)[\sigma]]\!]^{(:)}_t \rho\ \phi \\
&\qquad \text{as the predicate denotes true or false independently} \\
&\qquad \text{of the transition system} \\
&= [\![\psi \circ \Xi(\eta(\beta_1 \ldots, \beta_k)]\!]^{(:)}_t \rho\ \phi \\
&\qquad \text{where } \psi \circ \Xi \text{ is the predicate with } \psi \circ \Xi(a) = \psi(\Xi(a)).
\end{aligned}
$$

For the modality:

$$
\begin{aligned}
&[\![(\langle\alpha\rangle A)[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi \\
&\quad = \{s \in R_{t\{\Xi\}} \mid \exists s' \in R_{t\{\Xi\}}.\ s \overset{[\![\sigma(\alpha)]\!]\phi}{\rightarrow} s'\ \&\ s' \in [\![A[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi\} \\
&\quad = \{s \in R_{t\{\Xi\}} \mid \exists s' \in R_{t\{\Xi\}}.\ s \overset{\Xi([\![\eta(\beta_1 \ldots \beta_k)]\!]\phi)}{\rightarrow} s'\ \&\ s' \in [\![A[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi\} \\
&\qquad\qquad \text{where } \sigma(\alpha) = \eta(\beta_1, \ldots, \beta_k) \\
&\quad = \{s \in R_t \mid \exists s' \in R_t.\ s\{\Xi\} \overset{\Xi([\![\eta(\beta_1 \ldots \beta_k)]\!]\phi)}{\rightarrow} s'\{\Xi\}\ \&\ s'\{\Xi\} \in \\
&\qquad\qquad [\![A[\sigma]]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi\}\{\Xi\} \\
&\qquad\qquad \text{since } R_{t\{\Xi\}} = (R_t)\{\Xi\} \\
&\quad = \{s \in R_t \mid \exists s' \in R_t.\ s\{\Xi\} \overset{\Xi([\![\eta(\beta_1 \ldots \beta_k)]\!]\phi)}{\rightarrow} s'\{\Xi\}\ \&\ s'\{\Xi\} \in \\
&\qquad\qquad [\![\mathrm{red}_{\{\Xi\}}(A;\sigma)]\!]^{(:)}_{t\{\Xi\}}\ \rho\ \phi\}\{\Xi\} \\
&\qquad\qquad \text{by the induction hypothesis}
\end{aligned}
$$

$$= \{s \in R_t \mid \exists s' \in R_t.\ s \overset{[\![\eta(\beta_1 \ldots \beta_k)]\!]\phi}{\longrightarrow} s' \ \&\ s' \in [\![\mathrm{red}_{\{\Xi\}}(A;\sigma)]\!]_t^{(;)}\ \rho\ \phi\}\{\Xi\}$$

by the operational rule for relabelling

$$= ([\![\langle \eta(\beta_1 \ldots \beta_k)\rangle \mathrm{red}_{\{\Xi\}}(A;\sigma)]\!]_{t\{\Xi\}}^{(;)})\{\Xi\}$$

□

It can be useful to derive reductions for some abbreviations. For instance, if we consider the assertion $\langle\Delta\rangle A$ for a set $\Delta \subseteq \mathit{Act}$ abbreviating $\exists\alpha.(\alpha \in \Delta) \wedge \langle\alpha\rangle A$ we find the following derived reductions:

$$\mathrm{red}_{\restriction\Lambda}(\langle\Delta\rangle A) = \langle\Delta \cup \Lambda\rangle \mathrm{red}_{\restriction\Lambda}(A) \tag{3.6}$$

$$\mathrm{red}_{\{\Xi\}}(\langle\Delta\rangle A) = \langle\Xi^{-1}(\Delta)\rangle \mathrm{red}_{\{\Xi\}}(A) \tag{3.7}$$

$$\mathrm{red}_{\times p}(\langle\Delta\rangle A) = \bigvee_{a,p';p \xrightarrow{a} p'} \langle\Delta'\rangle \mathrm{red}_{\times p'}(A) \tag{3.8}$$

where in (3.8) $\Delta' = \{b \mid b \times a \in \Delta\}$.

3.6 Example: A Message Handling System

In the sequel we re-examine the message handling system of example 2.1. Recall that the system consists of a *sender* S, a *receiver* R, and a *medium* M communicating on the channels *send*, *rec*, ack_r, and ack_s. To illustrate the compositional approach we assume that S and R are known to be defined as

$$S = \mathit{send}!ack_s?S$$
$$R = \mathit{rec}?ack_r!R$$

and M is yet unknown. Furthermore let us assume that we do not want to put any restriction on the visible actions of the system, although we do forbid external communication. Hence the system we are constructing has the form

$$\mathit{Sys} = (S \parallel M \parallel R) \restriction \mathcal{A}$$

where $\parallel$ is the parallel composition of OPA and $\mathcal{A}$ are the neutral actions of OPA. Suppose we are interested in deducing an assertion for M to satisfy in order for the complete system to be deadlock-free, i.e. *Sys* should satisfy the formula *DeadLockFree* defined in section 2.5 by

$$DeadLockFree = Never(DeadLock)$$

where

$$DeadLock = [.]F \wedge \neg WellTerm$$

and

$$Never(X) = \nu Y.[.]Y \wedge \neg X.$$

But before proceeding with this example let us derive a reduction for $\|$ from the definition of $\|$:

$$\begin{array}{l} p \parallel q =_{\text{def}} (p \times q) \restriction \Lambda_{\text{OPA}}\{\Xi_{\text{OPA}}\} \\ \qquad\qquad \text{where} \\ \qquad\qquad\qquad \begin{array}{lcl} \Lambda_{\text{OPA}} & = & (Act \times \{*\}) \cup (\{*\} \times Act) \\ & & \cup \{c? \times c!, c! \times c? \mid c \in \mathcal{A}\} \\ \Xi_{\text{OPA}}(x) & = & \begin{cases} a & \text{if } x \equiv a \times * \text{ or } x \equiv * \times a, a \in Act \\ c & \text{if } x \equiv c? \times c! \text{ or } x \equiv c! \times c?, c \in \mathcal{A} \\ \text{undefined} & \text{otherwise} \end{cases} \end{array} \end{array}$$

First, assume we have given a closed assertion A and change of variables σ and σ' for the reductions of $\restriction \Lambda_{\text{OPA}}$ and $\{\Xi_{\text{OPA}}\}$. Let us compute $\text{red}_{\restriction\Lambda_{\text{OPA}}}$ $(\text{red}_{\{\Xi_{\text{OPA}}\}}(A;\sigma);\sigma')$, i.e. the combined effect of performing first the reduction for relabelling and then the reduction for restriction. Notice, that the relabelling with Ξ_{OPA} changes the type of the assertion from $\cdot$ to $\cdot \times \cdot$. Now, omitting for brevity the change of variables σ which we assume has $\sigma(\alpha) = \Xi(\beta_1 \times \beta_2)$ we rewrite as follows:

$$\begin{array}{rcl} \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(\exists\alpha.A)) & = & \text{red}_{\restriction\Lambda_{\text{OPA}}}(\exists\beta_1,\beta_2.A')) \\ & & \quad \text{where } A' = \text{red}_{\{\Xi_{\text{OPA}}\}}(A) \\ & & = \exists\beta_1,\beta_2.A' \\ & & \quad \text{where } A' = \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(A)) \\ \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(\alpha \in \Lambda)) & = & \text{red}_{\restriction\Lambda_{\text{OPA}}}(\alpha \in \Xi^{-1}(\Lambda)) \\ & & = \alpha \in \Xi^{-1}(\Lambda) \\ \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(\langle\alpha\rangle A)) & = & (\beta_1 \times \beta_2 \in \Lambda_{\text{OPA}}) \wedge \langle\beta_1 \times \beta_2\rangle A' \\ & & \quad \text{where } A' = \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(A)) \\ \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(\langle\Delta\rangle A)) & = & \text{red}_{\restriction\Lambda_{\text{OPA}}}(\langle\Xi_{\text{OPA}}^{-1}(\Delta)\rangle A') \\ & & \quad \text{where } A' = \text{red}_{\{\Xi_{\text{OPA}}\}}(A)) \text{ using}(3.7) \\ & = & \langle\Xi_{\text{OPA}}^{-1}(\Delta) \cap \Lambda_{\text{OPA}}\rangle A' \\ & & \quad \text{where } A' = \text{red}_{\restriction\Lambda_{\text{OPA}}}(\text{red}_{\{\Xi_{\text{OPA}}\}}(A)) \text{ using } (3.6) \end{array}$$

Using these, we can compute the reductions for $\mathrm{red}_{\|P}(A)$ (see figure 3.14). We also get the derived reduction:

$$\begin{aligned} \mathrm{red}_{\|p}(\langle\Delta\rangle A) &= \bigvee_{a,p';p\xrightarrow{a}p'} \langle\Delta'\rangle \mathrm{red}_{\|p'}(A) \\ &\quad \text{where } \Delta' = \{b \mid b \times a \in \Xi_{\mathrm{OPA}}^{-1}(\Delta) \cap \Lambda_{\mathrm{OPA}}\} \end{aligned} \tag{3.9}$$

$$\boxed{\begin{array}{rcl}
(A_1,\ldots,A_l)/\!\!/p_i &=& (A_1/\!\!/p_i,\ldots,A_l/\!\!/p_i) \\
(\vec{A}^l \ \mathtt{where}_\mu\ \vec{Y}^m = \vec{P}^m)/\!\!/p_i &=& (\vec{A}^l/\!\!/p_i)\ \mathtt{where}_\mu\ \vec{Z}^{mn} = (\vec{P}^m/\!\!/\vec{p}^n) \\
&& \quad \text{where } \vec{Z}^{mn} = \sigma(\vec{Y}^m) = (\sigma(Y_1),\ldots,\sigma(Y_m)) \\
\langle\alpha\rangle A/\!\!/p_i &=& \langle\beta_1\rangle \displaystyle\bigvee_{a,s;p_i\xrightarrow{a}s} (\beta_2 = a) \wedge (\beta_1 \times a \in \Lambda_{WPA}) \wedge (A/\!\!/s) \\
&& \quad \text{where } \sigma(\alpha) = \Xi_{WPA}(\beta_1 \times \beta_2) \\
Q/\!\!/p_i &=& Q' \\
&& \quad \text{where } (\mathcal{V}(Q') \times p_i) \restriction \Lambda WPA\{\Xi_{WPA}\} \\
((Q_{\mathrm{red}_{\{\Xi_{WPA}\}}})_{\mathrm{red}_{\restriction\Lambda_{WPA}}})_{\mathrm{red}_{\times p_i}} && \\
\exists\alpha.A/\!\!/p_i &=& \exists\beta_1,\beta_2.A/\!\!/p_i \\
&& \quad \text{where } \sigma(\alpha) = \Xi_{WPA}(\beta_1 \times \beta_2) \\
\psi(\alpha)/\!\!/p_i &=& \psi \circ \Xi_{WPA}(\beta_1 \times \beta_2) \\
&& \\
(\vec{A}^l \ \mathtt{where}_\mu\ \vec{Y}^m = \vec{P}^m)/\!\!/\vec{p}^n &=& (\vec{A}^l/\!\!/\vec{p}^n)\ \mathtt{where}_\mu\ \vec{Z}^{mn} = (\vec{P}^m/\!\!/\vec{p}^n) \\
&& \quad \text{where } \vec{Z}^{mn} = \sigma(\vec{Y}^n) \\
(A_1,\ldots,A_l)/\!\!/\vec{p}^n &=& (A_1/\!\!/\vec{p}^n,\ldots,A_l/\!\!/\vec{p}^n) \\
A/\!\!/\vec{p}^n &=& (A/\!\!/p_1,\ldots,A/\!\!/p_n) \\
&& \quad \text{if } A \text{ is not a product assertion}
\end{array}}$$

Figure 3.14: Reductions for the parallel operator $\|$ of OPA. We have abbreviated $\mathrm{red}_{\|P}(A;\sigma)$ by $A/\!\!/p$.

Example 3.1 Let us now compute in detail the assertion expressing the requirement to M in $Sys = (M \parallel S \parallel R) \restriction \mathcal{A}$ when we want Sys to satisfy *DeadLockFree*.

$$\begin{aligned} &\mathrm{red}_{\|S}\ (\mathrm{red}_{\|R}\mathrm{red}_{\restriction\mathcal{A}}(\nu Y.[.]Y \wedge (\langle.\rangle T \vee \mathit{WellTerm})))) \\ &\quad = \mathrm{red}_{\|S}(\mathrm{red}_{\|R}(\nu Y.[\mathcal{A}]Y \wedge (\langle\mathcal{A}\rangle T \vee \mathit{WellTerm}))) \end{aligned}$$

$$\begin{aligned}
&\text{using (3.6)}\\
&= \mathrm{red}_{\|S}(Y_R \ \mathtt{where}_\nu \begin{pmatrix} Y_R \\ Y_{R'} \end{pmatrix} = \begin{pmatrix} \mathrm{red}_{\|R}(A) \\ \mathrm{red}_{\|R'}(A) \end{pmatrix})\\
&\quad \text{where } A = [\mathcal{A}]Y \wedge (\langle\mathcal{A}\rangle T \vee \mathit{WellTerm})\\
&\quad \text{and } R' = \mathit{ack}_r!R
\end{aligned}$$

For $\mathrm{red}_{\|R}(A)$ we use figure 3.14 to get

$$\begin{aligned}
\mathrm{red}_{\|R}(\mathit{WellTerm}) &= F\\
\mathrm{red}_{\|R}(\langle A\rangle T) &= \langle\Delta'\rangle \mathrm{red}_{\|R}(T) \vee \langle\Delta''\rangle \mathrm{red}_{\|R'}(T)\\
&\quad \text{where } \Delta' = \{b \mid b \times * \in \Xi^{-1}_{\mathrm{OPA}}(\mathcal{A}) \cup \Lambda_{\mathrm{OPA}}\}\\
&\quad \text{and } \Delta'' = \{b \mid b \times \mathit{rec}? \in \Xi^{-1}_{\mathrm{OPA}}(\mathcal{A}) \cup \Lambda_{\mathrm{OPA}}\}\\
&\quad \text{by (3.9)}\\
&= \langle\Delta'\rangle T \vee \langle\Delta''\rangle T\\
&\quad \text{where } \Delta' = \mathcal{A}\\
&\quad \text{and } \Delta'' = \{\mathit{rec}!\}\\
&= \langle\mathcal{A}\rangle T \vee \langle \mathit{rec}!\rangle T\\
\mathrm{red}_{\|R}([\mathcal{A}]Y) &= [\mathcal{A}]Y_R \wedge [\mathit{rec}!]Y_{R'}\\
&\quad \text{with steps like above}
\end{aligned}$$

This yields

$$\mathrm{red}_{\|R}(A) = [\mathit{rec}!]Y_{R'} \wedge [\mathcal{A}]Y_R \wedge (\langle \mathit{rec}!\rangle T \vee \langle\mathcal{A}\rangle T).$$

The reduction $\mathrm{red}_{\|R'}(A)$ proceeds in an analogous way resulting in

$$\mathrm{red}_{\|R'}(A) = [\mathit{ack}_r?]Y_R \wedge [\mathcal{A}]Y_{r''} \wedge (\langle \mathit{ack}_r?\rangle T \vee \langle\mathcal{A}\rangle T).$$

Using the convention that $[\Delta]'A = [\Delta]A \wedge \langle\Delta\rangle T$ an re-arranging the two assertions above, we end up with

$$B = Y_R \ \mathtt{where}_\nu \begin{pmatrix} Y_R \\ Y_{R'} \end{pmatrix} = \begin{pmatrix} [\mathit{rec}!]'Y_{R'} \vee [\mathcal{A}]'Y_R \\ [\mathit{ack}_r?]'Y_R \vee [\mathcal{A}]'Y_{R'} \end{pmatrix}$$

Proceeding in the same way we get that $\mathrm{red}_{\|S}(B)$ reduces to:

$$C = Y_{RS} \ \mathtt{where}_\nu \begin{pmatrix} Y_{RS} \\ Y_{R'S} \\ Y_{RS'} \\ Y_{R'S'} \end{pmatrix} = \begin{pmatrix} [\mathit{rec}!]'Y_{R'S} \vee [\mathit{send}?]'Y_{RS'} \vee [\mathcal{A}]'Y_{RS} \\ [\mathit{ack}_r?]'Y_{RS} \vee [\mathit{send}?]'Y_{R'S'} \vee [\mathcal{A}]'Y_{R'S} \\ [\mathit{rec}!]'Y_{R'S'} \vee [\mathit{ack}_s!]'Y_{RS} \vee [\mathcal{A}]'Y_{RS'} \\ [\mathit{ack}_r?]'Y_{RS'} \vee [\mathit{ack}_s!]'Y_{R'S} \vee [\mathcal{A}]'Y_{R'S'} \end{pmatrix}$$

Now, *any* medium M must satisfy C to ensure that Sys is free of deadlock, i.e. we have shown

$$\models ((M \parallel S) \parallel R) \upharpoonright \mathcal{A} : DeadLockFree \Leftrightarrow \models M : C.$$

In particular, the medium called M in example 2.1 satisfy C, whereas M' in the same example does not. However, also some rather bizarre processes, behaving far from how we intended the overall system to work, satisfy C. Two distinct examples are

$$\begin{array}{rcl} M^1 & = & \tau.M^1. \\ M^2 & = & send.M^2. \end{array}$$

The problem with M^1 is that it all by itself keeps on being busy doing some irrelevant actions and therefore the complete system cannot go into a deadlock. We could avoid this by changing the restricting set from $\mathcal{A}$ to $\{send, rec, ack_r, ack_s\}$ thereby excluding any irrelevant actions from providing deadlock-freeness. However, this would not rule out M^2 which is simply autonomously performing neutral *send*-actions; M^2 could be ruled out by requiring the medium to only perform input and output on the channels $\{send, rec, ack_r, ack_s\}$ forbidding neutral actions to occur as anything else than the result of a communication (eg. by disallowing neutral prefixes).

On the other hand, these problems could be expected as we are only defining *requirements* to our system, that is a *partial specification* saying that no deadlock must be possible. Surely there are other important properties of the system we are interested in! These could be captured by other formulas, and through the reductions we could deduce other requirements for the missing medium. This could for instance be fairness properties like an assertion expressing that infinitely often a message could be send, captured by the assertion

$$\nu X.[.]'X \wedge (\mu Y.\langle.\rangle Y \vee \langle send \rangle T).$$

(In chapter 4 we describe a way of constructing such assertions.) □

3.7 Algorithmic Aspects

The reductions presented in this chapter have a clear algorithmic flavour. Given an assertion A and a term $t = op(t_1, \ldots, t_k)$ we can compute an expression B in $\mathcal{L}$ over correctness assertions involving $t_1, \ldots, t_k$ such that

$$\models (t : A) \leftrightarrow B.$$

As a very pragmatic question we might ask, how big can B get? For *nil*, prefix, restriction and relabelling it is not hard to see that B cannot be essentially bigger than A, i.e. $O(|A|)$ and for restriction and relabelling this even holds in the extended calculus. For sum we have

$$|\mathrm{red}^0(t_0 + t_1 : A; \sigma)| = |A|$$

as red^0 only renames variables, but for $\mathrm{red}^1(t_0 + t_1 : A; \sigma)$ the situation is different. The clause for a fixed-point is

$$\mathrm{red}^1(t_0{+}t_1 : \mu X.A; \sigma) = \mathrm{red}^1(t_0{+}t_1 : A; \sigma)[\mathrm{red}^0(t_0{+}t_1 : \mu X.A; \sigma)/X_0][F/X_1]$$

and there is nothing that prevents the resulting assertion from being "exponentially bigger" than A. An example indicating how this can take place is provided by $A \equiv \nu Y.\mu X.\langle a\rangle(Y \wedge X)$ which reduces to

$$\begin{gathered}((t_0 : \langle a\rangle(Y_0 \wedge X_0)) \vee (t_1 : \langle a\rangle(Y_0 \wedge X_0))) \\ [\mu X_0.\langle a\rangle(Y_0 \wedge X_0)/X_0][\nu Y_0.\mu X_0.\langle a\rangle(Y_0 \wedge X_0)/Y_0]\end{gathered}$$

which after performing the substitutions contains four ν's and six μ's, instead of the original one of each!

For product the simultaneous fixed-points actually allow us to get quite compact reduced assertions, especially for assertions in $\mu\mathrm{K}_{\texttt{where},Q}$ that are on a *simple form*.

Definition 3.2 A simultaneous fixed-point assertion $(\vec{A}^m \ \texttt{where}_\mu \ \vec{X}^n = \vec{B}^n)$ in positive, normal form is said to be *simple* if each of the components B_i and each of the A_i is *simple*, i.e. contains at most one operator corresponding to one of the forms

$$\begin{gathered}Q, \neg Q, F, T, X \vee X', X \wedge X', \langle a\rangle X, [a]X, X \\ A \ \texttt{where}_\mu \ \vec{X}^m = \vec{B}^m, \ \texttt{where}_\nu \ \vec{X}^m = \vec{B}^m.\end{gathered}$$

An assertion A in positive, normal form is *simple* if all subassertions of A are simple. □

Now, any assertion can be transformed to an equivalent simple assertion.

Lemma 3.5 *Let A be any assertion in $\mu K_{\mathtt{where},Q}$ in positive, normal form. Then there exists a simple assertion A' satisfying*

$$\begin{array}{rl} (i) & \models A \leftrightarrow A' \\ (ii) & A' \textit{ has at most } |A| \textit{ variables,} \\ (iii) & A' \textit{ has size } O(|A|), \\ (iv) & A' \textit{ can be computed in time } O(|A'|), \textit{ and} \\ (v) & ad(A') = \max\{1, ad(A)\}. \end{array}$$

Proof: (Sketch) If the top-level operator of A is a `where`-clause; take $A^0 \equiv A$, otherwise for an arbitrary X, take

$$A^0 \equiv (X \ \mathtt{where}_\mu \ X = A).$$

This initial transformation at most increases the size with three, and possibly increases the alternation depth from zero to one, otherwise it stays the same.

For each `where`-clause $(\vec{B}' \ \mathtt{where}_\mu \ \vec{X} = \vec{B})$ in A_0 with a $\vec{B}'$ that is not a tuple of variables, replace it with the assertion $(\vec{Y} \ \mathtt{where}_\mu \ \vec{Y}\vec{X} = \vec{B}'\vec{B})$ renaming variables such that no name-clashes occur. This does not increase the alternation depth, and all in all at most increases the size of the assertion with a factor of three. Call the resulting assertion A^1.

Now, suppose A^1 contains an assertion $B^1 \equiv (B' \ \mathtt{where}_\mu \ X = B)$ where B does not contain any `where`-assertions. To each subassertion of B we associate a variable. This gives $n = |B|$ variables $\{X_1, \ldots, X_n\}$. Define a new n-ary fixed-point assertion

$$B^2 \equiv (B'[X_1/X] \ \mathtt{where}_\mu \ \vec{X}^n \ = \ \vec{C}^n \ ).$$

with

$$C_i \quad = \quad \begin{array}{l}\text{the expression associated with } X_i \text{ where all proper} \\ \text{subexpressions are replaced by their associated va-} \\ \text{riables and } X \text{ is replaced by } X_1,\end{array}$$

assuming that X_1 is the variable associated with B. By Bekič's theorem B^2 is equivalent to the original assertion B^1 and we replace B^1 by B^2 in A^1.

Notice, that the introduction of the new $|B|$ variables at most doubles the size of the assertion and does not change the alternation depth.

This is easily generalized to the case where the fixed-point has arity higher than one.

Repeat the above transformation on all `where`-clauses in A^1 and call the resulting assertion A'. This does not increase the alternation depth and at most increases the size of the assertion with a factor of two.

These simple steps can easily be performed in linear time, and the resulting assertion is semantically equivalent with A since each step is semantics preserving and has $ad(A') = \max\{1, ad(A)\}$, size $O(|A|)$, and a number of variables which is at most $|A|$. $\Box$

As an example the assertion $(X \wedge \langle a\rangle X \ \texttt{where}_\mu \ X = [a]X \wedge T)$ will give rise to the simple assertion

$$X_1 \ \texttt{where}_\mu \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{pmatrix} = \begin{pmatrix} X_2 \wedge X_3 \\ \langle a\rangle X_3 \\ X_4 \wedge X_5 \\ [a]X_3 \\ T \end{pmatrix}$$

Lemma 3.6 *If A is a simple closed assertion in the standard calculus then for any state s of a transition system T, and any change of variables σ, the alternation depth of* $\mathrm{red}_{\times s}(A;\sigma)$ *is at most ad(A) and the size of* $\mathrm{red}_{\times s}(A;\sigma)$ *is $O(|A||T|)$.*

Proof: By inspecting figure 3.9 it is easily observed that the alternation depth is not increased, but might be decreased as dependencies between fixed-points can disappear as a result of dividing out a modality. Moreover, since A is simple, all of A is inside `where`-clauses, with simple assertions and equation systems. The total size of dividing a simple form with each state of a transition system is bounded by $|T|$ as is easily seen from the following calculations. Assume that σ is a change of variables with $\sigma(X) = IN(X_1, \ldots, X_n)$.

$$|\mathrm{red}_{\times \vec{s}}(\langle b \times a\rangle X)| \;=\; |(\langle b\rangle \bigvee_{s';s_1 \xrightarrow{a} s'} X_{s'}, \ldots, \langle b\rangle \bigvee_{s';s_n \xrightarrow{a} s'} X_{s'})|$$

$$\begin{aligned} &= \sum_{i=1}^{n}(1+\max\{1,|s_i \xrightarrow{a} |\}) \\ &\leq 2+|\rightarrow| \leq 2+|T| \end{aligned}$$

Hence, as there are at most $O(|A|)$ variables, each of which gives rise to $|S|$ new variables with a total size of the right-hand sides bounded by $2|T|$, we get the bound $O(|A||T|)$. □

If the assertion is not simple before the division takes place the above bound does not hold. To see why, consider the assertion $\mu X.\langle b\times a\rangle[b\times a]\ldots\langle b\times a\rangle[b\times a]X$ (l modalities), and assume that T is a transition system with n states, which all have a-transitions to all other states including themselves. Then the size of a single right-hand side of the resulting assertion will be:

$$\begin{aligned} |\langle b\times a\rangle[b\times a]\ldots\langle b\times a\rangle[b\times a]X/s_j| &= |\langle b\rangle \bigvee_{i_1}[b]\bigwedge_{i_2}\ldots\langle b\rangle\bigvee_{i_{l-1}}[b]\bigwedge_{i_l}X_{i-l}| \\ &= n^l, \end{aligned}$$

where all indices range over all states. The significance of making the fixed-points simple is precisely that values of subexpressions are shared across the disjunctions and conjunctions avoiding unnecessary repetitions. In this example, we will get a resulting assertion of size ln^2 (n^2 transitions) – instead of the above n^l.

3.8 Bibliographic Notes

As mentioned in the introductory chapter the search for compositional verification methods is one of the major challenges to the verification community. We have in this chapter presented a method which can be characterized as *de-compositional* in the sense that the task of verifying an assertion for a composite process is decomposed into verification tasks for the subprocesses. The method described is based on previous work by Winskel [89, 93, 94]; the main difference is in the presence of the propositional language $\mathcal{L}$ allowing for much more compact reductions, the treatment of fixed-points (which is along the lines of Winskel [94]) and the reductions for recursion and product which are new. We shall in later chapters see some applications of the reductions.

Larsen and Xinxin [57] describe a method which is compositional by using an 'operational semantics of contexts'. Whereas their method depends on the operational semantics, ours is driven by the syntax of the processes. Incidentally their 'decompositional rule' for the product turns out to be very similar to the operational reduction for product, although they lack the ability to get compact assertions as offered by the 'sharing across products' made possible by the `where`-construction.

Another approach to compositionality can be found in the compositional proof systems of Stirling [77, 76]. In the context of CTL* several heuristic methods have been described. For instance, Clarke, Long and McMillan [22] suggest using a notion of 'interface processes' that model the environment of a process in a concurrent composition. These interface processes, often simpler than the full environment, can be composed with the process and properties of the global system can be shown by reasoning locally about the process composed with its interface process. It would be interesting to find out to what extent this idea could be combined with the ideas of reductions; there seems to be no immediate conflict, but the benefits of combining the two approaches are not clear.

Chapter 4

Expressing Properties in the Logic

The idea of considering process algebras as models of concurrent systems has been well-studied in the last 10 years and by now numerous process algebras based on different intuitions on communication and concurrency primitives exist. There are well-developed equational theories, many results on decidability and undecidability of equivalences between processes and algorithms for determining equivalences. The relationship between the process algebraic models of concurrent computations and other models have been studied in great detail, and by now it is fair to say that the process algebraic approach has been quite successful in achieving results of a profound and universal character.

However, until recently the practical side of applying process algebras to *concrete* problems has received very little attention. Although the literature is full of examples they mostly have the flavour of being toy-examples to illustrate specific points of the method or theory being discussed – this thesis being no exception. The growing number of tools for performing various of the verification tasks either by means of algorithms performing the verification tasks automatically or as verification assistants, makes the promise of larger, realistic examples being performed.

When using the process algebraic approach in building concrete models of concurrent systems much help can be found in the increasing number of textbooks covering the area, but when it comes to writing specifications in the modal μ-calculus considerably less can be found. We give a small guide on how to express a rather generic set of properties which seems to be of general applicability by providing a set of '*macros*', i.e. some abbreviations of modal μ-calculus formulas that can be combined and actually viewed as forming a little, perhaps more comprehensible, language on its own.

This guide is by no means exhaustive and should not be taken as a complete survey of the known results on translating other temporal logics into the modal μ-calculus. For such results the reader is referred to Emerson and Clarke [34], Kozen [49], Emerson and Lei [35], Dam [31], and Stirling [79]. In fact, many of the results given in this section are well-known. However, the use of the extended calculus for expressing behavioural relations, allowing for the compositional method and the algorithms to be applied, is new.

4.1 Motivation

As the reader may have realized by now it can be difficult to express properties in the modal μ-calculus, i.e. to write down an assertion in the logic expressing the property of interest. The expressiveness results giving translations from various perhaps more easily accessible temporal logics offer one way of attack on this problem: Express the property in your favorite temporal logic and use the translation at hand. This appealing approach has, however, some disadvantages. First, the assertions tend to be extremely complicated, because of the blow-up in size caused by the translation. Second, these 'automatically generated' assertions are often quite unreadable which makes further verification difficult and small adjustments almost impossible. Third, they do not offer much insight into the modal μ-calculus and do not necessarily exploit the full capabilities of the logic. Finally, we have introduced some extensions to the logic which will only be fully exploited by working in the logic itself.

We first review in section 4.2 some results from the work on propositional dynamic logic (see for example Fischer and Ladner [38] and Emerson and Lei [35]) on how commonly used basic temporal constructions can be written as 'macros' that can be combined and used for expressing rather complicated properties, and then in section 4.3 take the unconventional view of using the logic as a meta-language for expressing equivalences and preorders, facilitating, through our reduction for product, the immediate construction of *characteristic formulae* characterizing equivalence classes and down- and upwards-closed subsets of preorders. We then in section 4.4.1 investigate in more general terms the more traditional problem of expressing linear and branching time temporal properties and give a collection of macros derived from a subset of the logic CTL*.

4.2 Basic Operators

We start with some very simple and useful operators allowing *regular expressions* of actions to be specified. First, however, we must introduce a little notation for sequences.

Definition 4.1 Let ϵ denote the empty sequence, and for a set Δ let $FinSeq(\Delta)$ be the set of *finite sequences* over Δ, i.e. the set

$$\{d_1 d_2 \dots d_n \mid n \in \omega, d_i \in \Delta\}$$

where we simply enumerate a sequence by juxtapositioning the elements. The *length* $|\delta|$ of a sequence δ is the number of elements in the sequence, i.e. $|d_1 d_2 \dots d_n| = n$. For two sequences $\delta_1 = d_1^1 \dots d_n^1$, $\delta_2 = d_1^2 \dots d_m^2$ in $FinSeq(\Delta)$ we denote their *concatenation* by juxtapositioning, i.e.

$$\delta_1 \delta_2 = d_1^1 \dots d_n^1 d_1^2 \dots d_m^2.$$

Let $InfSeq(\Delta)$ be the set of *infinite sequences* over Δ, i.e. the set

$$\{d_1 d_2 \dots d_i \dots \mid d_i \in \Delta\}$$

The length of an infinite sequence δ is ω. For $1 \leq i \leq |\delta|$ let δ_i be the i'th element of δ and define for $0 \leq i \leq |\delta|$ the i'th suffix δ^i of δ by

$$\begin{array}{rcl} \delta_0 & = & \delta \\ (d\delta)^{k+1} & = & \delta^k \end{array}$$

Let $Seq(\Delta) = FinSeq(\Delta) \cup InfSeq(\Delta)$ be the set of all sequences over Δ. We extend concatenation to all sequences by taking $\delta_1 \delta_2 = \delta_1$, if $|\delta_1| = \omega$. $\Box$

Example 4.1 (Regular expressions)
Regular expressions of sequences of actions R are formed from the following grammar:

$$R ::= * \mid a \mid R_0 R_1 \mid R_0 \cup R_1 \mid R^*$$

Here the idling action $*$ takes the role of the empty sequence of actions. We let a range over composite actions Act_*. A regular expression R denotes a set of sequences in $FinSeq(\Delta)$ defined by structural induction on R as follows:

$$\begin{array}{rcl} \| * \| & = & \{\epsilon\} \\ \|a\| & = & \{a\} \\ \|R_0 R_1\| & = & \{\delta_0\delta_1 \mid \delta_0 \in \|R_0\|, \delta_1 \in \|R_1\|\} \\ \|R_0 \cup R_1\| & = & \|R_0\| \cup \|R_1\| \\ \|R^*\| & = & \{\delta_0 \ldots \delta_n \mid n \in \omega, \delta_i \in \|R\|\} \end{array}$$

We now extend the diamond modality $\langle\ \rangle$ to regular expressions as follows:

$$\begin{array}{rcl} \langle * \rangle A & = & \langle * \rangle A \\ \langle a \rangle A & = & \langle a \rangle A \\ \langle R_0 R_1 \rangle A & = & \langle R_0 \rangle (\langle R_1 \rangle A) \\ \langle R_0 \cup R_1 \rangle A & = & \langle R_0 \rangle A \vee \langle R-1 \rangle A \\ \langle R^* \rangle A & = & \mu X. \langle R \rangle X \vee A, \text{ for some } X \notin FV(A) \end{array}$$

□

Given a transition relation $\rightarrow$, we can now extend it to sequences $\delta = a_1 \ldots a_n$ by taking

$$p \xrightarrow{\delta} p' \Leftrightarrow_{\text{def}} \exists p_0, \ldots, p_n . p = p_0 \xrightarrow{a_1} p_1 \ldots p_{n-1} \xrightarrow{a_n} p_n = p'$$

and to regular expressions R by

$$p \xrightarrow{R} p' \Leftrightarrow_{\text{def}} \exists \delta \in \|R\| \ . \ p \xrightarrow{\delta} p'$$

This means for instance, that using the convention that . abbreviates *Act* and allowing this as a regular expression with the obvious semantics, we could redefine the set of states reachable from p of monotype $\cdot$ as

$$R_p = \{p' \mid p \xrightarrow{.^*} p'\}.$$

In examples we will actually use any set of actions as regular expression with the obvious semantics.

Now, for the extension of the transition relation to regular expressions we can show:

Lemma 4.1 (Adequacy for $\xrightarrow{R}$) *Assume A is a closed assertion and T a transition system with states S. For all $s \in S$,*

$$s \models \langle R \rangle A \text{ if and only if } \exists s'.\ s \xrightarrow{R} s' \ \&\ s' \models A.$$

Proof: By structural induction on R. The interesting case is $R \equiv R'^*$, which amounts to showing that $\mu X.\langle R' \rangle X \vee A$ actually denotes a set of states which in a *finite* number of steps of transitions from R' ends up in a state satisfying A. To show this define $f : \mathcal{P}(S) \to \mathcal{P}(S)$ by

$$f(U) = [\![\langle R' \rangle X \vee A]\!]_T\ \rho[U/X]\phi$$

for some ρ and ϕ, and let $M = \{s \in S \mid \exists s'.\ s \xrightarrow{R'^*} s'\ \&\ s' \in [\![A]\!]_T \rho\ \phi\}$. We will argue that $\mu f = M$. First, notice that by definition of $\xrightarrow{R'^*}$ we have

$$\begin{aligned} M &= \{s \mid \exists n.\exists s_0, \ldots, s_n.s = s_0 \xrightarrow{R'} s_1 \xrightarrow{R'} s_2 \ldots \\ &\quad \ldots s_{n-1} \xrightarrow{R'} s_n \ \&\ s_n \in [\![A]\!]_T \rho\ \phi\}, \end{aligned} \tag{4.1}$$

hence

$$\begin{aligned} f(M) &= [\![\langle R' \rangle X \vee A]\!]_T \rho[M/X]\ \phi \\ &= \{p \mid \exists p'.p \xrightarrow{R'} p'\ \&\ p' \in M\} \cup [\![A]\!]_{\rho[M/X]\ \phi} \\ &\qquad \text{by definition} \\ &\subseteq M \\ &\qquad \text{by (4.1)} \end{aligned}$$

Therefore as μf is the least post-fixed point of f, we have $\mu f \subseteq M$. For the other direction we argue by induction on $n \in \omega$ that if $\exists s_0, \ldots, s_n.s_0 \xrightarrow{R'} s_1 \xrightarrow{R'} s_2 \ldots s_{n-1} \xrightarrow{R'} s_n\ \&\ s_n \in [\![A]\!]_T \rho\ \phi$ then $s_0 \in f^{n+1}(\emptyset) \subseteq \mu f$. For $n = 0$ this follows from $f(\emptyset) = [\![A]\!]_T \rho[\emptyset/X]\ \phi$ which equals $[\![A]\!]_T \rho\ \phi$ as $X \notin FV(A) = \emptyset$. The inductive step is just as simple. $\square$

This is called an *adequacy result* because it shows that the logic is adequate for expressing the operational behaviour given by $\xrightarrow{R}$. Notice, that the same result for $\xrightarrow{a}$ follows directly from the semantics of the assertion $\langle a \rangle A$.[1]

Example 4.2 (ω-regular expressions) (Park [69]) We can extend the above mentioned regular expressions to *ω-regular expressions* by allowing the regular expressions R^ω and extending the semantics to infinite sequences, defining

$$[\![R^\omega]\!] = \{s_1 \dots s_i \dots \mid \forall i \in \omega.\ s_i \in [\![R]\!]\},$$

where concatenating an infinite s with an arbitrary s' is simply $ss' = s$. For the extended diamond modality we take

$$\langle R^\omega \rangle A = \nu X.\langle R \rangle X$$

which is independent of A as A was intended to hold for a state reached *after* a performing a sequence of actions in R, and such a state will never be reached. An extension of the adequacy lemma also holds for ω-regular expressions implying that for an ω-free expression R we have

$$s \models \langle R^\omega \rangle A \text{ iff } \exists s_0, \dots, s_i, \dots\ \forall i \in \omega.\ s_i \xrightarrow{R} s_{i+1}.$$

□

Theorem 4.1 (Adequacy for ω-regular expressions) *Assume A is a closed assertion of type τ, T a transition system with states S and R an ω-regular expression of type τ. For all $s \in S$,*

$$\begin{gathered} s \models \langle R \rangle A \\ \textit{iff} \\ \exists \delta \in \mathit{FinSeq}([\![\tau]\!]) \cap \|R\|, s' \in S.\ s \xrightarrow{\delta} s' \ \&\ s' \models A \\ \textit{or} \\ \exists \delta \in \mathit{InfSeq}([\![\tau]\!]) \cap \|R\|. s \xrightarrow{\delta} \end{gathered}$$

Proof: See appendix B.1. □

[1] A stronger property one might imagine is if the logic is capable of expressing p' as an assertion $A_{p'}$ s.t. $p \xrightarrow{R} p'$ iff $p \models \langle R \rangle A_{p'}$. We will later see how to do that for finite-state processes – at least up to strong bisimulation equivalence.

4.3 Equivalences and Preorders

This section will describe a rather novel use of the modal μ-calculus as a *meta-language* for describing equivalences and preorders. The advantage of this approach is that having expressed an equivalence or a preorder as a formula in the logic, all the techniques and algorithms developed for the logic become directly applicable for that equivalence or preorder. We will see how familiar techniques of showing processes equivalent can be re-discovered as special cases of general techniques for the μ-calculus, and how new techniques emerges.

But first we consider some very familiar equivalences due to Milner [59].

Example 4.3 (Strong bisimilarity) (Milner [59, p.88ff]) Recall that two processes p and q are *strongly bisimilar* written $p \sim q$, if and only if, (p, q) belongs to the maximum fixed-point of the function F on $\mathcal{P}(S_p \times S_q)$ defined by $(p, q) \in F(R) \Leftrightarrow_{\mathsf{def}} \forall a \in Act.$

$$\begin{array}{rl} (i) & \forall p'.\ p \xrightarrow{a} p' \Rightarrow \exists q'.\ q \xrightarrow{a} q' \ \&\ (p', q') \in R \\ (ii) & \forall q'.\ q \xrightarrow{a} q' \Rightarrow \exists p'.\ p \xrightarrow{a} p' \ \&\ (p', q') \in R \end{array}$$

Now, notice that $F(R)$ can be expressed quite directly as

$$\tilde{\forall}\alpha.[\alpha \times *]\langle * \times \alpha\rangle R \wedge [* \times \alpha]\langle \alpha \times *\rangle R$$

hence the μ-calculus version of $\sim$ becomes

$$\mathcal{B} =_{\mathsf{def}} \nu R.\tilde{\forall}\alpha.[\alpha \times *]\langle * \times \alpha\rangle R \wedge [* \times \alpha]\langle \alpha \times *\rangle R$$

Notice, that $ad(\mathcal{B}) = 1$. $\square$

It is now straightforward to prove:

Proposition 4.1 *For processes p and q of type $\cdot$, $p \sim q$, if and only if, $p \times q \models \mathcal{B}$.*

Example 4.4 (Weak bisimilarity and observation congruence)
For *weak bisimulation* (also called *observation equivalence* Milner [59, p.108ff]) we need to introduce the notion of *weak* transitions capturing essentially what is 'visible' ignoring the 'invisible/internal' action τ. Following our previous discussion on regular expressions in example 4.1, we can simply define

$$\begin{array}{rcl} \overset{a}{\Rightarrow} & = & \overset{\tau^*a\tau^*}{\longrightarrow}, \quad \text{if } a \neq \tau \\ \overset{\tau}{\Rightarrow} & = & \overset{\tau^*}{\longrightarrow} \end{array}$$

Two processes p and q are *weakly bisimilar* written $p \approx q$, if and only if, (p, q) belongs to the maximum fixed-point of the function F on $\mathcal{P}(S_p \times S_q)$ defined by $(p, q) \in F(R) \Leftrightarrow_{\mathsf{def}} \forall a \in Act$.

$$\begin{array}{ll} (i) & \forall p' \cdot\ p \xrightarrow{a} p' \Rightarrow \exists q' \cdot\ q \overset{a}{\Rightarrow} q' \ \&\ (p', q') \in R \\ (ii) & \forall q' \cdot\ q \xrightarrow{a} q' \Rightarrow \exists p' \cdot\ p \overset{a}{\Rightarrow} p' \ \&\ (p', q') \in R \end{array}$$

Now, to express this as an assertion we need to introduce *weak modalities* of type $\cdot \times \cdot$, two versions allowing the left respectively the right component to make a weak transition while the other idles. Following the lines of example 4.1, we first define two more general constructions allowing the left and right component to perform transitions of arbitrary regular expressions. Define for an assertion A of type $\cdot \times \cdot$, and R of type $\cdot$, $\langle R\rangle_l A$ inductively as follows:

$$\begin{array}{rcl} \langle \alpha\rangle_l A & = & \langle \alpha \times *\rangle A \\ \langle R_0R_1\rangle_l A & = & \langle R_0\rangle_l(\langle R_1\rangle_l A) \\ \langle R_0 \cup R_1\rangle_l A & = & \langle R_0\rangle_l A \vee \langle R_1\rangle_l A \\ \langle R^*\rangle_l A & = & \mu X.\langle R\rangle_l X \vee A, \quad \text{for some } X \notin FV(A) \end{array}$$

and analogously for $\langle R\rangle_r A$ with the first clause changed to

$$\langle \alpha\rangle_r A = \langle * \times \alpha\rangle A.$$

We can now define a weak diamond left and weak diamond right modality:

$$\begin{array}{rcl} \langle\langle \alpha\rangle\rangle_l A & = & \langle \tau^*\alpha\tau^*\rangle_l A \vee (\alpha = \tau \wedge A) \\ \langle\langle \alpha\rangle\rangle_r A & = & \langle \tau^*\alpha\tau^*\rangle_r A \vee (\alpha = \tau \wedge A). \end{array}$$

The first assertion is satisfied by a product $p \times q$ if $p \overset{\alpha}{\Rightarrow} p'$ and $p' \times q$ satisfies A. Using this we can express $\approx$ as the assertion

$$\mathcal{W} =_{\mathsf{def}} \nu R.\tilde{\forall}\alpha.[\alpha \times *]\langle\langle \alpha\rangle\rangle_r R \wedge [* \times \alpha]\langle\langle \alpha\rangle\rangle_l R$$

For *observation congruence* (written '=' Milner [59, p.153]) we take

$$\mathcal{C} =_{\mathsf{def}} \tilde{\forall}\alpha.[\alpha \times *]\langle \tau^*\alpha\tau^*\rangle_r \mathcal{W} \wedge [* \times \alpha]\langle \tau^*\alpha\tau^*\rangle_l \mathcal{W}$$

Notice, that $ad(\mathcal{W}) = ad(\mathcal{C}) = 2$. □

It is again straightforward to show:

Proposition 4.2 *For processes p and q, $p \approx q$, if and only if, $p \times q \models \mathcal{W}$, and $p = q$, if and only if $p \times q \models \mathcal{C}$.*

Example 4.5 (Ready simulation) (Bloom [13], Bloom and Paige [14]) A state p is said to *ready simulate* a state q if (p, q) belongs to the maximal fixed-point of the function F on $S_p \times S_q$ defined by $(p, q) \in F(R) \Leftrightarrow_{\mathsf{def}}$

$$\begin{array}{rl} (i) & \forall a \in Act.\ \forall p'.\ p \xrightarrow{a} p' \Rightarrow \exists q'.\ q \xrightarrow{a} q' \ \&\ (p', q') \in R \\ (ii) & \forall a \in Act.\ p \xrightarrow{a} \Leftrightarrow q \xrightarrow{a}. \end{array}$$

This is captured by the assertion

$$\mathcal{R} = \nu R.\forall \alpha.[\alpha \times *]\langle * \times \alpha\rangle R \wedge (\langle \alpha \times *\rangle T \leftrightarrow \langle * \times \alpha\rangle T).$$

Ready bisimulation (also called 2/3-bisimulation) is simply the conjunction of $\mathcal{R}$ and its reverse $\mathcal{R}^{-1}$ defined by transforming $\alpha \times *$ (respectively $* \times \alpha$) to $* \times \alpha$ (respectively $\alpha \times *$) in the definition of $\mathcal{R}$. Hence $\mathcal{RB} = \mathcal{R} \wedge \mathcal{R}^{-1}$.

Notice that $ad(\mathcal{RB}) = ad(\mathcal{R}) = 1$. □

Example 4.6 (Simulation preorder)
As an example of a preorder we consider the *simulation* relation $\prec$ defined by Milner [59, p.208], as the maximum fixed-point of the function F on $\mathcal{P}(S \times S')$ defined by $(p, q) \in F(R) \Leftrightarrow_{\mathsf{def}}$

$$\forall a \in Act.\ \forall p'.\ p \overset{a}{\Rightarrow} p' \Rightarrow \exists q'.\ q \overset{a}{\Rightarrow} q' \ \&\ (p', q') \in R$$

Take

$$\mathcal{S} =_{\mathsf{def}} \nu R.\tilde{\forall} \alpha [[\alpha]]_l \langle\langle \alpha \rangle\rangle_r R.$$

Once again we can easily show $p \prec q$ if and only if $p \times q \models S$ and observe that $ad(\mathcal{S}) = 2$. □

Example 4.7 (Prebisimulation)
To take proper care of the possible divergent behaviour of processes coming

from for instance infinite 'internal chatter' manifesting itself as an infinite sequence of silent actions, transition systems with a divergence predicate has been studied, and a preorder called *prebisimulation* defined. This can also be expressed within the modal μ-calculus as an assertion. First, a *transition system with divergence* is a tuple $(S, i, L, \rightarrow, \uparrow)$ where $(S, i, L, \rightarrow)$ is a normal transition system and $\uparrow \subseteq S$ is a subset of divergent states. The divergence set $\uparrow$ is often thought of as a predicate using $s \uparrow$ for $s \in \uparrow$. We denote the complement of $\uparrow$ by $\downarrow$, i.e. $\downarrow = S \setminus \uparrow$.

To express this in the logic we assume the presence of a constant $Conv$ with valuation $V(Conv) = \downarrow$, and constants $Conv \times T$ and $T \times Conv$ of type $\cdot \times \cdot$ with valuations $V(Conv \times T) = \downarrow \times S$ and $V(T \times Conv) = S \times \downarrow$. Now, the *prebisimulation preorder* $\sqsubseteq$ is the greatest fixed-point of the function F on $\mathcal{P}(S \times S)$ defined by $(p, q) \in F(R) \Leftrightarrow_{\mathsf{def}} \forall a \in Act.$

$$\begin{array}{rl} (i) & \forall p'.\ p \xrightarrow{a} p' \Rightarrow \exists q'.\ q \xrightarrow{a} q' \ \&\ (p', q') \in R \\ (ii) & p \downarrow \Rightarrow q \downarrow \ \&\ (\forall q'.\ q \xrightarrow{a} q' \Leftrightarrow \exists p'.\ p \xrightarrow{a} p' \ \&\ (p', q') \in R). \end{array}$$

(Note that if $\downarrow = S$ then this definition degenerates to the usual definition of bisimulation.) In the logic this becomes

$$\nu R. \tilde{\forall} \alpha. [\alpha \times *] \langle * \times \alpha \rangle R \wedge ((Conv \times T) \rightarrow ((T \times Conv) \wedge [* \times \alpha] \langle \alpha \times * \rangle R))$$

with $ad = 1$. Again it is very easy to construct a weak version of this with $ad = 2$.[2] $\Box$

Example 4.8 (Characteristic formulae)
Through the product reduction, we can achieve characteristic formulae with respect to any equivalence, or preorder described as a μ-calculus formula. As an example, we consider the simple buffer

$$\begin{tikzcd} P \arrow[r, shift left, "in?"] & P' \arrow[l, shift left, "out!"] \end{tikzcd}$$

defined by

[2]The divergence predicate is sometimes chosen to be the set of states that might perform an infinite sequence of τ's. This could be expressed internally in the logic as $\langle \tau^\omega \rangle T = \nu X. \langle \tau \rangle X$.

$$\begin{array}{rcl} P & = & in?P' \\ P' & = & out!P \end{array}$$

and find a formula which is satisfied precisely by processes which are strongly bisimilar to the buffer:

$$\begin{array}{ll} \multicolumn{2}{l}{\mathrm{red}_{\times P}(\mathcal{B})} \\ = & (R_P\ \texttt{where}_\nu \begin{pmatrix} R_P \\ R_{P'} \end{pmatrix} = \begin{pmatrix} \tilde{\forall}\alpha.[\alpha](\alpha = in? \wedge R_{P'}) \wedge (\alpha = in? \rightarrow \langle\alpha\rangle R_{P'}) \\ \tilde{\forall}\alpha.[\alpha](\alpha = out! \wedge R_P) \wedge (\alpha = out! \rightarrow \langle\alpha\rangle R_P) \end{pmatrix}) \\ = & (R_P\ \texttt{where}_\nu \begin{pmatrix} R_P \\ R_{P'} \end{pmatrix} = \begin{pmatrix} (\tilde{\forall}\alpha.[\alpha](\alpha = in? \wedge R_{P'})) \wedge \langle in?\rangle R_{P'} \\ (\tilde{\forall}\alpha.[\alpha](\alpha = out! \wedge R_P)) \wedge \langle out!\rangle R_P \end{pmatrix}) \\ & \quad \text{by distributing } \tilde{\forall} \text{ over the conjunctions.} \end{array}$$

For weak bisimulation we first observe that for all s

$$(\langle\langle\alpha\rangle\rangle_l A/s = \langle\langle\alpha\rangle\rangle(A/s),$$

where

$$\langle\langle\alpha\rangle\rangle A = \langle\tau^*\alpha\tau^*\rangle A \vee (\alpha = \tau \wedge A),$$

and dually for $[[\alpha]]_l$:

$$([[\alpha]]_l A)/s = [[\alpha]](A/s),$$

where

$$[[\alpha]]A = [\tau^*\alpha\tau^*]A \wedge (\alpha = \tau \rightarrow A).$$

(As a sideremark we notice that $\langle\langle a\rangle\rangle$ has the following property:

$$\models s : \langle\langle a\rangle\rangle A \Leftrightarrow \exists s'.\ s \overset{a}{\Rightarrow} s' \ \&\ \models s' : A.$$

Hence the logic is also adequate for $\overset{a}{\Rightarrow}$.)

Returning to our example we observe that

$$(\langle\langle\alpha\rangle\rangle_r A)/P = (\alpha = in?) \wedge (A/P'),$$

and

$$(\langle\langle\alpha\rangle\rangle_r A)/P' = (\alpha = out!) \wedge (A/P).$$

Hence,

$$\mathcal{W}/p = (R_P \;\texttt{where}_\nu \begin{pmatrix} R_P \\ R_{P'} \end{pmatrix} = \begin{pmatrix} \tilde{\forall}\alpha.[\alpha](\alpha = in? \wedge R_{P'})) \wedge \langle\langle in?\rangle\rangle R_{P'} \\ (\tilde{\forall}\alpha.[\alpha](\alpha = out! \wedge R_P)) \wedge \langle\langle out!\rangle\rangle R_P \end{pmatrix})$$

□

4.4 General Temporal properties

Temporal properties - and temporal logics - are normally classified as relating to *linear time* or *branching time* dependent upon whether they express properties about paths of transitions (often also called runs) or trees of transitions. Arguments exists for and against both views, as well as arguments for coexistence of the views (see f.ex. Lamport [52] and Emerson and Halpern [36]). We take the rather pragmatic approach that both classes are useful, and show how some of each class are expressible within the logic. At first sight, the modal μ-calculus is very much a branching time logic. The semantics is in terms of transition systems, which can be thought of as compact representations of trees of transitions, and the modalities $\langle\alpha\rangle$ and $[\alpha]$ exploits the branching structure of the underlying models by quantifying over 'various future states'. However, as the translations from other logics – including logics with linear time features – shows, it is possible to mimic the linear time aspect.

4.4.1 A Linear Time Logic

The modal μ-calculus is classified as a *branching time logic* referring to the property that the modalities $\langle a\rangle$ and $[a]$ *quantify over possible futures*. Nevertheless, expressivity results show that also *linear time logics* referring in a certain sense to only one possible future, can be embedded into the modal μ-calculus ([49, 79, 30]). We will consider a very succinct embedding of a simple logic combining linear and branching time features, which apart from the usefulness of the translation will supply us with a collection of derived operators (or 'macros').

The logic will be a sub-logic of the logic CTL* (*Computation Tree Logic*) [36]. We first, however, describe CTL* in a slightly weaker version by omitting the 'until'-operator $\mathcal{U}$, instead having explicit versions of the 'eventually'- and 'always'-operators $\Diamond$ and $\Box$. The syntax of this mini-CTL* is given by the grammar:

$$\Phi ::= p \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \bigcirc' \; \Phi \mid \bigcirc' \; \Phi \mid \Box\Phi \mid \Diamond\Phi \mid \exists\Phi \mid \forall\Phi$$

where p is are a propositional constant, $\bigcirc$ a 'next'-operator, $\bigcirc'$ a 'weak next'-operator and $\exists$ and $\forall$ are *path quantifiers*. We use greek capital letters $\Phi, \Theta, \ldots$ to range over linear time formulae. A commonly used abbreviation is $\leadsto$ ('leads to'):

$$\Phi \leadsto \Phi' = \Box(\Phi \rightarrow \Diamond\Phi')$$

Here we have taken the liberty of using $\Phi \rightarrow \Phi'$ as an abbreviation for $\neg\Phi \vee \Phi'$. Although strictly speaking $\neg\Phi$ is not part of the syntax, we consider $\neg$ to be a syntactic operation dualizing every operator in Φ ($\bigcirc$ and $\bigcirc'$ respectively $\Diamond$ and $\Box$ being dual), assuming that every propositional constant has an associated negated version.

Semantics of linear time logics is given with respect to *runs* through a transition system, i.e. as finite and infinite *completed* sequences of transitions.

Definition 4.2 Given a transition system $T = (S, i, L, \rightarrow)$ of type $\cdot$. For a transition $(s, a, s') \in \rightarrow$ define $pre(s, a, s') = s$, $post(s, a, s') = s'$, $lbl(s, a, s') = a$. The set of *(completed) runs* $Runs_s$ through T starting at $s \in S$ is defined as the set of pairs (s, δ) where $\delta \in Seq(\rightarrow)$ is a sequence such that

$$\begin{array}{rl}
(i) & |\delta| > 0 \Rightarrow pre(\delta_0) = s \\
(ii) & \forall 1 \leq i < |\delta|.\; post(\delta_i) = pre(\delta_{i+1}) \\
(iii) & |\delta| < \omega \wedge post(\delta_{|\delta|}) \xrightarrow{a} \Rightarrow \; a = * \\
(iv) & lbl(\delta_i) \neq *
\end{array}$$

Let $Runs = \bigcup_{s \in S} Runs_s$ be the set of all completed runs through T. For a run (s, δ) let $init(s, \delta) = s$. $\Box$

The length of a run is the length of the underlying sequences, and the i'th element and i'th suffix of a run is the i'th element and i'th suffix of the underlying sequence (with an initial state added for the suffix).

As adjacent transitions in a run are required to match by (*ii*) we can think of a run as an alternating sequences of states and actions, i.e.

$$s_0 a_1 s_1 \ldots a_n s_n$$

for the run $(s_0, (s_0, a_1, s_1)(s_1, a_2, s_2) \ldots (s_{n-1}, a_n, s_n))$.

To summarize: *Completed runs are finite or infinite sequences of non-idling transitions with a final state (if any) that cannot perform any non-idling actions.*

Given a valuation V giving meaning to the propositional constants p as sets $V(p)$ of states, the semantics $\|\Phi\|_{T,V} \subseteq \mathit{Runs}$ is defined by structural induction on Φ as follows (omitting subscripts T and V for brevity):

$$\begin{array}{rcl}
\|p\| & = & \bigcup_{s \in V(p)} \mathit{Runs}_s \\
\|\Phi_0 \vee \Phi_1\| & = & \|\Phi_0\| \cup \|\Phi_1\| \\
\|\Phi_0 \wedge \Phi_1\| & = & \|\Phi_0\| \cap \|\Phi_1\| \\
\|\bigcirc' \Phi\| & = & \{\delta \in \mathit{Runs} \mid 1 \leq |\delta|, \delta^1 \in \|\Phi\|\} \\
\|\bigcirc' \Phi\| & = & \{\delta \in \mathit{Runs} \mid |\delta| = 0 \text{ or } \delta^k \in \|\Phi\|\} \\
\|\Box \Phi\| & = & \{\delta \in \mathit{Runs} \mid \forall k. |\delta| < k \text{ or } \delta^k \in \|\Phi\|\} \\
\|\Diamond \Phi\| & = & \{\delta \in \mathit{Runs} \mid \exists k \leq |\delta|.\ \delta^k \in \|\Phi\|\} \\
\|\exists \Phi\| & = & \{\delta \in \mathit{Runs} \mid \exists \delta' \in \mathit{Runs}_{init(\delta)}.\ \delta' \in \|\Phi\|\} \\
\|\forall \Phi\| & = & \{\delta \in \mathit{Runs} \mid \forall \delta' \in \mathit{Runs}_{init(\delta)}.\ \delta' \in \|\Phi\|\}
\end{array}$$

To compare this with formulas of the modal μ-calculus we need to define what it means for a state to satisfy a linear time formula. Here we have two possibilities:

$$s \models^{\exists} \Phi \Leftrightarrow_{\text{def}} \exists \delta \in \mathit{Runs}_s . \delta \in \|\Phi\|$$

or

$$s \models^{\forall} \Phi \Leftrightarrow_{\text{def}} \forall \delta \in \mathit{Runs}_s . \delta \in \|\Phi\|$$

Notice, however, that for a formula with a path quantifier at the top-level the two definitions coincide as satisfaction of a path-quantified formula only depends on the first state of a run. We will refer to such formulae as *state formulae* and as a set of runs U with the property that if δ is a path in U

then all paths with the same initial state as δ also belongs to U, as having *the state proverty*. Formally:

Definition 4.3 A set $U \subseteq \mathit{Runs}$ has the *state property* if

$$\delta \in U \Rightarrow (\forall \delta'.\mathit{init}(\delta) = \mathit{init}(\delta') \Rightarrow \delta' \in U.)$$

A formula Φ is a *state formula* if for all T and V, $\|\Phi\|_{T,V}$ has the state property. $\Box$

For a state formulae Ψ we define

$$s \models \Psi \Leftrightarrow_{\mathrm{def}} s \models^{\exists} \Psi (\Leftrightarrow s \models^{\forall} \Psi)$$

and we define

$$\begin{array}{rcl} [\![\Psi]\!] & =_{\mathrm{def}} & \{s \in S \mid \exists \delta \in \mathit{Runs}_s.\ \delta \in \|\Psi\|\} \\ & = & \{s \in S \mid s \models \Psi\} \end{array}$$

The translation of the full logic into the modal μ-calculus is quite involved due to the presence of assertions like: $\exists(\Diamond\Phi_0 \wedge \Diamond\Box\Phi_1 \wedge \Box\Phi_2\ )$ which require various patterns of behaviour to hold for the *same* path. Dam [31] describes one such translation. We consider instead a fragment of the logic – which we will refer to as CTL$^\bullet$ (pronounced 'CTL-dot') – which at certain places requires the modalities to be under the scope of path quantifiers:

$$\begin{array}{rcl} \Psi & ::= & p \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \exists\bigcirc\Psi \mid \\ & & \forall\bigcirc\Psi \mid \exists\bigcirc'\Psi \mid \forall\bigcirc'\Psi \mid \exists\Box\Phi \mid \forall\Box\Psi \mid \exists\Diamond\Psi \mid \forall\Diamond\Psi \mid \exists\Box\Diamond\Psi \mid \forall\Diamond\Box\Psi \end{array}$$

This fragment is actually less restrictive than one might imagine at first sight. Using the equivalences of figure 4.4.1, many more (although far from all) assertions can be transformed into CTL$^\bullet$.

The embedding of a CTL$^\bullet$ formula Ψ into the modal μ-calculus will be given as a function I defined recursively on the structure of Ψ. To describe the embedding we need a weak version of the diamond-modality $\langle . \rangle'$ defined by $\langle . \rangle' A = \langle . \rangle A \vee [.] F$, and a strong version of the box-modality $[.]'$ defined by $[.]' A = [.] A \wedge \langle . \rangle T$.

$$\begin{array}{rclrcl}
\Box\Box\Phi & = & \Box\Phi & \Diamond\Diamond\Phi & = & \Diamond\Phi \\
\Box\Diamond\Box\Phi & = & \Diamond\Box\Phi & \Diamond\Box\Diamond\Phi & = & \Box\Diamond\Phi \\
\bigcirc\Box\Phi & = & \Box\bigcirc\Phi & \bigcirc\Diamond\Phi & = & \Diamond\bigcirc\Phi \\
\forall\Box\Phi & = & \forall\Box\forall\Phi & \exists\Diamond\Phi & = & \exists\Diamond\exists\Phi \\
\forall\bigcirc\Phi & = & \forall\bigcirc\forall\Phi & \exists\bigcirc\Phi & = & \exists\bigcirc\exists\Phi \\
\Box(\Phi\wedge\Phi') & = & \Box\Phi\wedge\Box\Phi' & \Diamond(\Phi\vee\Phi') & = & \Diamond\Phi\vee\Diamond\Phi' \\
\Diamond\Box(\Phi\wedge\Phi') & = & \Diamond\Box\Phi\wedge\Diamond\Box\Phi' & \Box\Diamond(\Phi\vee\Phi') & = & \Box\Diamond\Phi\vee\Box\Diamond\Phi' \\
\bigcirc(\Phi\wedge\Phi') & = & \bigcirc\Phi\wedge\bigcirc\Phi' & \bigcirc(\Phi\vee\Phi') & = & \bigcirc\Phi\vee\bigcirc\Phi' \\
\forall(\Phi\wedge\Phi') & = & \forall\Phi\wedge\forall\Phi' & \exists(\Phi\vee\Phi') & = & \exists\Phi\vee\exists\Phi' \\
\forall(\Psi\vee\Phi) & = & \Psi\vee\forall\Phi & \exists(\Psi\wedge\Phi) & = & \Psi\wedge\exists\Phi \\
\forall\Psi & = & \Psi & \exists\Psi & = & \Psi
\end{array}$$

Figure 4.1: Some valid equivalences in CTL*. In the last two lines of clauses Ψ is a CTL$^\bullet$ formula. Equivalences involving the next operator also hold for the weak next operator.

$$\begin{array}{rcl}
I(p) & = & p \\
I(\Psi_0\vee\Psi_1) & = & I(\Psi_0)\vee I(\Psi_1) \\
I(\Psi_0\wedge\Psi_1) & = & I(\Psi_0)\wedge I(\Psi_1) \\
I(\exists\bigcirc\Psi) & = & \langle.\rangle I(\Psi) \\
I(\forall\bigcirc\Psi) & = & [.]'I(\Psi) \\
I(\exists\bigcirc'\Psi) & = & \langle.\rangle' I(\Psi) \\
I(\forall\bigcirc'\Psi) & = & [.]I(\Psi) \\
I(\exists\Box\Psi) & = & \nu X.\langle.\rangle'X\wedge I(\Psi) \\
I(\forall\Box\Psi) & = & \nu X.[.]X\wedge I(\Psi) \\
I(\exists\Diamond\Psi) & = & \mu X.\langle.\rangle X\vee I(\Psi) \\
I(\forall\Diamond\Psi) & = & \mu X.[.]'X\vee I(\Psi) \\
I(\exists\Box\Diamond\Psi) & = & \nu X.\mu Y.\langle.\rangle Y\vee(\langle.\rangle'X\wedge I(\Psi)) \\
I(\forall\Diamond\Box\Psi) & = & \mu X.\nu Y.[.]Y\wedge([.]'X\vee I(\Psi))
\end{array}$$

Notice, that I always yields closed assertions. Hence, the alternation depth of a translated formula $I(\Psi)$ can easily seen to be one – or two when any of the assertions $\exists\Box\Diamond$ and $\forall\Diamond\Box$ are present in Ψ. It is interesting to note that the 'dual' variations[3] $\forall\Box\Diamond$ and $\exists\Diamond\Box$ of $\exists\Box\Diamond$ and $\forall\Diamond\Box$, yield translations

$$I(\forall\Box\Diamond\Psi) = I(\forall\Box\forall\Diamond\Psi) = \nu X.[.]X\wedge(\mu Y.[.]'Y\vee I(\Psi))$$

and

[3]They are *not* dual in the technical sense

$$I(\exists\Diamond\Box\Psi) = I(\exists\Diamond\exists\Box\Psi) = \mu X.\langle . \rangle X \vee (\nu Y.\langle . \rangle' Y \wedge I(\Psi))$$

which are only of alternation depth one!

We can now prove:

Lemma 4.2 *For all $CTL^\bullet$ assertions Ψ and transition systems T with valuation V, we have*

$$[\![\Psi]\!]_{T,V} = [\![I(\Psi)]\!]_{T,V}$$

Proof: The proof is by structural induction on Ψ – the last two cases being a bit tricky, see appendix B.2. $\Box$

Operator	CTL*	Definition	Meaning
$EAlways(\alpha, A)$	$\exists\Box$	$\nu X.\langle\alpha\rangle' X \wedge A$	exists path on which always A
$AAlways(\alpha, A)$	$\forall\Box$	$\nu X.[\alpha]X \wedge A$	on all paths always A
$EEven(\alpha, A)$	$\exists\Diamond$	$\mu X.\langle\alpha\rangle X \vee A$	e.p.o.w. eventually A
$AEven(\alpha, A)$	$\forall\Diamond$	$\mu X.[\alpha]' X \vee A$	o.a.p. eventually A
$ERep(\alpha, A)$	$\exists\Box\Diamond$	$\nu X.\mu Y.\langle\alpha\rangle Y \vee (\langle\alpha\rangle' X \wedge A)$	e.p.o.w. repeatingly A
$ARep(\alpha, A)$	$\forall\Box\Diamond$	$AAlways(\alpha, AEven(\alpha, A))$	o.a.p. repeatingly A
$EInfOften(\alpha, A)$		$\nu X.\mu Y.\langle\alpha\rangle Y \vee (\langle\alpha\rangle X \wedge A)$	e.p.o.w. infinitely often A
$AInfOften(\alpha, A)$		$\nu X.[\alpha]' X \wedge AEven(\alpha, A)$	o.a.p. infinitely often A
$EEvenAlways(\alpha, A)$	$\exists\Diamond\Box$	$EEven(\alpha, EAlways(\alpha, A))$	e.p.o.w. eventually alw. A
$AEvenAlways(\alpha, A)$	$\forall\Diamond\Box$	$\mu X.\nu Y.[\alpha]Y \wedge ([\alpha]' X \vee A)$	o.a.p. eventually alw. A
$ELeadsTo(\alpha, A, B)$	$\exists \leadsto$	$\nu X.\langle\alpha\rangle' X \wedge$ $(A \rightarrow \mu Y.\langle\alpha\rangle Y \vee (X \wedge B))$	e.p.o.w. A alw. leads to B
$ALeadsTo(\alpha, A, B)$	$\forall \leadsto$	$AAlways(\alpha$ $A \rightarrow AEven(\alpha, B))$	o.a.p. A alw. leads to B

Table 4.1: Derived path operators. We will use $EEven(A)$ as an abbreviation for $EEven(., A)$.

It is illustrative to write out the I translation of $\forall(P \leadsto Q)$. Notice first that $\forall(P \leadsto Q) = \forall\Box(P \rightarrow \Diamond Q) = \forall\Box\forall(P \rightarrow \Diamond Q) = \forall\Box(P \rightarrow \forall\Diamond Q)$. We now get

$$I(\forall\Box(P \rightarrow \forall\Diamond Q)) = \nu X.[.]X \wedge (P \rightarrow \mu X.[.]' X \vee Q)$$

With this translation we can define a nice little collection of macros, which we systematically name as *EAlways* for 'there exists a path on which always ... ', and *AAlways* for 'on all paths always ... '. To fit the general framework we generalize the translated assertions to allow for relativized actions. Moreover, a slight anomaly arises in connection with the CTL assertion $\Box\Diamond$, which is usually understood as 'infinitely often'. With the semantics we have defined, this interpretation is only correct if attention is restricted to transition systems with total transition relations enforcing all maximal sequences to be infinite. Instead of this rather unpleasant restriction on the models, we give two variations of pair of macros for $\Box\Diamond$, one which requires the sequence to be infinite (*EInfOften*/*AInfOften*) and one which does not (*ERep*/*ARep*). Similar variations could be made of the other macros by removing or adding quotes. Table 4.1 shows the list of macros. Notice, that all assertions built using these macros will have alternation depth at most two. Figure 4.2 gives a visual explanation of some of the macros. The cones are to be thought of as the 'computation tree' of the underlying transition system with the initial state as the root.

This collection of macros includes the possibility of expressing what is known as *safety properties* (*Always*), *liveness properties* (*Even*), *fairness properties* (*Rep*/*InfOften*), and other *progress properties* like *responsiveness* (*Leadsto*). By combining these macros with constants and the abbreviations introduced in earlier chapters quite powerful assertions can be formed without having to "re-encode" directly in the modal μ-calculus.

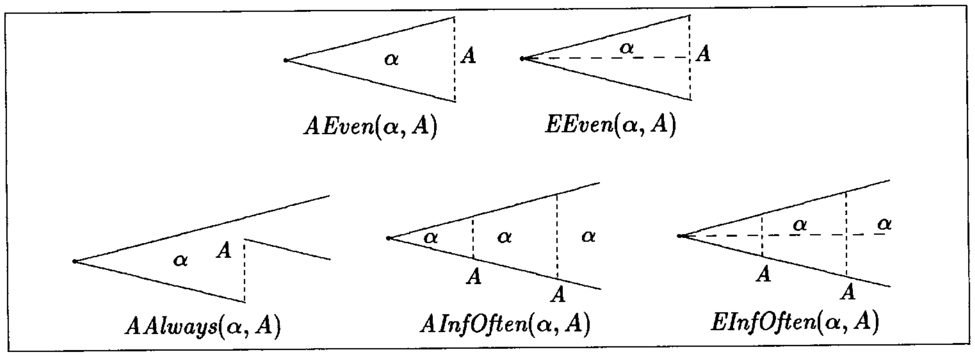


Figure 4.2: Some of the path operators.

Example 4.9

1. "Whenever ab^*a has occurred we always end up with a deadlock:"

$$AAlways([ab^*a]AEven(DeadLock)).$$

2. "Execution of a can always be followed in a finite number of steps by a b without performing any a's:"

$$AAlways([a]AEven(Act \setminus \{a\}, \langle\langle b \rangle\rangle T)).$$

3. "Requesting to get into a critical region by req will in a finite number of steps result in being in the critical region (denoted by the constant CR):"

$$AAlways([req]AEven(CR))$$

□

Combining the characteristic formulas from section 4.3 with the macros defined here we can express some rather complicated properties:

Example 4.10

1. "Eventually this process will end up behaving weakly bisimilar to q:"

$$AEven(\mathcal{W}/q).$$

2. "There exists a computation path on which it infinitely often will be possible to ready-simulate q:"

$$EInfOften(\mathcal{R}/q).$$

3. "All the states that are ready bisimilar to q are also strongly bisimilar to r:"

$$AAlways((\mathcal{RB}/q) \rightarrow (\mathcal{B}/r)).$$

4. "The pair of states that are weakly bisimilar but neither strongly bisimilar nor ready bisimilar all belongs to A except if they both can perform an a:"

$$((\mathcal{W} \wedge \neg\mathcal{B} \wedge \neg\mathcal{RB}) \rightarrow A) \vee \langle a \times a \rangle T.$$

It is not obvious how useful such constructions are. But at least we are able to be more refined in our analysis than using an approach purely based on equational reasoning between specifications and implementations; we can for instance try to express properties about the states for which the equality fails or for instance change the equivalences to only consider certain actions or states. It is even possible to build in assumptions like: 'Is p and q equivalent under the assumption that p' and q' are bisimilar?' and so on. $\Box$

4.4.2 Beyond CTL$^\bullet$

Now, one might ask what kind of formulae are not in CTL$^\bullet$ – and are impossible to translate into CTL$^\bullet$ using the equations of figure 4.1? A typical example would be

$$\Theta = \forall(\Box\Psi \vee \Box\Psi')$$

where Ψ and Ψ' are formulae in CTL$^\bullet$. None of the equations are applicable as neither $\forall$ nor $\Box$ distributes over disjunction. Nevertheless, we will show how one can find an assertion in the modal μ-calculus equivalent to Θ using a slightly more advanced idea than the simple translation I.

First, suppose that we have a constant $\dagger$ with valuation the set of states that cannot perform any transitions, i.e.

$$\mathcal{V}(\dagger) = \{s \in S \mid s \not\rightarrow\}$$

Then $\| \dagger \|_{T,V}$ will be a set of 'empty' sequences (consisting of single states). Using this constant $\bigcirc'$ can now be written as:

$$\bigcirc'\Phi = \bigcirc\ \Phi \vee \dagger.$$

It is not hard to prove that $\Box$ satisfy $\quad \Box\Phi \quad = \quad \Phi \vee \bigcirc' \Box\Phi$

$$(4.2)$$

(If we stick to the class of models with total transition relations, here and everywhere else the prime could be removed and everything will still be true.) Now, let us rewrite Θ using this equation:

$$\begin{aligned}
\Theta &= \forall(\vee\Box\Psi') \\
&= \forall((\Psi\wedge \bigcirc' \Box\Psi) \vee ((\Psi'\wedge \bigcirc' \Box\Psi')) \\
&= \forall((\Psi \vee \Psi') \wedge (\Psi\vee \bigcirc' \Box\Psi') \wedge (\Psi'\vee \bigcirc'\Box\Psi)\wedge \bigcirc' \Box\Psi\vee \bigcirc'\Box\Psi')) \\
&= \Delta \wedge \forall(\bigcirc' \Box\Psi\vee \bigcirc' \Box\Psi')) \\
&\qquad \text{where } \Delta = (\Psi \vee \Psi') \wedge (\Psi \vee \forall \bigcirc' \Box\Psi') \wedge (\Psi' \vee \forall \bigcirc' \Box\Psi) \\
&\qquad \text{using the equations of figure 4.4.1 as } \Psi \text{ and } \Psi' \text{ are CTL}^\bullet \\
&\qquad \text{formulae} \\
&= \Delta \wedge \forall \bigcirc' (\Box\Psi \vee \Box\Psi')) \\
&= \Delta \wedge \forall \bigcirc' \forall(\Box\Psi \vee \Box\Psi')) \\
&= \Delta \wedge \forall \bigcirc' \Theta
\end{aligned}$$

Hence, taking $F(X) = \Delta \vee \forall \bigcirc' X$ we have that $F(\Theta) = \Theta$. Now, suppose that Φ is another formula satisfying $F(\Phi) = \Phi$, is there any relationship between Φ and Θ? In fact, we will argue that $\Phi \leq \Theta$ where

$$\Phi \leq \Theta \Leftrightarrow_{\text{def}} \|\Phi\|_{T,V} \subseteq \|\Theta\|_{T,V} \text{ for all } T \text{ and } V.$$

In other words, Θ *is the maximum fixed-point of* F. Actually, we will prove the (apparently) stronger result that if U is a post-fixed point of the map on $\mathcal{P}(R_T)$ induced by F, which we denote by $\|F\|$, hence $\|F\|(U) \supseteq U$, then

$$U \subseteq \|\Theta\| \tag{4.3}$$

for some fixed T and V. As F yields state formulae when applied to state formulae (and $\|F\|$ yields subsets with the state property when applied to such subsets) we can intuitively speaking 'translate F' into a modal μ-calculus formula:

Proposition 4.3 *Giuen a transition system T and a valuation V. For all $CTL^\bullet$ formulae Ψ and Ψ' we have*

$$s \models \forall(\Box\Psi \vee \Box\Psi')$$
$$\Leftrightarrow$$
$$s \in [\![\nu X.(A \vee A') \wedge (A \vee [.]\nu Y.A' \wedge [.]Y) \wedge (A' \vee [.]\nu Y.A \wedge [.]Y) \wedge [.]X]\!]$$
$$\Leftrightarrow$$
$$s \in [\![\nu X.(A \vee \nu Y.A' \wedge [.]Y) \wedge (A' \vee \nu Y.A \wedge [.]Y) \wedge [.]X]\!]$$

where $A = I(\Psi), A' = I(\Psi')$.

Proof: The last bi-implication is by simple rewriting. For the first bi-implication, we continue the above discussion assuming that U is a set of runs, s.t. $U \subseteq \|F\|(U)$, and prove 4.3. We first prove that U is suffix-closed, i.e. for all $k \in \omega$,

$$\forall \delta \in U.\ k < |\delta| \Rightarrow \delta^{k+1} \in U$$

For the base case we get

$$\delta \in U = \|F\|(U) = \|\Delta\| \cap \|\forall \bigcirc'\|(U)$$

which implies that, for all δ' with $\delta_0 = \delta'_0, (\delta')^1 \in U$ hence in particular $\delta^1 \in U$. From this it can also be seen that U has the state property.

For the inductive step we assume that $\delta \in U$. Then as above we get $\delta^1 \in U$, which by the induction hypothesis implies $(\delta^1)^{k+1} = \delta^{k+2} \in U$.

As $\delta^k \in U$ for all $k \leq |\delta|$ and hence $\delta^k \in \|\Delta\|$ we have $\delta_k \in [\![\Delta]\!]$.

Now, we must argue that for all $\delta \in U$ we have

$$\begin{array}{c} \forall k \leq |\delta|. \quad \delta_k \in [\![\Psi]\!] \\ \text{or} \\ \forall k \leq |\delta|. \quad \delta_k \in [\![\Psi']\!] \end{array} \tag{4.4}$$

implying that $\delta \in \|\Box\Psi \vee \Box\Psi'\|$ and hence as U has the state property that $U \subseteq \|\Theta\|$. We already know that for all $k \leq |\delta|$ we have $\delta_k \in [\![\Delta]\!]$ thus in particular $\delta_k \in [\![\Psi]\!]$ or $\delta_k \in [\![\Psi']\!]$. Hence it is enough proving

$$\delta_k \notin [\![\Psi]\!] \quad \Rightarrow \quad \forall l \geq k.\ \delta_l \in [\![\Psi']\!] \tag{4.5}$$

Assume given such a k. Then $\delta_k \in [\![\Psi']\!]$ and as $\delta_k \in [\![\Delta]\!]$ also

$$\delta_k \in [\![\Psi]\!] \vee [\![\forall \bigcirc' \Box\Psi']\!],$$

hence

$$\delta_k \in [\![\forall \bigcirc' \Box\Psi']\!],$$

This implies that for all $l > k$, we have $\delta_l \in [\![\Psi']\!]$ and we are done.

Assuming for a moment that we extend the syntax for linear time formula with fixed-point operators with semantics an element of the lattice $\mathcal{P}(\mathit{Runs})$, we have actually shown that

$$\Theta = \nu X.\Delta \wedge \forall \bigcirc' X.$$

Now, notice that Δ is a CTL$^\bullet$ formula with

$$I(\Delta) = (A \vee A') \wedge (A \vee [.]\nu Y.A' \wedge [.]Y) \wedge (A' \vee [.]\nu Y.A \wedge [.]Y)$$

where $A = I(\Psi)$ and $A' = I(\Psi')$. We extend I to fixed-points by taking

$$\begin{aligned} I(\nu X.\Psi) &= \nu X.I(\Psi) \\ I(X) &= X \end{aligned}$$

hence

$$I(\Theta) = \nu X.I(\Delta) \wedge [.]X$$

and we claim that

$$[\![\Theta]\!] = [\![\nu X.\Delta \wedge \forall \bigcirc' X]\!] = [\![\nu X.I(\Delta) \wedge [.]X]\!].$$

(Proving this formally requires the use of a theorem relating fixed-points in different lattices, the reduction lemma of section 3.2 suffices.) □

If we considered instead another assertion outside CTL$^\bullet$, $\Theta = \forall(\Diamond\Psi \vee \Box\Psi')$ with Ψ and Ψ' CTL$^\bullet$ formulae we could try to employ the same trick using

$$\Diamond\Psi = \Psi \vee \bigcirc \Diamond\Psi$$

for the $\Diamond$-modality. This time both a minimum and a maximum fixed-point will be involved. We start rewriting Θ:

$$\begin{aligned} \Theta &= \forall(\Diamond\Psi \vee \Box\Psi') \\ &= \forall(\Psi \vee \bigcirc \Diamond\Psi \vee (\Psi' \wedge \bigcirc' \Box\Psi')) \\ &= \forall((\Psi \vee \Psi' \vee \bigcirc \Diamond\Psi) \wedge (\Psi \vee \bigcirc \Diamond\Psi \vee \bigcirc \Box\Psi' \vee \dagger)) \\ &= (\Psi \vee \Psi' \vee \forall \bigcirc \Diamond\Psi) \wedge (\Psi \vee \dagger \vee \forall\bigcirc\forall(\Diamond\Psi \vee \Box\Psi')) \\ &= \Delta \wedge (\Xi \vee \forall \bigcirc \Theta) \\ &\quad\text{where } \Delta = \Psi \vee \Psi' \vee \forall \bigcirc \Diamond\Psi \\ &\quad\text{and } \Xi = \Psi \vee \dagger \end{aligned}$$

Notice, that Δ and Ξ are in CTL$^\bullet$. Hence, Θ is a fixed-point of the function $F(X) = \Delta \wedge (\Xi \vee \forall \bigcirc X)$ and again it can actually be shown that all other (post-)fixed-points will be less, thus

$$\Theta = \nu X.\Delta \wedge (\Xi \vee \forall \bigcirc X)$$

and we find the μ-calculus formula

$$\nu X.I(\Delta) \wedge (I(\Xi) \vee [.]X),$$

where

$$I(\Delta) = I(\Psi) \vee I(\Psi') \vee [.]'\mu Y.[.]'Y \vee I(\Psi)$$

and

$$I(\Xi) = I(\Psi) \vee \dagger.$$

It is not obvious how far this idea of 'pulling out CTL$^\bullet$-formulae' can be taken and it is an interesting task to find out using this idea whether a more succinct translation than that of Dam [31] from CTL* into the modal μ-calculus is possible.

4.5 Bibliographic Notes

More literature on how to express properties in the modal μ-calculus can be found in for example Emerson and Clarke [34], Kozen [49], Emerson and Lei [35], Dam [31], and Stirling [79]. Case studies using the modal μ-calculus can be found in Walker [87] and Bruns [18].

We have shown how a variety of properties can be expressed in the extended modal μ-calculus including linear time temporal properties, equivalences, preorders, and characteristic formulae. We shall later in section 5.10 see how to get algorithms for automatically checking all these relations.

As we have seen, the modal μ-calculus seems to be a good candidate for a low-level general language for expressing behaviours of concurrent systems and could as such be used as the backbone of a general verification tool.

Cleaveland and Steffen present in [26] ideas of computing preorders very much like the present approach, however, whereas they suggest checking

$s \leq s'$ for a preorder $\leq$ by generating a characteristic formula C of $\leq$ with respect to s by *ad hoc* means and verify whether s' satisfies s, we get the same effect much simpler by writing down directly the definition of $\leq$ as a formula in the extended modal μ-calculus allowing model checking algorithms to be applied directly, and in effect we can get characteristic formulas *for free* through the reduction for product; hence also characteristic formulas for all the preorders investigated by Steffen [75].

Chapter 5

Model Checking in Finite-State Systems

In this chapter we consider the problem of determining satisfaction for finite-state systems, a task which is often referred to as *model checking*.

The compositional method from chapter 3 already offers one way of performing model checking: Given a finite-state process and a closed assertion we can repeatedly apply the reductions until we end up with a boolean expression with atoms that are correctness assertions about the *nil* process. These correctness assertions can easily be removed by the reduction for *nil*, and we arrive at a boolean expression which can be evaluated to produce the answer. This, however, is not an efficient algorithm.

Instead, using the reduction for product we will describe a very simple way of transforming the satisfaction problem into a problem of determining the value of a *boolean fixed-point expression* – a boolean expression involving simultaneous fixed-point operators over boolean-valued variables – and describe algorithms for evaluating such expressions in an efficient manner.

Remark 5.1 In the analysis of time and space complexities we are going to make in the sequel, we will make use of some general assumptions about the representations of assertions and transition systems. Firstly, variables and labels will be assumed to be represented by natural numbers, which in turn will be assumed to be representable in a constant amount of memory.[1]

[1]As usual in camplexity analysis we make the assumptions that integers amount of memory and that an arbitrary memory address can be accessed in can be stored in a

Secondly, functions from an interval of the natural numbers to a set of 'simple' values, e.g. numbers, will be represented efficiently such that access to the value at one particular element in the domain can be performed in constant time (like 'arrays' in many programming languages). Thirdly, we assume that directed graphs consisting of a set of nodes (an interval of natural numbers) and a set of edges (pairs of natural numbers) are represented such that a list containing the edges out of one particular node can be found in constant time and such that this list can be traversed in linear time. A labelled transition system and a simple equation system can be implemented by such a graph representation – for the transition systems, labels are also attached to the edges. Fourth, the set of states is assumed to be an interval of the natural numbers such that subsets of states can be represented as their characteristic functions with contant time tests for membership.

These assumptions are met by the class of machine models called *RAM models*; an abstract machine model which in practice is realized by all general purpose computers. We will later discuss to what extent even weaker models suffice for implementing our algorithms.

Often we will use statements like this algorithm runs 'in time and space $K(n)$', where it actually should be 'in time and space asymptotically bounded by $K(n)$'. We will use the notation $O(K(n))$ for this statement. All these assumptions and slight abuses of language are standard when analyzing complexities of algorithms (see for example Hopcroft and Ullman [43]). □

5.1 Tansforming Satisfaction to Boolean Expressions

We will start by looking only at assertions in $\mu\mathrm{K}_{\texttt{where},Q}$, i.e. the standard calculus extended with $\texttt{where}_\mu$-clauses and constants, and discuss the generalizations to $\mu\mathrm{K}_{ext}$ in section 5.9. Assume we have given a finite-state process p and a closed assertion in $\mu\mathrm{K}_{\texttt{where},Q}$. The transformation of the satisfaction problem

$$\models p : A$$

constant constant time (the 'uniform cost criterion', cf. Aho, Hopcroft and Ullman [3]).

to a boolean expression with fixed-points will proceed in three steps. First A is transformed to positive, normal form by pushing negations inwards. Secondly, the fixed-points of A are transformed to a *simple form.* And thirdly we 'divide' this assertion by the process p to get a boolean fixed-point expression.

The first two steps were described in section 2.6.1 and section 3.7. (The idea of translating into a simple form is due to Arnold and Crubille [9].) The third step of the translation will turn out to be a special case of the reduction for product! To motivate this translation, assume given a finite-state process p and a closed assertion A. Instead of deciding whether

$$\models p : A \tag{5.1}$$

holds, we could take an apparent detour by considering instead the process $(nil \times p)\{\Xi\}$ where $\Xi : Act_* \rightharpoonup Act_*$ is defined by

$$\Xi(x) = \begin{cases} a & \text{if } x \equiv * \times a, a \in Act \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then it should be obvious that (5.1) is valid, if and only if,

$$\models (nil \times p)\{\Xi\} : A \tag{5.2}$$

is valid.

For (5.2) we can apply first the reduction for relabelling to get an assertion $\text{red}_{\{\Xi\}}(A)$ which is actually going to be like A except that all modalities $\langle a \rangle$ have changed to $\langle * \times a \rangle$ and then proceed with the reduction for product to get the assertion $B = \text{red}_{\times p}(\text{red}_{\{\Xi\}}(A))$ which by theorems 3.4 and 3.7 has the property that (5.2) is valid, if and only if,

$$\models nil : B \tag{5.3}$$

is valid.

However, if we consider how B looks, we discover that in it *no modalities except modaliities over the idling action appear.* As $[\![\langle * \rangle A]\!]_{nil}\rho = [\![A]\!]_{nil}\rho$ even these trivial modalities can be removed and we get an equivalent b which is a *boolean expression with simultaneous fixed-points*, the value of which determines the validity of (5.3) and hence of (5.1).

The combined reduction consisting of applying first the reduction for relabelling, then the reduction for product and finally removing the trivial modalities will be called *dividing A with p* and is tabulated in figure 5.1.

$$
\begin{array}{rcl}
(\vec{A}^{l}\ \mathtt{where}_{\mu}\ \vec{Y}^{m} = \vec{P}^{m})/s_i & = & (\vec{A}^{l}/s_i)\ \mathtt{where}_{\mu}\ \vec{Z}^{nm} = (\vec{P}^{m}/\vec{s}) \\
 & & \quad \text{where } \vec{Z}^{nm} = \sigma(\vec{Y}^{m}) \\
(\vec{A}^{l}\ \mathtt{where}_{\nu}\ \vec{Y}^{m} = \vec{P}^{m})/s_i & = & (\vec{A}^{l}/s_i)\ \mathtt{where}_{\nu}\ \vec{Z}^{nm} = (\vec{P}^{m}/\vec{s}) \\
 & & \quad \text{where } \vec{Z}^{nm} = \sigma(\vec{Y}^{m}) \\
(A_1,\ldots,A_l)/s_i & = & (A_1/s_i,\ldots,A_l/s_i) \\
F/s_i & = & F \\
T/s_i & = & T \\
A_0 \vee A_1/s_i & = & (A_0/s_i) \vee (A_1/s_i) \\
A_0 \wedge A_1/s_i & = & (A_0/s_i) \wedge (A_1/s_i) \\
X/s_i & = & X_{s_i} \quad \text{where } \sigma(X) = \mathit{IN}(X_{s_1},\ldots,X_{s_n}) \\
\langle a\rangle A/s_i & = & \bigvee\{A/s' \mid s_i \xrightarrow{a} s'\} \\
[a]A/s_i & = & \bigwedge\{A/s' \mid s_i \xrightarrow{a} s'\} \\
Q/s_i & = & \begin{cases} T & \text{if } s_i \in V(Q) \\ F & \text{if } s_i \notin V(Q) \end{cases} \\
 & & \\
(\vec{A}^{l}\ \mathtt{where}_{\mu}\ \vec{Y}^{m} = \vec{P}^{m})/\vec{s}^{n} & = & (\vec{A}^{l}/\vec{s}^{n})\ \mathtt{where}_{\mu}\ \vec{Z}^{nm} = (\vec{P}^{m}/\vec{s}^{n}) \\
 & & \quad \text{where } \vec{Z}^{nm} = \sigma(\vec{Y}^{m}) \\
(\vec{A}^{l}\ \mathtt{where}_{\nu}\ \vec{Y}^{m} = \vec{P}^{m})/\vec{s}^{n} & = & (\vec{A}^{l}/\vec{s}^{n})\ \mathtt{where}_{\nu}\ \vec{Z}^{nm} = (\vec{P}^{m}/\vec{s}^{n}) \\
 & & \quad \text{where } \vec{Z}^{nm} = \sigma(\vec{Y}^{m}) \\
(A_1,\ldots,A_l)/\vec{s}^{n} & = & (A_1/\vec{s}^{n},\ldots,A_l/\vec{s}^{n}) \\
A/\vec{s}^{n} & = & (A/s_1,\ldots,A/s_n) \\
 & & \quad \text{if } A \text{ is not a product assertion}
\end{array}
$$

Figure 5.1: The division operator; A/s (respectively $(A/\vec{s})$) abbreviates $\mathrm{red}_s(A;\sigma,V)$ (respectively $\mathrm{red}_{\vec{s}}(A;\sigma,V)$).

From the above discussion we get as an immediate corollary of theorem 3.8:

Corollary 5.1 *Assume given a finite transition system T with states $S = \{s_1,\ldots,s_n\}$, a valuation V and an assertion A in $\mu K_{\mathsf{where},Q}$. Then for a change of variables σ which is fresh for A,*

$$[\![A[\sigma]]\!]_{T,V}\rho = in([\![\mathrm{red}_{\vec{s}^{n}}(A;\sigma,V)]\!]_{nil,V'}\rho)$$

where V' is an arbitrary valuation and in is the composition of the in-map from the reduction of relabelling and the reduction of product, i.e. the map $in : (\mathcal{P}(nil))^n \to \mathcal{P}(S)$ with

$$in(\vec{u}^{n}) = \{s_i \mid u_i = \{nil\}\}.$$

How do we now evaluate $b = \mathrm{red}_{s_i}(A; \sigma, V)$? As b contains fixed-points this is not a trivial task. Semantically, when interpreting b over the one-state transition system pointed by *nil*, the denotation of b will be either the property $\emptyset$ or the property $\{nil\}$. However, the two-point lattice $\mathcal{P}(\{nil\})$ of properties of *nil* is nothing else than an isomorphic copy of the well-known Sierpinski space $\mathbb{O} = \{0, 1\}$ with ordering $0 < 1$, so for convenience we consider the semantics of boolean fixed-point expressions as being given as an assertion about *nil*, but we will use 0, for false, and 1, for true, for the values $\emptyset$ and $\{nil\}$. Hence, we can define for a closed b,

$$[\![b]\!] = \begin{cases} 0 & \text{if } [\![b]\!]_{nil}\rho = \emptyset \\ 1 & \text{if } [\![b]\!]_{nil}\rho = \{nil\} \end{cases}$$

for an arbitrary environment ρ. We proceed similarly for tuples $\vec{b}$.

Now, for a monotonic function f on $\mathbb{O}$ we observe the following property of fixed-points:

Proposition 5.1 *If $f : \mathbb{O} \to \mathbb{O}$ is a monotonic function then $\mu f = f(0)$ and $\nu f = f(1)$.*

Proof: Trivial. $\Box$

Using this simple observation we can compute the value of b by first applying Bekič's theorem to get only unary fixed-points and replace all the fixed-points with their bodies applied to 0 (for minimum fixed-points) or to 1 (for maximum fixed-points) resulting in a simple boolean expression that only contains disjunctions and conjunctions over the atoms 0 and 1 and which is easily evaluated to yield the result.

However, the use of Bekič's theorem has the potential danger of increasing exponentially the size of the expression, so although the evaluation is simple ("linear time") the expression to evaluate can be huge. Curiously enough, this simple observation seems to offer an explanation of why the tableau-based methods of Stirling and Walker [80] and Cleaveland [23], and the methods of Larsen [53] and Winskel [92] have bad complexities compared to the algorithms we are going to present (see section 5.2 below for a discussion of this point).

Instead we will transform simple, simultaneous fixed-points $\vec{x} = \vec{b}$ to a normalized form, which resembles a kind of directed graphs. Inspired by this analogy we give graph-like algorithms for computing the fixed-points. In section 5.3 we present a *global* algorithm computing the values of all of $\vec{x}$ and in section 5.6 a *local* algorithm computing only a certain minimal part of $\vec{x}$.

But before proceeding to the evaluation we state and prove the correctness of the three-step translation just given.

Theorem 5.1 *Given a closed assertion A in $\mu K_{\texttt{where},Q}$, a state s in a finite transition system $T = (S, L, \rightarrow)$ and a valuation V, then we can find a boolean fixed-point expression b such that*

$$\begin{array}{rl} (i) & \models_{T,V} s : A \Leftrightarrow [\![b]\!] = 1, \\ (ii) & b \textit{ has } O(|A||S|) \textit{ variables}, \\ (iii) & \textit{the size of } b \textit{ is } O(|A||T|), \\ (iv) & b \textit{ can be computed in time } O(|A||T|), \\ (v) & ad(b) = \max\{1, ad(A)\}, \textit{ and} \\ (vi) & b \textit{ is simple.} \end{array}$$

Proof: Take A^0 to be the positive normal form of A. Surely, A^0 can be computed from A in linear time and this without changing the alternation depth. Let A^1 be the simple form of A^0 as given by lemma 3.5. Take $b = \text{red}_s(A^1; \sigma, V)$ for some change of variables σ. Hence, since the division is a special case of the reduction for product, $(i) - (v)$ follows from lemma 3.5 and lemma 3.6. Finally, it is easy from the definition of division, to observe that since A^1 is simple so is b. $\Box$

For a single unnested fixed-point $(X_i \; \texttt{where}_\mu \; \vec{X}^{\,l} = \vec{A}^{\,l})$ with $|\;\vec{A}^{\,m}\;| = k$ our transformation first derives a simple k-ary fixed-point $(Y_j \; \texttt{where}_\mu \; \vec{Y}^{\,k} = \vec{B}^{\,k})$ and then, given a transition system with n states, transforms this into a nk-ary fixed-point $(y_{ji}\texttt{where}_\mu \; \vec{y}^{\,nk} = \vec{b}^{\,nk})$ where $\vec{b}^{\,nk}$ only consists of conjunctions and disjunctions over variables. By these transformations we have reduced the problem of finding a fixed-point over the lattice $\mathcal{P}(S)^l$ to a problem of finding a fixed-point of a boolean function over the lattice $\mathbb{O}^{nk}$.

We will normalize the equations of the boolean fixed-point expression slightly more:

Definition 5.1 The boolean equation system $\vec{x}^{\,n} = \vec{b}^{\,n}$ with free variables V is on *normalized, simple form* if for all $i, 1 \leq i \leq n$, we have

$$b_i = \bigvee \chi \text{ or } b_i = \bigwedge \chi$$

for a $\chi \subseteq V$. $\Box$

Any boolean expression in simple form is easily normalized by using (somewhat arbitrarily) disjunction of a singleton set if the right-hand side is a variable, and empty disjunctions and conjunctions for false and true.

An equational system $\vec{x}^{\,n} = \vec{b}^{\,n}$ in normalized, simple form can be thought of as a directed graph taking the variables as nodes, letting occurrences of variables on the right-hand side induce edges, and labelling the nodes with disjunctions and conjunctions depending on the connective of the right-hand sides. An example is provided in figure 5.2.

5.2 Relation to Other Model-Checking Algorithms

It might appear that the boolean expression resulting from the translation of the satisfaction problem $\models p : A$ has very little to do with the original problem. However, the situation is indeed very different. Because of the step transforming A into an equivalent simple assertion, in effect, adding a variable for each subassertion, and the subsequent division by p, b actually contains a *boolean variable $x_{s':A'}$ for each state s' and subussertion A' of A*, corresponding precisely to the satisfaction problem

$$\models_\rho s' : A'$$

for some proper environment ρ.

In this sense, there is a *very* close relationship between variables of b and the satisfaction problem; what we are doing is actually to be quite explicit about which states satisfy which subassertions of A and it is by explicit representation of the dependencies between these satisfaction problems that we succeed in getting efficient algorithms — and it is the lack of this which makes other algorithms inefficient.

$$\begin{aligned} x_1 &= \bigwedge\{x_1, x_3, x_5\} \\ x_2 &= \bigvee\{x_1, x_2\} \\ x_3 &= x_2 \\ x_4 &= T \\ x_5 &= F \end{aligned}$$

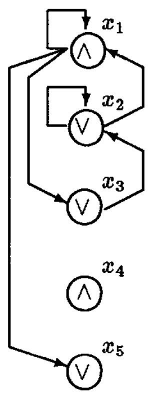


Figure 5.2: A simple equational system and its normalized version given as a graph.

This is easiest to see for the algorithm of Winskel [92] since it is closest to the algorithms here. First, however, it will be useful to review Bekič's theorem for two-point lattices:

Corollary 5.2 (Bekič's theorem for two-point lattices.) *Let* $f_i : \mathbb{O}^n \to \mathbb{O}$ *for* $1 \leq i \leq n$ *be monotonic maps. Then*

$$\mu(x_1, \ldots, x_n).(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)) = (E_1^{\emptyset}, \ldots, E_n^{\emptyset})$$

where

$$E_i^J = \begin{cases} 0 & \textit{if } i \in J \\ f_i(E_1^{J \cup \{i\}}, \ldots, E_n^{J \cup \{i\}}) & \textit{otherwise} \end{cases}$$

Proof: From theorem 2.3 using proposition 5.1. □

The corollary works by unfolding the fixed-points of each component in a tree-like manner, starting with a variable and replacing any variable the first time it is met by its right-hand side and the second time by a zero, such that on all paths from the root of the tree no zero will have been substituted for a variable without the path having passed the right-hand side of that variable.

Winskel's [92] model-checking algorithm is described as a set of rewrite rules on correctness assertions. The fixed-points are decorated with sets of states U such as $\mu X\{U\}A$, and the semantics is[2]

$$[\![\mu X\{U\}A]\!]\rho = \mu W.([\![A]\!]\rho[W/X] \setminus U).$$

For example, he gives for the diamond-modality and the minimum fixed-point the rules

$$\begin{array}{rcl} (s : \langle a\rangle A) & \to & \bigvee_{s';s\xrightarrow{a}s'} (s' : A) \\ (s : \mu X\{U\}A) & \to & \begin{cases} F & \text{if } s \in U \\ (s : A[\mu X\{U,s\}A/X]) & \text{if } s \notin U \end{cases} \end{array} \tag{5.4}$$

The μ-rule (5.4) is based on the observation that

$$s \in [\![\mu X\{U\}A]\!]\rho \Leftrightarrow s \in [\![A]\!]\rho[[\![\mu X\{U,s\}A]\!]\rho/X] \setminus U \tag{5.5}$$

assuming that $\mu X\{U\}A$ is a closed assertion. (This follows from a lemma proven by Winskel, and cited as the "second reduction lemma" (lemma 7.1) in chapter 7 on page 189). We might already notice that the unfolding of the fixed-point, attaching another state to the fixed-point, is very similar to how the E_i^J's in Bekič's theorem adds new indices to J. We will actually formalize this and show a very precise correspondence.

Let us consider the fixed-point assertion $\mu X.A$ abbreviating (X $\texttt{where}_\mu$ $X = A$). By performing the division with respect to the states $S = \{s_1, \ldots, s_n\}$ of a transition system T, we get the assertion

$$\vec{x}^{\,n} \ \texttt{where}_\mu \ \vec{x}^{\,n} = (A/\ \vec{s}^{\,n})$$

where $A/\ \vec{s}^{\,n} = \mathrm{red}_{\vec{s}^{\,n}}(A;\sigma)$ for a change of variables σ. From corollary 5.1 it follows that

$$[\![\mu X.A]\!]\rho = in([\![\ \vec{x}^{\,n} \ \texttt{where}_\mu \ \vec{x}^{\,n} = (A/\ \vec{s}^{\,n})]\!]\rho)$$

where $in : \mathbb{O}^n \to \mathcal{P}(S)$ is the isomorphism taking $\vec{u}^{\,n}$ to $\{s_i \mid u_i = 1\}$. Hence in particular

[2] Winskel uses maximum fixed-points; we dualize to minimum fixed-points to fit our present discussion.

$$s_i \in [\![\mu X.A]\!]\rho \Leftrightarrow [\![x_i \ \texttt{where}_\mu \ \vec{x}^{\,n} \ = (A/ \ \vec{s}^{\,n} \ )]\!]\rho = 1 \Leftrightarrow [\![E_i^{\emptyset}]\!]\rho = 1$$

where the last bi-implication follows from corollary 5.2, and E_i^J is the expression from that corollary. We will actually strengthen this statement and prove that for all $U \subseteq S$

$$[\![\mu X\{U\}A]\!]\rho \quad = \quad in([\![\vec{E}^{\{j|s_j \in U\}}]\!]\rho) \tag{5.6}$$

where for all $J, \vec{E}^J = (E_1^J, \ldots, E_n^J)$. The proof is by induction on the size of $S \setminus U$. For the base case, $U = S$, and the statement is vacuously true as both sides are empty. For the inductive step assume $s_i \notin U$, and deduce as follows:

$$\begin{aligned}
s_i \in [\![\mu X\{U\}A]\!]\rho &\Leftrightarrow s_i \in [\![A]\!]\rho[[\![\mu X\{U, s_i\}A]\!]\rho/X] \\
&\qquad \text{by } \ 5.5 \text{ as } s_i \notin U \\
&\Leftrightarrow s_i \in [\![A]\!]\rho[in([\![\vec{E}^{\{j|s_j \in U \cup \{s_i\}}]\!])/X] \\
&\qquad \text{by the induction hypothesis} \\
&\Leftrightarrow s_i \in [\![A[\sigma]]\!]\rho[[\![\vec{E}^{\{j|s_j \in U \cup \{s_i\}\}}]\!]/\vec{x}] = 1 \\
&\qquad \text{for a change of variables } \sigma \text{ with } \sigma(X) = IN(\vec{x}) \\
&\Leftrightarrow [\![A/s_i]\!]\rho[[\![\vec{E}^{\{j|s_j \in U \cup \{s_i\}\}}]\!]/\vec{x}] = 1 \\
&\qquad \text{by the definition of } A/s_i \\
&\Leftrightarrow [\![E_i^{\{j|s_j \in U\}}]\!] = 1 \\
&\qquad \text{by definition of } E_i^J
\end{aligned}$$

If $s_i \in U$ then clearly $s_i \notin [\![\mu X\{U\}A]\!]\rho$ by (5.5) and $E_i^{\{j|s_j \in U\}} = 0$, completing the inductive step.

This proves that the two rules for $\mu X\{U\}A$ are precisely matched by the two defining clauses for $\vec{E}$ in Bekič's theorem, that is, performing a rewrite step according to the rule (5.4), corresponds to unfolding the definition of E_i^J:

$$\begin{aligned}
E_i^{j|s_j \in U} \quad &= \quad \begin{cases} 0 & \text{if } s_i \in U \\ A/s_i[\vec{E}^{\{j|s_j \in U \cup \{s_i\}\}}/\vec{x}] & \text{if } s_i \notin U \end{cases} \\
s_i : \mu X\{U\}A \quad &\rightarrow \quad \begin{cases} F & \text{if } s_i \in U \\ s_i : A[\mu X\{U, s_i\}A/X] & \text{if } s_i \notin U \end{cases}
\end{aligned}$$

This shows that

> *the model-checker of Winskel can be viewed as a way of generating and evaluating at the same time the expression that results from applying Bekič's theorem to the boolean fixed-point expression originating from the division $(\mu X.A)/s_i$.*

As we have already seen Bekič's theorem can generate highly exponentially sized expres-sions, explaining why the algorithm as it stands is very bad.[3] Exactly the same thing happens with the tableau-methods of Larsen [53] and Stirling and Walker [80] (see for instance Winskel's discussion about the relations between the methods), even the improvements suggested by Cleaveland [23] do not remove the exponential behaviour. However, Larsen [54] has recently proposed an improvement to his original algorithm, which for one fixed-point gives a polynomial time algorithm. His improvement can be seen as exploiting the observation that some of the subexpressions generated in applying Bekič's theorem imply others allowing for a certain amount of "re-use" of information. This re-use is, however, not enough to achieve the efficiency of the algorithms we present in subsequent sections.

5.3 A Global Algorithm

In this section we will describe an algorithm for computing the minimum fixed-point of a normalized, simple equation system $\vec{x}^{\,n} = \vec{b}^{\,n}$. It will be *global* in the sense that it computes the complete fixed-point, and it will have time and space complexity $O(|\vec{b}^{\,n}|)$. If $\vec{b}^{\,n}$ is constructed from an unnested fixed-point formula (X_i $\texttt{where}_\mu$ $\vec{X}^{\,l} = \vec{A}^{\,l}$) and a transition system T as described in section 5.1, the size of $\vec{b}^{\,n}$ will be $O(|A||T|)$, hence we have a global model checking algorithm that in the worst-case is linear in the product of the size of the assertion and the size of the transition system.

We present the algorithm in the version for finding minimum fixed-points, the case of maximum fixed-points being completely dual.

[3]Note that the fix is not simply a question of storing information (called "dynamic programming" see e.g. Aho, Hopcroft and Ullman [3]), since Bekič's theorem generates exponentially many *different* subexpressions.

Recall, that $\vec{b}^n$ with free variables $V = \{x_1, \ldots, x_n\}$ induces a monotonic function $f : \mathbb{O}^V \to \mathbb{O}^V$, and the fixed-point we are interested in computing is the 'environment' μf. The algorithm will start with the bottom element of the lattice $\mathbb{O}^V$ and gradually increase it until eventually the minimum fixed-point will be reached. Pictorially one can think of the algorithm as 'chasing ones' around the graph described by $\vec{b}^n$: Starting with variables that have empty conjunctions on the right-hand sides and therefore must have value one, we look for dependent variables that can be forced to be one, repeating until no further variables can be forced to one thereby as it will turn out having found the minimum fixed-point.

Figure (5.3) describes the algorithm. The function $st : V \to \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers, denotes the 'strength' of variables, i.e. $st(x_i)$ is the number of successors that must be one before the variable x_i will be forced to be one. The function f induced by $\vec{b}^n$ can be extended to a function on strengths by taking for all $x_i \in V$:

$$\dot{f}(st)(x_i) = \begin{cases} |\chi_i \cap st_{>0}| & \text{if } b_i = \bigwedge \chi_i \\ 1 - |\chi_i \cap st_{\leq 0}| & \text{if } b_i = \bigvee \chi_i \end{cases}$$

where $st_{>0} = \{v \mid st(v) > 0\}$, i.e. the set of variables which still need some successors to become one, and $st_{\leq 0} = \{v \mid st(v) \leq 0\}$, i.e. the set of variables which have enough successors that are one (negative values indicating the 'excess' of ones). A strength defines an environment $\widehat{st} \in \mathbb{O}^V$ by

$$\widehat{st}(v) = \begin{cases} 1 & \text{if } st(v) \leq 0 \\ 0 & \text{if } st(v) > 0 \end{cases}$$

It is now easy to see that if $\dot{f}(st) = st$ then $f(\widehat{st}) = \widehat{st}$, implying that $\widehat{st}$ is a fixed-point of f. The algorithm will compute a strength st with this property.

In the algorithm, the set A will denote an 'active' set of variables with value one for which the consequences of becoming one has not yet been computed. Correctness can be shown from the invariant I:

$$\begin{aligned} I \quad \Leftrightarrow_{\text{def}} \quad & A \subseteq st_{\leq 0} \ \& \\ & \widehat{st} \leq \mu f \ \& \\ & \forall x_i \in V.\ (st)(x_i) = \begin{cases} |\chi_i \cap (st_{>0} \cup A| & \text{if } b_i = \bigwedge \chi_i \\ 1 - |\chi_i \cap (st_{\leq 0} \backslash A| & \text{if } b_i = \bigvee \chi_i \end{cases} \end{aligned}$$

Theorem 5.2 *The algorithm of figure* 5.3 *correctly computes the minimum fixed-point μf and it can be implemented to run in time $O(|\vec{b}^{\,n}|)$.*

Input: A normalized, simple equation system $\vec{x}^n = \vec{b}^n$ where $\vec{b}^n$ has free variables in $V = \{x_1, \ldots, x_n\}$ and induces the function $f : \mathbb{O}^V \to \mathbb{O}^V$.
Output: An environment $m : \mathbb{O}^V$ equal to μf.

```
for all $x_i \in V$ do $st(x_i) := \begin{cases} |\chi_i| & \text{if } b_i = \bigwedge \chi_i \\ 1 & \text{if } b_i = \bigvee \chi_i \end{cases}$
$A := st_{\leq 0}$
while $A \neq \emptyset$ do
        pick an $x_j \in A$  $A := A \setminus \{x_j\}$
        for all $x_i \in Pred(x_j)$ do
                $st(x_i) := st(x_i) - 1$
                if $st(x_i) = 0$ then $A := A \cup \{x_i\}$ fi
        od
od
$m := \widehat{st}$
```

Figure 5.3: A global algorithm: Chasing 1's. The set of predecessors of a variable x_j is the set $Pred(x_j) = \{x_i \mid x_j$ belongs to the χ_i on the right-hand side of $x_i\}$.

Proof: It is a simple exercise to show that the invariant I holds immediately before the while-loop and that it is preserved by the body. When the while-loop terminates we have $A = \emptyset$ which from the invariant implies that $st = \dot{f}\ (st)$ and $\widehat{st}$ is a fixed-point, which by the second conjunct of the invariant is less than or equal to the minimum fixed-point, hence $\hat{st} = \mu f$.

For the time complexity, we assume that A is implemented as for instance a stack with constant insertion and deletion times. The strength is simply a map from the variables to the integers, which according to our assumptions can be implemented with constant access times. Now, first notice that whenever a variable has been removed from A, it will never be inserted again as this only happens when its strength equals zero, and strengths always decrease. Hence the body of the while-loop will at most be executed once for each variable. Each execution of the innermost for-all-loop takes time proportional to the size of $Pred(v)$, i.e. the number of predecessors of the variable v. In total the while-loop takes time proportional to the sum of the number of predecessors, i.e. the total number of edges in G, and is thus bounded by $|\vec{b}^{\,n}|$. The first loop and the last assignment are also bounded by $|\vec{b}^{\,n}|$.

As the algorithm looks at predecessors of variables the 'graph' must initially be reversed, which can easily be done in linear time (see e.g. Tarjan [81]). $\Box$

5.4 A Global Algorithm for Alternating Fixed-Points

In this section we will describe how the global algorithm, Chasing 1's, can be extended to yield a model checker for assertions with nested fixed-points with running time $O(|A|^k|S|^{k-1}|T|)$, where $k = \max\{1, ad(A)\}$.

The algorithm, presented in figure 5.4, needs the notion of a top μ-assertion. Recall, that a μ-subassertion of A is a subassertion of A with main connective $\texttt{where}_\mu$.

Definition 5.2 A *μ-subassertion* of A is a subassertion of A with main connective $\texttt{where}_\mu$. Dually for a ν-subassertion. The *top μ-assertions* of A is the set of μ-subassertions of A which are not subassertions of any ν-subassertion of A. $\Box$

The algorithm follows very closely the definition of alternation depth given in 2.6, which simplifies the analysis considerably.

Theorem 5.3 *Assume given an algorithm that can compute closed, simultaneous, unnested fixed-points* ($\vec{X}$ $\texttt{where}_\mu$ $\vec{X} = \vec{A}$) and ($\vec{X}$ $\texttt{where}_\nu$ $\vec{X} = \vec{A}$) *on a transition system T in time $O(|A||T|)$. There exists an algorithm that will compute the set of states denoted by a closed assertion A in $\mu K_{\texttt{where},Q}$ in time $O(|A|^k|S|^{k-1}|T|)$ and space $O(|A||T|)$, where $k = \max\{ad(A), 1\}$.*

Proof: Assume that the efficient algorithm for unnested fixed-points runs in time bounded by $c|A||T|$ for a constant c.

Define the predicate P on closed assertions by

$P(A) \Leftrightarrow_{\mathrm{def}}$ for all V. Compute(A, V) executes in time asymptotically bounded by $c|A|^k|S|^{k-1}|T|$,

where $k = \max\{ad(A), 1\}$. Assume inductively that for all $A', |A'| < |A| \Rightarrow P(A')$. We show by cases that $P(A)$ holds. (In the sequel k will always be $\max\{ad(A), 1\}$.)

Case $m > 0$. By the induction hypothesis Compute(B_i, V) takes time $c|B_i|^{k_i}|S|^{k_i-1}|T|$, where $k_i = \max\{ad(B_i), 1\}$.
Hence letting $k' = \max\{ad(A'), 1\}$ the total time for computing A is

$$\begin{aligned} c|A'|^{k'}|S|^{k'-1}|T| + \sum_{i=1}^{m} c|B_i|^{k_i}|S|^{k_i-1}|T| &\leq c|A'|^{k}|S|^{k-1}|T| + \sum_{i=1}^{m} c|B_i|^{k}|S|^{k-1}|T| \\ &\quad \text{by definition of alternation depth} \\ &\leq c(|A'| + \sum_{i=1}^{m}|B_i|)^k|S|^{k-1}|T| \\ &\leq c|A|^k|S|^{k-1}|T|. \end{aligned}$$

Case $A \equiv Q$. Trivial.

Case $A \equiv \langle a \rangle B$. The time to compute the diamond-modality is bounded by $c|T|$, hence the total cost is $c|T| + c|B|^k|S|^{k-1}|T| \leq c(|B| + 1)^k|S|^{k-1}||T| = c|A|S|^{k-1}||T|$.

Case $A \equiv B_0 \wedge B_1$. As for $\langle a \rangle B$.

Case $A \equiv (C\ \mathtt{where}_\mu\ \vec{X}^n = \vec{B}^n)$. Observe that $|\vec{B}'| \leq |A|$.

Subcase $\vec{B}'$ unnested. As $ad(A) = ad(\vec{B}') = 1$ the claim follows immediately from the assumption about the efficient algorithm for unnested fixed-points and the induction hypothesis on C'.

Subcase $\vec{B}'$ nested. Let $k'_i = ad(B'_i)$ and $k' = ad(C')$. Observe, that $k = \max\{k', 1 + \max\{k'_i \mid 1 \leq i \leq m\}\}$. By the induction hypothesis each iterate $\vec{U}^m$ is computed in time

$$\sum_{i=1}^{m} c|B'_i|^{k'_i}|S|^{k'_i-1}|T| \leq c|\vec{B}'|^{\max\{k'_i|1\leq i\leq m\}}|S|^{\max\{k'_i|1\leq i\leq m\}-1}|T|$$

The number of required iterations are bounded by the height of the lattice $\mathcal{P}(S)^m$, which is $m|S| \leq |\vec{B}'||S|$. Hence, the total cost of computing the fixed-point is

$$|\vec{B}'||S|c|\vec{B}'|^{\max\{k'_i|1\leq i\leq m\}}|S|^{\max\{k'_i|1\leq i\leq m\}-1}|T|$$

$$= c|\vec{B}'|^{\max\{k'_i|1\leq i\leq m\}+1}|S|^{\max\{k'_i|1\leq i\leq m\}}|T|$$

By the induction hypothesis Compute$(C', V[\ \vec{U}^m\ /\ \vec{X}^m\ ])$ takes time $c|C'|^{k'}\ |S|^{k'-1}|T|$, hence the total cost is

$$\begin{aligned}
& c|\vec{B}'|^{\max\{k'_i|1\leq i\leq m\}+1}|S|^{\max\{k'_i|1\leq i\leq m\}}|T| + c|C'|^{k'}|S|^{k'-1}|T| \\
& \leq c|A|^k|S|^{k-1}|T| \\
& \qquad \text{as } k = \max\{k', 1+\max\{k'_i \mid 1 \leq i \leq m\}\} \\
& \qquad \text{and } |C'| + |\vec{B}'| \leq |A|
\end{aligned}$$

Throughout the algorithm, we have assumed that maximal, closed, and top μ/ν-subassertions of A can be detected in time $O(|A|)$ and therefore does not increase the overall complexity. This is justified by the assumption that variables are represented by elements of an interval of integers so that sets of variables can be represented effectively by a function into $\mathbb{O}$ - a 'bitvector'. □

5.5 Other Global Algorithms

The algorithm of the previous section only assumes the presence of an efficient algorithm for handling the unnested case, and then by applying this at appropriate places handles the general case. Boolean graphs are not used, except perhaps in the base-case. Another attempt of extending the global algorithm to the full modal μ-calculus, could be through a generalization of the graph-like ideas of 'Chasing 1's'. Let us concentrate on an alternation depth two formula with one minimum and one maximum fixed-point:

$$\begin{aligned}
& X_i \\
& \texttt{where}_\mu\ \vec{X} = \vec{B} \\
& \qquad\qquad \texttt{where}_\nu\ \vec{Y} = \vec{C}
\end{aligned}$$

```
Compute($A, V$):            ($A$ closed, $V$ a valuation)

Let $B_1, \ldots, B_m$ be the maximal, closed, proper $\mu/\nu$-subassertions of $A$.
if $m > 0$ then
    Replace $B_1, \ldots, B_m$ in $A$ with new constants $Q_1, \ldots, Q_m$ yielding $A'$.
    $V' := V[\text{Compute}(B_1, V)/Q_1, \ldots, \text{Compute}(B_m, V)/Q_m]$.
    return Compute($A', V'$)
ff $A \equiv Q$ then
    return $V(Q)$
ff $A \equiv \langle a \rangle B$ then
    return $\{s \in S \mid \exists s' \in S.\ s \xrightarrow{a} s' \ \&\ s' \in \text{Compute}(B, V)\}$
ff $A \equiv B_0 \wedge B_1$ then
    return Compute($B_0, V$) $\cap$ Compute($B_1, V$)
ff $A \equiv (C \ \mathtt{where}_\mu\ \vec{X}^n = \vec{B}^n)$ then
    Let $X_{n+1}, \ldots, X_m$ be the variables bound by top $\mu$-subassertions of
    $\vec{B}^n$ and $C$, and let $B_1, \ldots, B_m$ be the right-hand sides corresponding to
    $X_1, \ldots, X_n, X_{n+1}, \ldots, X_m$.[a]
    Define
        $B'' \equiv C' \ \mathtt{where}_\mu \begin{pmatrix} X_1 \\ \vdots \\ X_m \end{pmatrix} = \begin{pmatrix} B'_1 \\ \vdots \\ B'_m \end{pmatrix}$,
    where $B'_i$ is constructed from $B_i$ by replacing all occurrences of top
    $\mu$-subassertions ($D \ \mathtt{where}_\mu\ \vec{Y} = \vec{C}$) by $D$ and similarly for $C'$
    constructed from $C$.
    if $B''$ is an unnested fixed-point then
        Compute the minimum fixed-point $\vec{X} = \vec{B'}$ using the efficient
        algorithm for unnested fixed-points and evaluate $C'$ with the
        resulting valuation.
    else
        $\vec{U}^m := (\emptyset, \ldots, \emptyset)$
        View the variables $X_i$ as constants.
        repeat
            $\vec{U_0}^m := \vec{U}^m$
            $\vec{U}^m := \text{Compute}((B'_1, \ldots, B'_m), V[\vec{U_0}^m / \vec{X}^m])$
        until $\vec{U}^m = \vec{U_0}^m$
        return Compute($C', V[\vec{U}^m / \vec{X}^m]$)
ff All remaining cases are analogous.
fi

Compute(($A_1, \ldots, A_l$), $V$) = (Compute($A_1, V$), ..., Compute($A_l, V$))
```

[a] Here variables should be thought of as identifying occurrences instead of just names, or equivalently all variables should be assumed to have different names.

Figure 5.4: Global algorithm for $\mu K_{\mathtt{where},Q}$ given as a recursive function; '`ff`' means '`else if`' ...

where $\vec{B}$ contains free occurrences of variables from $\vec{Y}$ and $\vec{C}$ contains free occurrences of variables from $\vec{X}$ and $\vec{Y}$. Having performed the translation into a boolean fixed-point expression we end up with:

$$\begin{array}{l} x_i \\ \texttt{where}_\mu\ \vec{x}^{\,n} \ = \ \vec{b}^{\,n} \\ \qquad\qquad \texttt{where}_\mu\ \vec{y}^{\,m} \ = \ \vec{c}^{\,m} \end{array}$$

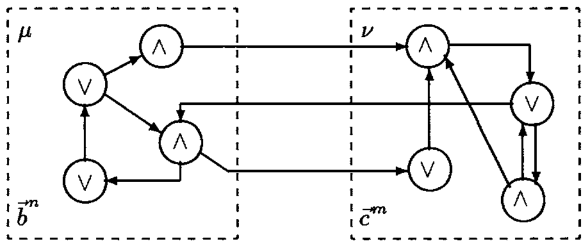
where $\vec{b}^{\,n}$ and $\vec{b}^{\,n}$ has free variables in $V = V_x \cup V_y$ where $V_x = \{x_1, \dots, x_n\}$ and $V_y = \{y_1, \dots, y_m\}$. Ignoring that we are only interested in the i'th component we are really involved with evaluating the expression

$$e = \mu x.f(x, \nu y.g(x, y)) \tag{5.7}$$

where $f : \mathbb{O}^{V_x} \times \mathbb{O}^{V_y} \to \mathbb{O}^{V_x}$ is the function induced by $\vec{b}^{\,n}$ and $g : \mathbb{O}^{V_x} \times \mathbb{O}^{V_y} \to \mathbb{O}^{V_y}$ is the function induced by $\vec{c}^{\,m}$.

Method 1

Pictorially, we now have two graphs connected by some edges, for instance:



The idea of the algorithm of the previous section is essentially to compute $e \in \mathbb{O}^{V_x}$ by an increasing sequence of approximations $x_0 \sqsubseteq x_1 \sqsubseteq \dots$ in $\mathbb{O}^{V_x}$ defined by

$$\begin{aligned} x_0 &= 0 \\ x_{i+1} &= f(x_i, \nu h_i) \\ &\qquad \text{where } h_i(y) = g(x_i, y)) \end{aligned} \tag{5.8}$$

In each step from approximation number i to number $i+1$ the maximum fixed-point $\nu y.h_i(y)$ is computed with the efficient algorithm 'Chasing O's' running in time $c_2|\vec{c}^{\,m}|)|$. Therefore the running time is determined by the length of the approximation sequence i.e. $|\{x_i\}_{i\in\omega}|$ or in other words the least i such that $x_i = x_{i-1}$. The total running time for this method is then

$$(c_1|\vec{b}^{\,n}| + c_2|\vec{c}^{\,m}|)|\{x_i\}_{i\in\omega}|$$

as the time for computing one application of f is proportional to the size of the boolean graph representing f. Notice, that $|\{x_i\}_{i\in\omega}|$ is bounded by n, height of the lattice $\mathbb{O}^{V_x}$.

For the model checking problem this gives running time

$$O((|\vec{B}||T| + |\vec{C}||T|)|\vec{B}||S|) = O(|A|^2|S||T|).$$

Method 2

One way of improving upon this is to try to make bigger steps thereby reaching the fixed-point in fewer – and not more expensive steps. Here is one way of doing it. Consider the sequence:

$$\begin{aligned} u_0 &= 0 \\ u_{i+1} &= \mu z.f(u_i \vee z, \nu h_i) \qquad (5.9)\\ &\qquad \text{where } h_i(y) = g(u_i, y)) \end{aligned}$$

Again we can compute νh by 'Chasing 0's' in time $c_2|\vec{c}^{\,m}|$ but in each step we compute the least fixed-point $\mu z.f(u_i \vee z, \nu h_i)$ starting from the previous approximation and using 'Chasing 1's.' As the u_i sequence is increasing, each computation of νh must be started from scratch taking time $c_2|\vec{c}^{\,m}|$ at each step, but as the arguments to the minimum fixed-point in u_{i+1} are increasing we can reuse the earlier configuration and just restart 'Chasing 1's' with the newly increased values (i.e. by properly decreasing strengths and adding to A), yielding a total cost for the minimum fixed-points of $c_1|\vec{b}^{\,n}|)|$. Hence the total cost is

$$c_1|\vec{b}^{\,n}|)| + c_2|\vec{c}^{\,m}|)||\{u_i\}_{i\in\omega}|.$$

For the correctness of using the u_i sequence, it is not difficult to see by mathematical induction that for all i,

$$x_i \leq u_i \leq e$$

and therefore taking least upper bounds,

$$e = \sqcup x_i \leq \sqcup u_i \leq e$$

implying $\sqcup u_i = e$. Notice, that the sequence $\{u_i\}_{i \in \omega}$ is never longer than the sequence $\{x_i\}_{i \in \omega}$ and *possibly very much shorter.*

Method 3

In order to increase the steps even more, we could consider the following sequence where we are increasing the ν-part by using a minimum fixed-point (!):

$$\begin{array}{rcl} v_0 & = & 0 \\ v_{i+1} & = & \mu z.f(v_i \vee z, \mu z'.g(v_i \vee z, \nu h_i \vee z')) \\ & & \quad \text{where } h_i(y) = g(v_i, y)) \end{array} \tag{5.10}$$

It is straightforward to see that this yields an increasing sequence and by mathematical induction that $u_i \leq v_i$.

Now, we will argue that $v_i \leq e$ by induction on i. The base case is trivial. Hence, assume that $v_i \leq e$. Then

$$\begin{array}{rcl} v_{i+1} & = & \mu z.f(v_i \vee z, \mu z'.g(v_i \vee z, \nu h_i \vee z')) \\ & \leq & \mu z.f(v_i \vee z, \mu z'.g(v_i \vee z, (\nu y.g(v_i \vee z, y)) \vee z')) \\ & & \quad \text{using monotonicity, noting that } \nu h_i \leq \nu y.g(v_i \vee z, y) \\ & \leq & \mu z.f(v_i \vee z, \nu z'.g(v_i \vee z, (\nu y.g(v_i \vee z, y)) \vee z')) \\ & & \quad \text{as } \mu f \leq \nu f \text{ for all } f \\ & = & \mu z.f(v_i \vee z, \nu y.g(v_i \vee z, y)) \\ & & \quad \text{collapsing the two } \nu'\text{s to one by lemma 5.2 below} \\ & = & \mu z.f(z, \nu y.g(z, y)) = e \\ & & \quad \text{replacing } v_i \vee z \text{ by } z \text{ as justified by lemma 5.1 below since } v_i \leq e. \end{array}$$

Hence

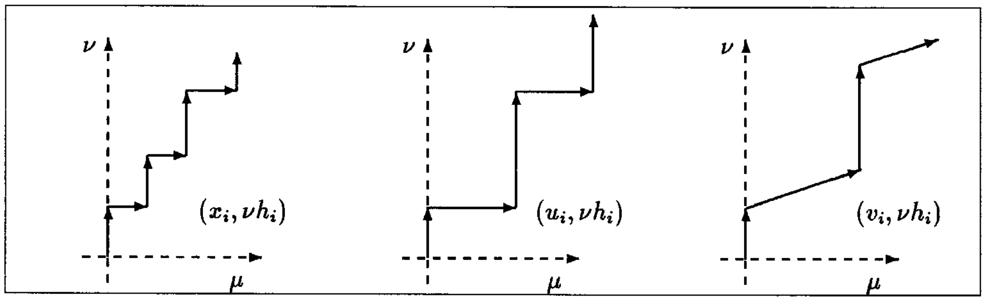


Figure 5.5: The three approximation sequences. Horizontal arrows indicate increases in the 'μ'-approximants x_i, u_i, and v_i, whereas vertical arrows indicate increases in the ν-parts $\nu \mathrm{h}_i$. Notice, that for the last method horizontal and vertical moves are combined by the computation of the simultaneous minimum fixed-point of the two parts.

$$x_i \leq u_i \leq v_i \leq e$$

implying that $\sqcup v_i = e$ and that the sequence $\{v_i\}_{i \in \omega}$ is always the shortest. How do we compute each new approximation? Again, we must every time use 'Chasing 0's' to compute the ν-part taking time $c_3|\vec{c}^{\,m}|$. The two minimum fixed-points will have increasing arguments and can be computed simultaneously in each step, yielding a total cost of $c_1|\vec{b}^{\,n}| + c_2|\vec{c}^{\,m}|$. This gives a total cost for computing the solution of

$$c_1|\vec{b}^{\,n}| + c_2|\vec{c}^{\,m}| + c_3|\vec{c}^{\,m}||\{v_i\}_{i \in \omega}| = c_1|\vec{b}^{\,n}| + c_3|\vec{c}^{\,m}|(c_2/c_3 + |\{v_i\}_{i \in \omega}|)$$

In other words, if just $c_2/c_3 + |\{v_i\}_{i \in \omega}| < c_2/c_3|\{u_i\}_{i \in \omega}|$ this last approach is the better. As the constant factors c_2 and c_3 of Chasing 1's and Chasing 0's should be the same c_2/c_3 is 1 and this last methods seems to almost always pay off. Hence, although for the model-checking problem, in terms of worst-case complexities, we have gained nothing – it is still $O(|A|^2|S||T|)$ – in practice the third method could turn out to perform much better on average and at least never worse than the two others! Figure 5.5 illustrates the behaviour of the approximation sequences.

Lemma 5.1 *Let f be an ω-continuous function on a complete lattice D. Assume $\dot{x} \in D$ is an element satisfying $\dot{x} \leq \mu f$. Then*

$$\mu x.f(\dot{x} \vee x) = \mu f$$

and dually if f is ω-anticontinuous and $\dot{y} \geq \nu f$ then

$$\nu y.f(\dot{y} \wedge y) = \nu y$$

Proof: Take the sequence $x_0 = 0, x_{i+1} = f(\dot{x} \vee x_i)$ which surely is below μf and bigger than the usual approximation sequence for f, hence must have least upper bound μf. □

Lemma 5.2 *For a monotonic function f on a complete lattice D we have*

$$\nu x.f(\nu f \vee x) = \nu f.$$

Proof: Easy, using e.g. the second reduction lemma (lemma 7.1). □

5.6 A Local Algorithm

Model checking is usually involved with deciding satisfaction for just one particular state, so it might seem overwhelming to have to compute the complete fixed-point in order to decide the value at just one particular state. This observation is central to the development of *local* model checkers with the idea being that starting from one particular state, only a 'necessary' part of the transition system will be investigated in order to determine satisfaction.

In this section we present a local algorithm for finding a fixed-point of a normalized, simple equation system, which will only visit a subset of the system in the search for deciding the minimum fixed-point value for one particular node. We will explain how it can be implemented to run in time proportional to (within a logarithmic factor) the size of the subset being visited.

First, we need to review the problem. We want to find the value of an expression

$$x_i \ \mathtt{where}_\mu \ \vec{x}^{\,n} \ = \ \vec{b}^{\,n}$$

where $\vec{x}^{\,n} \ = \ \vec{b}^{\,n}$ is a normalized, simple equation system with free variables $V = \{x_1, \ldots, x_n\}$ inducing a function $\vec{f}^{\,n} : \mathbb{O}^n \rightarrow \mathbb{O}^n$ in the obvious way:

$$\vec{f}^{\,n}\ (\vec{u}) = [\![\ \vec{b}^{\,n}\ ]\!]\rho[\vec{u}/\ \vec{x}^{\,n}\ ]$$

In view of the trivial isomorphism $\iota : \mathbb{O}^n \cong \mathbb{O}^V$ between a product lattice of $\mathbb{O}$ and a function space into $\mathbb{O}$ ordered pointwise, the function $\vec{f}^{\,n}$ can also be viewed as a function $f : \mathbb{O}^V \to \mathbb{O}^V$ i. e.

$$f(u) = \iota(\ \vec{f}^{\,n}\ (\iota_x^{-1}(u)))$$

We will distinguish between the two views by the presence or absence of the vector arrow.

The function space $\mathbb{O}^V$ is a kind of *variable environment* with domain V, hence the fixed-point $\mu u.f(u,v)$ is an environment giving the values of the variables $x_1, \ldots, x_n$ in the minimum solution to $\vec{x}^{\,n} = \vec{b}^{\,n}$.

To state and reason about the local algorithm, we will be faced with situations in which the value of some of the variables are unknown, and we therefore must be quite explicit in our treatment of ‘unknown values’. Hence, we model an unknown value by the symbol ? and use a lifting-construction well-known from domain theory to add an unknown value to a lattice.

Definition 5.3 The ?-*lifting* $D_?$ of a poset D is the poset

$$D_? = \{?\} \cup \{\lfloor d \rfloor \mid d \in D\}$$

with ordering $\leq_D$ defined by

$$x \leq_{D_?} y \ \Leftrightarrow_{\mathrm{def}}\ x = ?\ \text{or}\ \exists a, b \in D.\ x = \lfloor a \rfloor\ \&\ y = \lfloor b \rfloor\ \&\ a \leq_{D_?} b$$

Hence ? is a ‘bottom element’ of $D_?$ (The function $\lfloor\ \rfloor$ is any injective function without ? in its image.) □

The ?-lifting of a complete lattice is again a complete lattice. Notice, that $\mathbb{O}^n$ can be embedded into $(\mathbb{O}_?)^n$ by mapping $(u_1, \ldots, u_n)$ to $(\lfloor u_1 \rfloor, \ldots, \lfloor u_n \rfloor)$. We will, somewhat ambiguously, use $\lfloor\ \rfloor$ for this embedding. Similarly, for any set D, $\mathbb{O}^D$ can be embedded into $(\mathbb{O}_?)^D$ by pointwise ?-lifting, a map which we also use $\lfloor\ \rfloor$ to denote. Moreover, for $u \in (\mathbb{O}_?)^D$ we take $\mathrm{dom}(u)$ to be the *domain of* u, i.e. $\mathrm{dom}(u) = \{d \in D \mid u(d) \neq ?\}$.

Now, the boolean expression $\vec{b}^{\,n}$ with free variables $V = \{x_1, \ldots, x_n\}$ induces a function $\vec{f'}^{\,n} : (\mathbb{O}_?)^n \to (\mathbb{O}_?)^n$ by extending $\bigvee$ and $\bigwedge$ to maps $\bigvee : \mathcal{P}(\mathbb{O}_?) \to \mathbb{O}_?$ and $\bigwedge : \mathcal{P}(\mathbb{O}_?) \to \mathbb{O}_?$ as follows:

$$\bigvee C = \begin{cases} \lfloor 1 \rfloor & \text{if } \lfloor 1 \rfloor \in C \\ \lfloor 0 \rfloor & \text{if } C \subseteq \{\lfloor 0 \rfloor\} \\ ? & \text{if } ? \in C \ \& \ \lfloor 1 \rfloor \notin C \end{cases}$$

$$\bigwedge C = \begin{cases} \lfloor 1 \rfloor & \text{if } C \subseteq \{\lfloor 1 \rfloor\} \\ \lfloor 0 \rfloor & \text{if } \lfloor 0 \rfloor \in C \\ ? & \text{if } ? \in C \ \& \ \lfloor 0 \rfloor \notin C \end{cases}$$

Again we refer to the function-space version of $\vec{f'}^{\,n}$ as simply f' which is a map $f' : (\mathbb{O}_?)^V \to (\mathbb{O}_?)^V$.

The following fact is easily proven (see e.g. section 6.3.1):

$$\forall \vec{v} \in \mathbb{O}^m.\mu\vec{v}.\ \vec{f'}^{\,n}\ (\vec{u} \vee \lfloor \vec{0} \rfloor, \lfloor \vec{v} \rfloor) = \lfloor \mu\vec{v}.\ \vec{f'}^{\,n}\ (\vec{u}, \vec{v}) \rfloor \tag{5.11}$$

This, somewhat trivial relationship, will be used by our algorithm: By partially finding the fixed-point on the left-hand side we also gain knowledge about the fixed-point on the right-hand side. This will be based on a notion of relativized equivalence and relativized partial order on functions.

Definition 5.4 Assume D is a set and E a poset. If u and v are functions $u, v : D \to E$ then define for all $S \subseteq D$,

$$\begin{array}{lll} u =_S v & \Leftrightarrow_{\text{def}} & \forall d \in S.\ u(d) = v(d) \\ u \leq_S v & \Leftrightarrow_{\text{def}} & \forall d \in S.\ u(d) \leq_E v(d) \end{array}$$

□

In particular we have relativized equivalences and partial orders on the partial environments $\mathbb{O}^V$, which due to the isomorphism ι induces relativized equivalences and partial orders on $\mathbb{O}^n$. We now have for minimum fixed-points:

Lemma 5.3 (Projection lemma) *Suppose D and E are cpo's with bottoms. Let $p : E \to D$ be a surjective, ω-continuous function. For any ω-continuous function $f : E \to E$ and element $y \in D$, which satisfy*

$$\begin{array}{rl} (i) & \forall x \in p^{-1}(y).p(f(x)) = y \\ (ii) & y \leq p(\mu x.f(x)) \end{array}$$

we have

$$y = p(\mu x.f(x)).$$

Proof: For any $x \in p^{-1}(y)$ it is easy to show by induction on $n \in \omega$ from the monotonicity of p and (i) that $p(f^n(\perp_E)) \leq p(f^n(x)) = y$. Then by continuity of p, $p(\mu x.f(x)) = p(\sqcup_{n\in\omega} f^n(\perp_E)) = \sqcup_{n\in\omega} p(f^n(\perp_E)) \leq y$, and the lemma follows from (ii). $\square$

As an easy corollary of the projection lemma we have:

Lemma 5.4 (Partial fixed-point lemma) *Let I be an indexing set and for each $i \in I$, E_i a cpo, and let $E = \prod_{i\in I} E_i$ be the product cpo of the E_i's ordered pointwise. Assume $f : E \to E$ is ω-continuous and that $u \in E$. Then for all $S \subseteq I$, if*

$$\begin{array}{rl} (i) & \forall u'.u =_S u' \Rightarrow u =_S f(u'), \textit{ and} \\ (ii) & u \leq_S \mu f \end{array}$$

then

$$u =_S \mu f.$$

Proof: Let $D = \Pi_{i\in S} E_i$. Define $p : E \to D$ by $p(m)(s) = m(s)$ which is easily seen to be surjective and ω-continuous. Notice, that $m =_S m' \Leftrightarrow p(m) = p(m')$. Then, if $x \in p^{-1}(p(u))$, i.e. $p(x) = p(u)$, we have $p(f(x)) = p(u)$ by (i). Moreover, $u \leq_S \mu f$ implies $p(u) \leq p(\mu f)$, hence from the projection lemma we get $p(u) = p(\mu f)$, i.e. $u =_S \mu f$. $\square$

The local algorithm can be found in figure 5.6.

In the presentation of the algorithm we leave out the lifting operations for improved readability. The variables of the algorithm serve the following purposes:

The task of the algorithm is to find a partial environment m, giving partially the fixed-point of the function f induced by $\vec{b}^{\,n}$. The set A is a set of variables that have to be recomputed either because their value is needed by some other variable, or because the value of one of the successors have changed. The partial map d associates to each variable on which it is defined a set of its predecessors that need to be 'informed' if the variable change value. The partial map p associates to each variable a number indicating the next successor to be investigated, assuming that in each right-hand side, $\bigvee \chi_j$ or$\bigwedge \chi_j$ the variables of χ_j are numbered from 1 to $|\chi_j|$. All successors with index less than the current value of p has defined values in m. Finally, the partial map h indicates for each variable on which it is defined the number of successors (with index below that given by p) which has value 1.

Input: A variable x_i and a normalized, simple equation system $\vec{x}^n = \vec{b}^n$ where $\vec{b}^n$ has free variables in $V = \{x_1, \ldots, x_n\}$ and induces the function $f : \mathbb{O}^V \to \mathbb{O}^V$.
Output: An environment $m : (\mathbb{O}_?)^V$ such that $x_i \in \mathrm{dom}(m)$ and $m =_{\mathrm{dom}(m)} \lfloor \mu f \rfloor$.

```
        for all $v \in V$ do  $m(v) := ?$  $d(v) := ?$  $p(v) := ?$  $h(v) := ?$ od
1:      $A := \{x_i\}$  $m(x_i) := 0$  $d(x_i) := \emptyset$  $p(x_i) := 0$  $h(x_i) := 0$
        while $A \neq \emptyset$ do
            pick an $x_j \in A$  $A := A \setminus \{x_j\}$
            $r := eval(b_j, h(x_j), p(x_j))$
            if $r = ?$ then
2:              $p(x_j) := p(x_j) + 1$
                if $m(\chi_{jp(x_j)}) = ?$ then
                    $m(\chi_{jp(x_j)}) := 0$  $d(\chi_{jp(x_j)}) := \{x_j\}$
                    $p(\chi_{jp(x_j)}) := 0$  $h(\chi_{jp(x_j)}) := 0$
3:                  $A := \{\chi_{jp(x_j)}, x_j\} \cup A$
                else
4:                  $d(\chi_{jp(x_j)}) := \{x_j\} \cup d(\chi_{jp(x_j)})$  $A := \{x_j\} \cup A$
5:                  if $m(\chi_{jp(x_j)}) = 1$ then $h(x_j) := h(x_j) + 1$fi
                fi
6:          else  if $r = 1$ & $m(x_j) = 0$ then
                $m(x_j) := 1$
7:              for all $v \in d(x_j)$ do $A := \{v\} \cup A$  $h(v) := h(v) + 1$ od
            fi
        od
```

Figure 5.6: The local algorithm for (x_i where$_\mu$ $\vec{x}^n = \vec{b}^n$).

The algorithm starts with a particular variable x_i assumes it has value 0 and tries to verify this by evaluating the right-hand of x_i, adding other variables to A if their values are needed, and tries to verify that these have value 0 and so on. In doing so p keeps track of which successors of a variable have been visited and d dynamically keeps track of the dependencies between variables. If a variable change value from 0 to 1 this change is propagated to all dependent variables stored in d.

The new value of a variable, $eval(b,h,p)$, is computed as

$$eval(b,h,p) = \begin{cases} 1 & \text{if } (b \equiv \bigvee \chi \;\&\; h > 0) \\ & \text{or } (b \equiv \bigwedge \chi \;\&\; h = p = |\chi|) \\ 0 & \text{if } (b \equiv \bigvee \chi \;\&\; h = 0 \;\&\; p = |\chi|) \\ & \text{or } (b \equiv \bigwedge \chi \;\&\; h < p) \\ ? & \text{otherwise.} \end{cases}$$

Theorem 5.4 *When the local algorithm of figure* 5.6 *terminates, we have* $\begin{array}{ll} (i) & x_i \in \mathrm{dom}(m) \textit{ and} \\ (ii) & m =_{\mathrm{dom}(m)} \lfloor \mu f \rfloor \end{array}$ *It can be implemented to run in time proportional to* $|\vec{b'}|\log|\mathrm{dom}(m)|)$ *where* $\vec{b'}$ *is the part of* $\vec{b}$ *examined by the algorithm, hence the running time is* $O(|\vec{b}|\ \log n)$.

Proof: Correctness is postponed to after the proof of correctness for the more general algorithm in chapter 6 (see lemma 6.1 in 6.5).

For the complexity we assume that the partial maps m, d, p and h are implemented as e.g. balanced search trees such that extensions and alternations of maps can be performed in logarithmic time. The sets of variables A could be implemented as a stack giving constant-time insertions and deletions. We use an amortized cost argument by counting the number of insertions to A and observe that between any two insertions and deletions only a constant number of logarithmic time operations are performed.

Now, considering each variable at a time, how many times can x_j be inserted into A? First, perhaps once in line (1:). Secondly, due to line (2:), incrementing $p(x_j)$, x_j is at (3:) and (4:) at most added to A a number of times bounded by $|\chi_j|$ and it is at most added as the successor $\chi_{ip(x_i)}$ of some x_i once since the value at x_j is then changed from ? to 0. Thirdly, due to the condition of (6:) and the increase of $m(x_j)$ to one, line (7:) is at most executed once, adding to A at most $|d(x_j)| \leq |Pred(x_j)|$ times. Summing over all variables x_j we get a total number of insertions that is $O(|\vec{b}|)$ (or more precisely $c|\vec{b'}|$ where $\vec{b'}$ is the part of $\vec{b}$ being examined by the algorithm and c is some constant).

Hence, we have a total number of insertions and deletions on A bounded by $O(|\vec{b}|)$ giving the bound $O(|\vec{b}|\ \log n)$. □

5.7 A Local Algorithm for Alternating Fixed-Points

As a step towards a local algorithm for computing alternating fixed-points, we refine the algorithm of the previous section, such that it computes the value of the expression

$$x_i \; \texttt{where}_\mu \; \vec{x}^{\,n} \;=\; \vec{b}^{\,n}$$

where $\vec{b}^{\,n}$ contains not only the free variables $V_x = \{x_1, \ldots, x_n\}$ but also the free variables $V_y = \{y_1, \ldots, y_m\}$, without an initial values for the variables of V_y. These values will be demanded in a lazy fashion, as they become needed. We implement this by a data-structure – a 'Mu-Component' – which is essentially the previous local algorithm with proper interruptions for demanding the values of variables in V_y and operations for supplying these values and restarting the evaluation.

Dually, a *Nu-Component* will compute a maximum fixed-point in the same way as a Mu-Component computes a minimum fixed-point. A Mu- and a Nu-Component will later together form the basis of an algorithm computing alternating fixed-points in a local fashion.

5.7.1 A Mu-Component

Now, the boolean expression $\vec{b}^{\,n}$ with free variables $V = V_x \cup V_y$ where $V_x = \{x_1, \ldots, x_n\}$ and $V_y = \{y_1, \ldots, y_m\}$ induces a function $\vec{f}^{\,n} : \mathbb{O}^n \times \mathbb{O}^m \to \mathbb{O}^n$ in the obvious way:

$$\vec{f}^{\,n}(\vec{u}, \vec{v}) = [\![\; \vec{b}^{\,n} \;]\!] \rho[\vec{u}/\; \vec{x}^{\,n}, \vec{v}/\; \vec{y}^{\,m}\;]$$

As before, in view of the trivial isomorphisms $\iota_x : \mathbb{O}^n \cong \mathbb{O}^{V_x}$ and $\iota_y : \mathbb{O}^m \cong \mathbb{O}^{V_y}$ between product lattices of $\mathbb{O}$ and function spaces into $\mathbb{O}$ ordered pointwise, the function $\vec{f}^{\,n}$ can also be viewed as a function $f : \mathbb{O}^{V_x} \times \mathbb{O}^{V_y} \to \mathbb{O}^{V_x}$ i.e.

$$f(u, v) = \iota_x(\; \vec{f}^{\,n} \; (\iota_x^{-1}(u), \iota_y^{-1}(v))).$$

Furthermore, $\vec{b}^{\,n}$ also induces a function $\vec{f'}^{\,n} : (\mathbb{O}_?)^n \times (\mathbb{O}_?)^m \rightarrow (\mathbb{O}_?)^n$ and a function $f' : (\mathbb{O}_?)^{V_x} \times (\mathbb{O}_?)^{V_y} \rightarrow (\mathbb{O}_?)^{V_x}$ using the extensions of $\bigvee$ and $\bigwedge$ defined previously.

Now, the data-structure will maintain a partial environment $m : V \rightarrow \mathbb{O}_?$ with the two parts $m_x : V_x \rightarrow \mathbb{O}_?$ and $m_y : V_y \rightarrow \mathbb{O}_?$ storing information about the 'current' value . of variables. The data-structure (see figure 5.7.1) is operated through the four operations *init, find, set,* and *update* which has intuitive behaviour as follows:

init (). We use a program variable $R \subseteq V_y$ recording the variables of V_y for which the values have been requested. All program variables are initialized properly.

find($\mathbf{x_j}$) $\rightarrow$ y. The variable x_j is added to the set of variables for which the value must be determined, and the evaluation is continued. The returned value y is either $\bullet$ implying that $A = \emptyset$, in which case the component is said to be *stable* and the partial fixed-point has been found, or y is a variable from V_y, the value of which must be supplied.

set($\mathbf{y}, \mathbf{b}$). Sets the value of the variable y to b. Notice, that it is required that the previous value of y was *smaller or the same* reflecting the fact that in a Mu-Component it is easy to propagate increases in values, but not decreases.

update() $\rightarrow$ y. Restarts the evaluation, for instance, after a requested variable has been set, and returns a y as 'find' above.

Theorem 5.5 (Correctness of Mu-Component) *Let K be a Mu-Component for $\vec{x}^{\,n} = \vec{b}^{\,n}$ with free variables $V_x = \{x_1, \ldots, x_n\}$* and $V_y = \{y_1, \ldots, y_m\}$ *inducing the function $f' : (\mathbb{O}_?)^{V_x \cup V_y} \rightarrow (\mathbb{O}_?)^{V_x}$. If, after having performed an init and any sequence of update, find(x_i) and legal set(y_j, b) operations, K is stable (i.e. $A = \emptyset$) then, letting S be the set of variables used as arguments to find, we have,*

$$\begin{array}{rl} (i) & S \subseteq \mathrm{dom}(m_x) \\ (ii) & \forall v \in (\mathbb{O}_?)^{V_y}.\ v =_R m_y \Rightarrow m_x =_{\mathrm{dom}(m_x)} \mu u.f'(u \vee \lfloor 0 \rfloor, v) \end{array}$$

Proof: Line (i) is obvious from the behaviour of find. Line (ii) is postponed to after the presentation of the general fixed-point finder in chapter 6 (see lemma 6.2 in section 6.5). □

The theorem states that, when a Mu-Component is stable all variables requested by find has the 'correct values' as given by the minimum fixed-point relative to m_y. Moreover, the values are independent of whatever is supplied for the yet unknown values of variables of V_y.

A Nu-Component is constructed like a Mu-Component by dualizing everything: zeroes are replaced by ones, disjunctions by conjunctions etc. Similarly, the ?-lifting $(\ )_?$ and the embedding $\lfloor\ \rfloor$ is replaced by '?-lowering' $(\ )^?$ and the embedding $\lceil\ \rceil$. By the principle of duality we immediately get:

Corollary 5.3 (Correctness of Nu-Component) *Let L be a Nu-Component for $\vec{y}^{\,m} = \vec{c}^{\,m}$* with free variables *$V_y = \{y_1, \ldots, y_m\}$ and $V_x = \{x_1, \ldots, x_n\}$ inducing the function $g' : (\mathbb{O}^?)^{V_x} \times (\mathbb{O}^?)^{V_y} \to (\mathbb{O}^?)^{V_y}$. If after having performed an init and any sequence of update, $find(y_j)$ and legal set(x_i, b) operations, L is stable (i.e. $A = \emptyset$) then, letting S be the set of variables used as arguments to find we have,*

$$\begin{array}{ll} (i) & S \subseteq \mathrm{dom}(m_y) \\ (ii) & \forall u \in (\mathbb{O}_?)^{V_x}.\ u =_R m_x \Rightarrow m_y =_{\mathrm{dom}(m_y)} \nu u.g'(u, v \vee \lfloor 1 \rfloor, v) \end{array}$$

Theorem 5.6 (Complexity of Mu- and Nu-Components) *A Mu-Component (and a Nu-Component) for the normalized, simple equation system $\vec{x}^{\,n} = \vec{b}^{\,n}$ with m free variables besides $\vec{x}^{\,n}$ can be implemented such that after a call to init any sequence of length N of calls to find, update, 'lookup' $(K.m_x)$ and set takes amortized time*

$$O((N + |\vec{b}^{\,n}|)\log n).$$

Stability can always be reached in fewer than m update-operations.

Proof: The complexity follows easily from the proof of theorem 5.4 – the only difference being that we interrupt the algorithm now-and-then, so still only $O(|\vec{b}^{\,n}|)$ insertions on A are nerformed.

Moreover, after m call to update, each of the m y_j's has (at least) value 0, so no y_j's will be requested and an update-call will end with $A = \emptyset$, i.e. K is stable. $\square$

```
**Variables:** $R \subseteq V_y$, $A \subseteq V_x$, $m : (\mathbb{O}_?)^{V_x \cup V_y}$, $d : (\mathcal{P}(V_x)_?)^V$, $p : (\mathbb{N}_?)^{V_x}$, $h : (\mathbb{N}_?)^{V_x}$

init() :
    for all $v \in V_x \cup V_y$ do  $m(v) := ?$  $d(v) := ?$  $p(v) := ?$  $h(v) := ?$ do
    $R := \emptyset$  $A := \emptyset$

find($x_i$) $\rightarrow y$ :
    if $x_i \notin \mathrm{dom}(m_x)$ then
        $A := A \cup \{x_i\}$  $m(x_i) := 0$  $d(x_i) := \emptyset$  $p(x_i) := 0$  $h(x_i) := 0$
    fi
    $y :=$ update()

set($y, b$) :
    if $m(y) =?$ then $m(y) := b$  $d(y) := \emptyset$
    else if $m(y) = 0$ & $b = 1$ then
        for all $v \in d(y)$ do $A := \{v\} \cup A$  $h(v) := h(v) + 1$ od
    else if $m(y) = 1$ & $b = 0$ then error
    fi

update() $\rightarrow y$ :
    $y := \bullet$
    while $A \neq \emptyset$ & $y = \bullet$ do
        *pick an* $x_j \in A$  $A := A \setminus \{x_j\}$
        $r := eval(b_j, h(x_j), p(x_j))$
        if $r =?$ then
            $p(x_j) := p(x_j) + 1$  $v := \chi_{jp(x_j)}$
            if $v \in V_y$ then $R := \{v\} \cup R$ fi
            $A := \{x_j\} \cup A$
            if $m(v) =?$ then
                $m(v) := 0$  $d(v) := \{x_j\}$  $p(v) := 0$  $h(v) := 0$
                if $v \in V_y$ then $y := v$ else  $A := \{v\} \cup A$ fi
            else
                $d(v) := \{x_j\} \cup d(v)$
                if $m(v) = 1$ then  $h(x_j) := h(x_j) + 1$ fi
            fi
        else if $r = 1$ & $m(x_j) = 0$ then
            $m(x_j) := 1$
            for all $v \in d(x_j)$ do $A := \{v\} \cup A$  $h(v) := h(v) + 1$ od
        fi
    od
```

Figure 5.7: The Mu-Component for $\overset{\rightarrow n}{x} = \overset{\rightarrow n}{b}$ with variables $V_x = \{x_1, \ldots, x_n\}$ and $V_y = \{y_1, \ldots, y_m\}$.

5.7.2 Connecting Two Components

In order to evaluate an expression like

$$\begin{array}{llll} x_i & & & \\ \texttt{where}_\mu\ \vec{x}^{\,n} & = & \vec{b}^{\,n} & \\ & & \texttt{where}_\mu\ \vec{y}^{\,m} & = \ \vec{c}^{\,m} \end{array}$$

in a local fashion, we shall use a Mu-Component K and a Nu-Component L and 'connect' them. Let f be the map induced by $\vec{b}^{\,n}$ and g the map induced by $\vec{c}^{\,m}$ as defined in the previous section. Then we want to compute part of the alternating fixed-point

$$e = \mu u.f(u, \nu v.g(u, v)) \in \mathbb{O}^{V_x}.$$

Now, the central idea will be to locally start searching in the Mu-Component K for f, moving to searches in the Nu-Component L for g whenever an input node from L is needed. If while searching L an input from K will be needed, we check whether the value is already present, if not we *assume that the input has value* 0 thereby

$$\begin{array}{l} \textit{not only approaching the minimum fixed}-\textit{point from below}, \\ \textit{but also having a successive set of maximum fixed}-\textit{points} \\ \textit{approaching the final maximum fixed}-\textit{point from below}. \end{array} \quad (5.12)$$

(In this respect our approach can be seen as a local version of the global method called method 2 in section 5.5.) To be a bit more specific, we only stop searching in L when the component is *stable* and the overall search in K, forcing searches in L, only stops when K is stable, thereby ensuring that the environments of the components really contain a (partial) fixed-point with respect to the current value of input variables. Any false assumptions made in this process must be properly propagated, which might involve re-computation of variables in K and complete re-computation of L.

The algorithm is presented in figure 5.8. The annotations $I^{(1)}, I^{(2)}$ and $I^{(3)}$ refer to invariants used in proving correctness.

In stating the invariants we use the following list of assertions:

```
K.init()  L.init()
y := K.find(x_i)
if y = • then R := ∅ else R := {y} fi
R' := ∅  S := ∅
while R ≠ ∅ do {I^(1)}
          while R ≠ ∅ do {I^(2)}
                    select an y ∈ R  R := R \ y  R' := R' ∪ {y}
                    x := L.find(y)
                    while x ≠ • do {I^(3)}
                              S := S ∪ {x}
                              if K.m_x(x) ≠ ⊥ then L.set(x, K.m_x(x)) else L.set(x, 0) fi
                              x := L.update()
                    od
                    K.set(y, L.m_y(y))
                    y := K.update()
                    if y ≠ • then R := R ∪ {y} fi
          od
          if ∃x ∈ S. L.m_x(x) ≠ K.m_x(x) then
                    R := R'  R' := ∅  S := ∅  L.init()
          fi
od
```

Figure 5.8: The two-components algorithm. Finds the value of x_i.

(i)	$K.m_x \leq_{\mathrm{dom}(K.m_x)} \lfloor e \rfloor$		
(ii)	$L.m_y \leq_{\mathrm{dom}(L.m_y)} \lfloor \nu v.g(v,e) \rfloor$		
(iii)	$R \cup R' = K.R$	(iii')	$R \cup R' = K.R \neq \emptyset$
(iv)	$S = L.R$		
(v)	$R = \emptyset \Rightarrow K$ stable		
(vi)	$R = \emptyset \Rightarrow L.m_x =_S K.m_x$		
(vii)	$K.m_y =_{R'} L.m_y$	(vii')	$K.m_y =_{R' \setminus y} L.m_y$
$(viii)$	$x = \bullet \Rightarrow L$ stable		
(ix)	$K.R \subseteq \mathrm{dom}(L.m_y)$ & $L.R \subseteq \mathrm{dom}(K.mJ_x)$ & $x_i \in \mathrm{dom}(K.m_x)$		

The invariants are now:

$I^{(1)}$ $\Leftrightarrow_{\mathrm{def}}$ (i) & (ii) & (iii) & (iv) & (v) & (vi) & (vii) & (ix) & L stable
$I^{(2)}$ $\Leftrightarrow_{\mathrm{def}}$ (i) & (ii) & (iii') & (iv) & (v) & (vii) & (ix) & L stable
$I^{(3)}$ $\Leftrightarrow_{\mathrm{def}}$ (i) & (ii) & (iii) & (iv) & (vii') & $(viii)$ & (ix) & $r \in R'$

Theorem 5.7 (Correctness of two-components algorithm)

Let K be a Mu-Component for $\vec{x}^{\,n} = \vec{b}^{\,n}$ *and let L be a Nu-Component for* $\vec{y}^{\,m} = \vec{c}^{\,m}$ *. Let $V_x = \{x_1, \ldots, x_n\}$ and $V_y = \{y_1, \ldots, y_m\}$. Then when the algorithm of figure 5.8 terminates,*

$$\begin{array}{ll} (i) & x_i \in \mathrm{dom}(K.m_x) \\ (ii) & K.m_x =_{\mathrm{dom}(K.m_x)} \lfloor \mu x.f(x, \nu y.g(x,y)) \rfloor \end{array}$$

where $f : \mathbb{O}^{V_x} \times \mathbb{O}^{V_y} \to \mathbb{O}^{V_x}$ *and* $g : \mathbb{O}^{V_x} \times \mathbb{O}^{V_y} \to \mathbb{O}^{V_y}$ *are the functions induced by* $\vec{b}^{\,n}$ *and* $\vec{c}^{\,m}$ *.*

Proof: The proof of the validity of the invariants is standard using Hoare logic: To show that $I^{(1)}$ is valid, we first assume that $I^{(2)}$ and $I^{(3)}$ are valid. After the initializations $I^{(1)}$ holds trivially. After an iteration of the loop most conjuncts of $I^{(1)}$ follows easily from $I^{(2)}$. The only non-trivial case is (vi) which follows from (iii') of $I^{(2)}$ and the fact that after the last if-statement, if $R = \emptyset$ then $R' \neq \emptyset$ by (iii') and therefore the conditional must have been false, implying $L.m_x =_S K.m_x$.

The two other invariants are no more difficult utilizing theorem 5.5 and corollary 5.3.

To show that $I^{(1)}$ is strong enough to prove the theorem, let us assume that $I^{(1)}$ is indeed an invariant for the outermost while-loop. Then at termination we have $I^{(1)}$ & $R = \emptyset$. For $e = \mu u.f(u, \nu v.g(u,v))$ we now first deduce as follows:

$$\begin{array}{lll} \lfloor e \rfloor & = & \lfloor \mu u.f(\nu v.g(u,v)) \rfloor \\ & & \qquad \text{by definition} \\ & = & \mu u'.f'(u' \vee \lfloor 0 \rfloor, \lfloor \nu v.g(\lfloor u' \rfloor, v) \rfloor) \\ & & \qquad \text{by (5.11)} \\ & = & \mu u'.f'(u' \vee \lfloor 0 \rfloor, \nu v'.g(u', v \wedge \lfloor 1 \rfloor)) \\ & & \qquad \text{by the dual of (5.11)} \\ & = & \mu u''.\mu u'.f'(u' \vee \lfloor 0 \rfloor, \nu v'.g(u'', v \wedge \lfloor 1 \rfloor)) \\ & & \qquad \text{by simple fixed-point theory} \\ & & \mu u''.h(u'') \\ & & \qquad \text{by taking the right-hand side below } \mu u'' \\ & & \qquad \text{as definition of } h(u''). \end{array}$$

Hence, $\lfloor e \rfloor$ is the minimum fixed-point of h. Moreover, for all $w =_{\mathrm{dom}(K.m_x)} K.m_x$ we deduce

$$\nu v'.g(w, v \wedge \lfloor 1 \rfloor) =_{\mathrm{dom}(L.m_y)} L.m_y$$

from corollary 5.3 which applies as (iv), (vi) and $R = \emptyset$ implies $K.m_x =_{L.R} L.m_x$ which from (ix) implies $w =_{L.R} K.m_x =_{L.R} L.m_x$. Using this, we notice that (iii) and (vii) implies $L.m_y =_{K.R} K.m_y$ and hence using (ix), $L.m_y =_{\mathrm{dom}(L.m_y)} K.m_y$, therefore from theorem 5.5 we get:

$$K.m_x =_{\mathrm{dom}(K.m_x)} h(w).$$

Combining this with

$$K.m_x \leq_{\mathrm{dom}(K.m_x)} \lfloor e \rfloor = \mu h$$

which follows from (i), we get by lemma 5.4 that

$$K.m_x =_{\mathrm{dom}(K.m_x)} \lfloor e \rfloor.$$

Since $x_i \in \mathrm{dom}(K.m_x)$ follows directly from (ix), we have proven the theorem. □

Intuitively, when the algorithm terminates the theorem tells us that everything known about the fixed-point is correct, i.e. all variables that have a known value, have the *correct* value. Of course, when using the algorithm, it is safe to terminate if at a point the node of interest reaches the value one.

Theorem 5.8 (Complexity of two-components algorithm) *The two-components algorithm of figure* 5.8 *can be implemented to run in worst-case time when evaluating the expression*

$$O((|\vec{b}^{\,n}| + n|\vec{c}^{\,m}|)\log(n+m))$$

when evaluating the expression $(x_i\ \texttt{where}_\mu\ \vec{x}^{\,n} = (\vec{b}^{\,n}\ \texttt{where}_\nu\ \vec{y}^{\,m} = \vec{c}^{\,m}))$

Proof: We again use an amortized cost argument. Assume that R and S are implemented as simple list structures allowing constant insertion and removal times and linear time traversals.

First, we note by theorem 5.6 that for the contribution from K and L, it suffices to count the number of initializations of K and L and the number of operations performed between initializations. Secondly, we notice that between any two calls to K and L except for the last if-statement (**if** $\exists x \in S \ldots$), only an input-independent bounded number of constant-time operations are performed. Thirdly, the amortized cost for this last if-statement is bounded by the number of iterations of the outermost while-loop multiplied with the maximum size of S, which is n.

For K only one initialization takes place and the number of *set, lookup* ("$K.m_x$") and *update*-calls is bounded by the number of iterations of the second while-loop. Since a y is never re-entered into R in the second while-loop, at most m y's can be entered into R, hence this loop executes at most m times for each iteration of the outermost loop. Let n' be the number of x's referred to in $\vec{c}^{\,n}$. The outermost while-loop executes at most $2n'$ times, because each iteration requires an $x \in S$ on which K and L disagree of the value, and in the following iterations such an x will never cause disagreement more than at most one more time (if its value increases from zero to one). Finally, the innermost while-loop executes at most n' times for each iteration of the *outermost* while-loop as an x is never re-entered into S between executions of the last if-statement.

Ignoring the constants, this gives:

for K:	$(2n'm + \lvert\vec{b}^{\,n}\rvert) \log n$ because of at most $2n'm$ executions of the body of the second while-loop
for L:	$2n'(n' + m + \lvert\vec{c}^{\,m}\rvert) \log m$ since up to $2n'$ initializations and maximally $n' + m$ calls between each
for last **if**:	$2n'n'$
in total:	$O((\lvert\vec{b}^{\,n}\rvert + n'\lvert\vec{c}^{\,m}\rvert) \log(n+m))$

As $n' \leq n$ we get the bound in the theorem. □

Corollary 5.4 *There exists a local model-checker for assertions on the form* $A \equiv (X\ \texttt{where}_\mu\ \vec{X} = (\vec{B}\ \texttt{where}_\nu\ \vec{Y} = \vec{C}))$ *running in worst-case time*

$$O(|A|^2|S||T| \log(|A||S|)$$

for a finite transition system T with states S.

Proof: Use the translation in section 5.1 and apply the two-components algorithm. The correctness and complexity now follows immediately from theorem 5.7 and theorem 5.8. □

Remark 5.2 The choice of assuming that undefined 'outputs' from K have value 0 is crucial to the validity of $K.m_x \leq_{\mathrm{dom}(K.m_x)} \lfloor e \rfloor$ (assertion (i) in the invariants) stating that $K.m_x$ is always smaller than the needed result. Taking the value 1 could result in this property being violated. Now, alternatively one might ask: Why, when an output s from K is needed, do we not just start searching for it in K? The reason for the failure of this approach is intricate: When L is not stable, $L.m_y$ might be *bigger* than the true result for this component. (Recall that L is a Nu-Component, and approaches the maximum fixed-point from above.) This in turn can make $K.m_x$ be too big, and the result wrong.

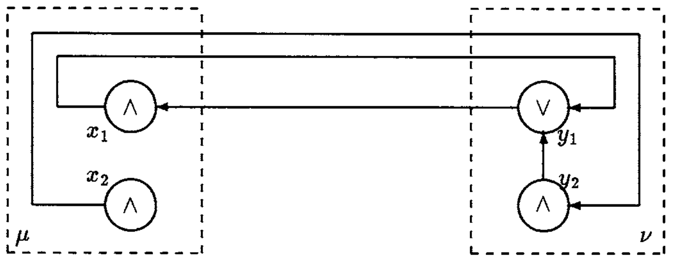


Figure 5.9: An intricate example. What is the correct value of x_2?.

Figure 5.9 shows in a graphical manner a simple example illustrating the point. Assume we are interested in determining the value of x_2. As we are in a Mu-Component we assume that x_2 has value 0, and investigate the son y_2 to try to verify this. Now, as y_2 belongs to a Nu-Component we assume it has value 1, and we find the value of y_1, which is also assumed to be 1 and depends on x_1. Contrary to the algorithm which at this point simply assumes that x_1 has value 0 and continues with L, we will proceed the search in K, trying to verify that the assumption of 0 is correct. Hence, we investigate the single son of x_1, which is y_1 and discovers that y_1 already is defined, with value 1, so x_1 gets the value 1, thereby confirming that y_1 and y_2 are indeed 1, and also x_2 must be changed to 1. Yielding the result that all nodes have value 1.

This is wrong! The fixed-point we are computing is really the very trivial one

$$\begin{aligned} x_2\ \texttt{where}_\mu\ (x_1,x_2) = (y_1,y_2) \\ \texttt{where}_\mu\ (y_1,y_2) = (x_1,y_1) \end{aligned}$$

which is easily seen to have solution $(x_1,x_2)=(0,0)$ by computing approximants. The problem is the cycle $x_1 - y_1$, which in this case is very obvious and perhaps could be handled properly, but in general much more complicated cycles could be present.

It is, however, possible that by being careful, some searching in K is safe — if it does not involve inspecting output from L, which is too big. This, however, requires a more refined algorithm, and will not be further investigated here. □

5.7.3 Extensions

In order to extend the two-components algorithm to a model checker for the full alternation depth two case, we must use some properties of the definition of alternation depth. We claim that any closed assertion of alternation depth two can be transformed to an assertion

$$\begin{aligned}
((\ldots(A\ \texttt{where}_\mu\ \vec{X}_1 = \vec{B}_1 & \\
& \texttt{where}_\nu\ \vec{Y}_1 = \vec{C}_1) \\
\texttt{where}_\nu\ \vec{X}_2 = \vec{B}_2 & \\
& \texttt{where}_\mu\ \vec{Y}_2 = \vec{C}_2) \\
\vdots \qquad & \\
\texttt{where}_\mu\ \vec{X}_k = \vec{B}_k & \\
& \texttt{where}_\nu\ \vec{Y}_k = \vec{C}_k)
\end{aligned}$$

where A, $\vec{B}_i$, and $\vec{C}_i$ contain no fixed-points. Actually, this is an easy consequence of Bekič's theorem and the definition of alternation depth: If we start with a closed assertion A_0 we can take out all the top μ-subassertions (recall, that this is μ-subassertions that are not contained in any ν-subassertion) and get an assertion

$$A_1 \texttt{ where}_\mu\ \vec{X}_1 = \vec{B}_1$$

where A_1 and $\vec{B}_1$ only contain μ-assertions inside ν's. Hence, take out all the top ν-assertions of A_1 and $\vec{B}_1$ which does not contain variables from $\vec{X}_1$ to get the assertion

$$\begin{array}{l}(A_2 \texttt{ where}_\mu\ \vec{X}_1 = \vec{B}_1 \\ \qquad \texttt{where}_\nu\ \vec{X}_2 = \vec{B}_2\end{array}$$

Now, the only fixed-points that can remain in B_1' are (a) ν-fixed-points which contain variables from $\vec{X}_1$, and (b) μ-fixed-points contained in ν-fixed-ponts. The μ-assertions from (b) cannot contain any free variables from any enclosing ν-assertions, because this would immediately yield an assertion of alternation depth at least three. (A generic instance is $\mu X.A[X, \nu Y.B[X, Y, \mu Z.C [Y, Z]]]$ assuming that all free variables are accounted for in the square brackets, which is easily seen to have alternation depth at least three.) Hence, such assertions can be pulled out to give an equivalent assertion

$$\begin{array}{l}(A_2 \texttt{ where}_\mu \vec{X}_1 \vec{X''}_1 = \vec{B'}_1 \vec{B''}_1 \\ \qquad \texttt{where}_\nu \vec{X}_2 = \vec{B}_2\end{array}$$

Finally, the ν-assertions from (a) can be pulled out to get the assertion

$$\begin{array}{lll}(A_2 \texttt{ where}_\mu\ \vec{X}_1 \vec{X}_1'' & = \vec{B}_1' \vec{B}_1'' & \\ & \quad \texttt{where}_\nu \vec{Y}_1 & = \vec{C}_1 \\ \qquad \texttt{where}_\nu \vec{X}_2 & = \vec{B}_2 & \end{array}$$

in which neither A_2 nor $\vec{B}_1', \vec{B}_1''$ or $\vec{C}_1$ contain any fixed-points. Repeating this procedure, we end up with an assertion of the claimed form.

The size of the resulting assertion is no bigger than the original; we are only moving subassertions around and do not copy anything, hence using Mu-Nu two-components algorithms and Nu-Mu two-components algorithms for

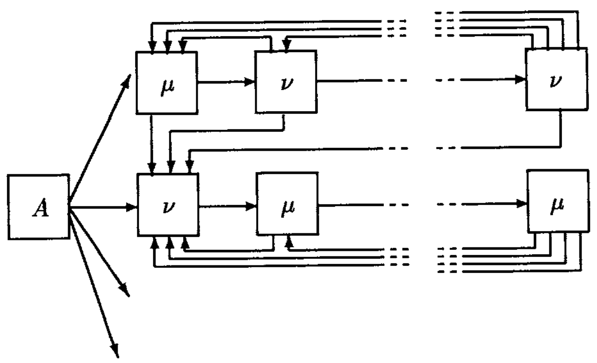


Figure 5.10: A normalized assertion of even alternation depth. Arrows indicate dependencies.

the two different kinds of nested `where`-clauses on the assertion after division with a transition system $|T|$, gives us a local model checker for alternation depth two that runs in time $O(|A|^2|S||T|\ \log(|A||S|))$, i.e. only a logarithmic factor worse than the global algorithm from section 5.4.

By similar transformations, any closed assertion of alternation depth k can be transformed to a normalized form like

$$\begin{array}{llll}
(\ldots((A\ \texttt{where}_\mu\ \vec{X}^1 = \vec{B}^1 & & & \\
 & \texttt{where}_\nu\ \vec{X}^2 = \vec{B}^2) & & \\
 & & \texttt{where}_\mu \ldots & \\
 & & & \texttt{where}_\nu\ \vec{X}^k = \vec{B}^k) \\
\quad\texttt{where}_\nu\ \vec{Y}^1 = \vec{C}^1 & & & \\
 & \texttt{where}_\mu\ \vec{Y}^2 = \vec{C}^2) & & \\
 & & \texttt{where}_\nu \ldots & \\
 & & & \texttt{where}_\mu\ \vec{Y}^k = \vec{C}^k) \\
\quad\vdots & & &
\end{array}$$

as sketched in figure 5.10. The generalization of the algorithm from the previous section could be done along the following lines for one μ-ν-branch:

Partition the input nodes V_i of each Mu- or Nu-Component i into two: let V_i^L be the set of input nodes connected to components to the left of i, and let V_i^R be the set of input nodes connected to components to the right of i. Hence $V_1^L = \emptyset$ and $V_k^R = \emptyset$. (For the two-components case we had $V_1^R = V_y, V_1^L = V_x$.) Now, each component must have attached two sets of input nodes $R_i \subseteq V_i^R$ and $S_i \subseteq V_i^L$. The algorithm will start searching for the value of an output node of component 1, and when an input is needed start searching in the corresponding component. For the i'th component, whenever an input x in V_i^R is needed, the value of which is undefined, computation is suspended in i and computation of the value of x is initiated. Whenever an input y in V_i^L is needed, and the value of y is undefined, it is assumed to have the value 0 if y is an output node from a Mu-Component, and assumed to have the value 1 if it is a Nu-Component, and computation can proceed. The two sets R_i and S_i are used to collect these input nodes, and if output nodes of a component disagree with the connected input nodes (because of the 'assume 0/assume 1'-strategy) components must be re-initialized, and values recomputed. Whenever a component is re-initialized all components to the right must be re-initialized too. As this brief explanation shows there are a lot of details to take care of and we refrain from giving the algorithm and the associated proof of correctness.

Nevertheless, we conjecture that this sketch can be formalized to a correct algorithm, and that properly implemented, for the modal μ-calculus it will run in worst-case time $O(|A|^{ad}|S|^{ad-1}|T| \log(|A||S|))$, thus have worst-case behaviour which is only a logarithmic factor worse than the global algorithm from section 5.4 – and of course on average it could be much better.

5.8 Implementational Aspects

When giving the complexity bounds we assumed the full power of a RAM model. The main feature of RAM models, besides the ability to perform simple operations like integer arithmetic and copying of integers, is that there is constant-time access to computed addresses like in array indexing (see Aho, Hopcroft and Ullman [3] for a discussion of RAM models). However, if one looks at the algorithms and the complexity arguments we only use this feature for the global algorithm for alternating fixed-points: In 'Chasing 1's' the strength can be stored together with the predecessor list of each variable, and the traversal of a predecessor list is an easy 'chasing of pointers.' Similarly, the local algorithms make no use of the feature. Hence, they could be used on a weaker machine model where there is constant-time access to addresses, and addresses can only be allocated and copied, not computed.

To get the full computational benefits of the local algorithms, it is important that the transition systems are generated in a local fashion. One potential problem here is that the operational semantics is defined as syntactic manipulations of process terms, which is not easy to implement efficiently. For the static processes, however, another approach could be used. The transition systems of the regular subprocesses could be computed first. This is an easy task since a regular process is very close to actually being just another representation of a transition system. Now, each state of the transition system of the static process corresponds to a tuple of states in the regular process (cf. figure 2.6). The set of transitions out of a state of the static process can now be computed lazily by considering the transitions out of each of the local states of the regular processes.

5.9 Model Checking the Extended Calculus

In order to extend the model checkers from $\mu K_{\mathtt{where},Q}$ to the full μK_{ext} we must somehow deal with action quantifiers and action predicates. The translations given in section 2.7 offer at least two subclasses of assertions for which this is easily done: If the only predicates of the assertion are matches against constants or if all predicates are guarded, a simple replacement of the existential quantifiers with finite disjunctions suffice to get an assertion in $\mu K_{\mathtt{where},Q}$ on which the model checkers are immediately applicable. Of course, this increases the size of the assertion. To see how much, we define a notion of depth of action quantifiers.

Definition 5.5 Let the *action quantifier depth* $aqd(A)$ of an assertion A be defined by

$$\begin{array}{rcl}
aqd(\neg A) = aqd(\langle\gamma\rangle A) & = & aqd(A) \\
aqd(A_0 \vee A_1) & = & \max\{aqd(A_0), aqd(A_1)\} \\
adq(\exists\alpha.A) & = & 1 + aqd(A) \\
aqd(Q) = aqd(X) & = & 0 \\
aqd(\ \vec{A}^{\,n}\ ) & = & \max\{aqd(A_i)\}_{1\le i\le n} \\
aqd(\ \vec{A}^{\,n}\ \mathtt{where}_\mu\ \vec{X}^{\,m}\ =\ \vec{B}^{\,m}\ ) & = & \max(\{aqd(A_i)\}_{1\le i\le n} \cup \{aqd(B_j)\}_{1\le j\le m})
\end{array}$$

□

Proposition 5.2 *Let A be an assertion, T a transition system with finite basic label-set L. The assertion A' constructed from A by replacing each existential quantifier by a finite disjunction over $L \cup \{\#\}$ has size $O(|A|(|L|+1)^{aqd(A)})$.*

Performing division on A' results in a boolean fixed-point expression of size $O(|A|(|L|+1)^{aqd(A)}|T|)$. However, sometimes we can do much better. A crucial point is that instead of transforming the assertion A into a quantifier-free A' and then perform the division, we instead perform the division first, in the way we extended it to μK_{ext}, and first then eliminate the action quantifiers.

Example 5.1 As a simple example consider $\tilde{\exists}\alpha\langle\alpha\rangle A$ with $\alpha \notin fv(A)$ which we have abbreviated $\langle . \rangle A$. The division yields:

$$\tilde{\exists}\alpha\langle\alpha\rangle A/s \;=\; \tilde{\exists}\alpha \bigvee_{a,s';s\xrightarrow{a}s'} ((\alpha = a) \wedge A/s')$$

Using some simple facts from predicate calculus we can rewrite this as follows:

$$\begin{aligned}
\tilde{\exists}\alpha. \bigvee_{a,s';s\xrightarrow{a}s'} ((\alpha = a) \wedge A/s') &= \bigvee_{a,s';s\xrightarrow{a}s'} (\tilde{\exists}\alpha.\alpha = a) \wedge (A/s') \\
&\qquad \text{as } \exists \text{ and } \vee \text{ commutes, and } \alpha \notin fv(A) \\
&= \bigvee_{a\neq *,s';s\xrightarrow{a}s'} A/s'
\end{aligned}$$

This means that the total size of $(\exists\alpha\langle\alpha\rangle A/\vec{s})$ is no bigger than $|\{(a,s') \mid s \xrightarrow{a} s'\}| + |A||T| \leq (1+|A|)|T|$ if the expressions A/s' are properly shared by means of a `where`-clause. This is much better than what proposition 5.2 suggests and no worse than for a modality in the standard calculus. □

Here are two more examples which give surprisingly compact divisions:

Example 5.2 Consider an assertion $A \equiv \tilde{\forall}\alpha.[\alpha \times *]\langle * \times \alpha\rangle X$. Let T_p and T_q be the transition systems induced by the processes p and q. Then the total size of $A/p_i \times q_j$ when p' and q' range over R_p and R_q is bounded by $|T_p||T_q|$.

To see that this is true, observe that

$$\begin{aligned}
& (\tilde{\forall}\alpha.[\alpha \times *]\langle * \times \alpha\rangle X)/p_i \times q_j \\
= & \bigwedge_{a \in (bls(L_p \times L_q)\setminus\{*\}) \cup \{\#\}} [a \times *]\langle * \times \alpha\rangle X/p_i \times q_j \\
& \quad \text{by lemma 2.5} \\
= & \bigwedge_{a \in (bls(L_p)\setminus\{*\}) \cup \{\#\}} \bigwedge_{p'; p_i \xrightarrow{a} p'} \langle * \times \alpha\rangle X/p' \times q_j \\
= & \bigwedge_{a \in (bls(L_p)\setminus\{*\})} \bigwedge_{p'; p_i \xrightarrow{a} p'} \bigvee_{q'; q_i \xrightarrow{a} q'} X_{p' \times q'} \\
& \quad \text{assuming that} X_{p' \times q_j} \text{is the proper component of} \sigma(X) \\
= & \bigwedge_{a \neq *; p'; p_i \xrightarrow{a} p'} \bigvee_{q'; q_i \xrightarrow{a} q'} X_{p' \times q'}
\end{aligned}$$

Summing over all q_j this gives the bound $|\{(a, p') \mid a \neq *, p_i \xrightarrow{a} p'\}||T_q|$ for each p_i, which by summing over all p_i gives the bound $|T_p||T_q|$ for the total reduction. □

Example 5.3 Let us now consider a slightly more complicated example; a 'weak' version of the above: $A \equiv \tilde{\forall}\alpha.[\alpha \times *]\langle\langle\alpha\rangle\rangle_r X$. Given the processes p and q inducing the transition systems T_p and T_q with the states R_p and R_q, and basic label-set $L = bls(L_p \times L_q)$, we can rewrite A using lemma 2.5:

$$A = \bigwedge_{a \in L\setminus *} [a \times *]\langle\langle\alpha\rangle\rangle_r X$$

Moreover, if we let

$$B_a = \langle\langle\alpha\rangle\rangle_r X$$

we can write rewrite A as

$$A = \bigwedge_{a \in L\setminus *} [a \times *]B_a.$$

Then, as before we get

$$\mathrm{red}_{p_i \times q_j}(A) = \bigwedge_{a \neq *, p_i \xrightarrow{a} p'} \mathrm{red}_{p' \times q_j}(B_a)$$

Now, B_a is an abbreviation for $\langle\langle a\rangle\rangle_r X = \mu Y.(* \times \tau)Y \ \vee \ \langle * \times a\rangle \mu Z.\langle * \times \tau\rangle Z \ \vee \ X$ which can be written in simple form as,

$$\begin{aligned} B_a = Y_a \ \texttt{where}_\mu \ Y_a &= Y'_a \vee Y''_a \\ Y'_a &= \langle * \times \tau\rangle Y_a \\ Y''_a &= \langle * \times a\rangle Z_a \\ Z_a &= Z'_a \vee X \\ Z'_a &= \langle * \times \tau\rangle Z_a \end{aligned}$$

For each q_j we only need $\text{red}_{p' \times q_j}(B_a)$ for pairs (a, p') such that for at least one $p_i, p_i \xrightarrow{a} p'$, hence at most $|T_p|$ of these. Moreover, for each of these the result of performing $\text{red}_{p' \times q_j}(B_a)$ gives an assertion of size at most $O(|T_q|)$ hence the overall size of the reduced assertion is $O(|T_p||T_q|)$. $\square$

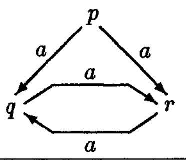


Figure 5.11: A three-state transition system.

Remark 5.3 The sharing across products made possible by the `where`-clause is crucial for keeping the size of assertions small. To see why, consider the three-state transition system of figure 5.9 and the assertion

$$\begin{aligned} A \equiv X \ \texttt{where}_\mu \ X &= \langle a\rangle Y \\ &\quad \texttt{where}_\mu \ Y = X \vee \langle a\rangle Y \end{aligned}$$

(We will not be bothered by the fact that A is equivalent to the simpler $\nu Y.\langle a\rangle Y$, which is irrelevant for the present discussion.) If we perform the division according to figure 5.1 we get the following assertion

$$\begin{array}{llcl} A/p = X_p \ \mathtt{where}_\mu & X_p & = & Y_q \vee Y_r \\ & X_q & = & Y_r \\ & X_r & = & Y_q \\ & & & \mathtt{where}_\nu \quad \begin{array}{lcl} Y_p & = & X_p \vee Y_q \vee Y_r \\ Y_q & = & X_q \vee Y_r \\ Y_r & = & X_r \vee Y_q \end{array} \end{array}$$

The size of A/p is clearly $c|A||T|$ where c is some constant, and T is the transition system pointed by p. If we did not allow sharing across products the ν-part would have to be attached to all three of the right-hand sides of the μ-part, and it is not hard to see that as fixed-points get nested, this gives an exponential blow-up. $\Box$

5.10 Some Applications: Equivalences and Preorders Revisited

We have now presented various model-checking algorithms for finite-state systems. Besides their immediate use as an automatic way of getting assertions verified for finite-state systems, they provide a means of getting algorithms for computing equivalences, preorders and other relations that can be expressed as assertions in the extended calculus – provided these assertions meet criteria which make them transformable into the standard calculus. This gives for instance algorithms for *all* relations that are expressible as guarded assertions in $\mu\mathrm{K}_{ext}$, including very familiar relations like those described in chapter 4. How does this – in principle, infinity of algorithms – compare with more ad hoc constructed algorithms? To answer this question we utilize the two examples from the previous section giving rather precise bounds on the sizes of certain kinds of assertions.

For the ‘strong relations’ like strong bisimulation, ready simulation, ready bisimulation, and prebisimulation example 5.2 shows that they have reduced boolean fixed-points assertions of size $O(tt')$ of alternation depth one, hence the global algorithms will execute in time $O(tt')$. For the ‘weak

Relation	Modal μ-calculus	Algorithm from elsewhere	Note
Strong bisim., $\sim$	$O(tt')$	$O((l+l')(t+t')\log(s+s'))$	1
Weak bisim., $\approx$	$O(ss'tt')$	$O((l+l')(t^2+t'^2)\log(s+s'))$	2
Congruence, $=$	$O(ss'tt')$	-	
Ready sim. / Ready bisim.	$O(tt')$	$O((s+s')(t+t'))$	3
Sim. preorder, $\prec$	$O(ss'tt')$	-	
Prebisim., $\sqsubseteq$	$O(tt')$	$O(tt')$	4
Weak prebisim.	$O(ss'tt')$	-	

$I = |L| l' = |L'| t = |T| t' = |T'| s = |S| s' = |S'|$

Notes:

(1) from Cai and Paige [20] — generalized to non-singleton sets of actions from the original paper Paige and Tarjan [65].

(2) from Estenfeld et. al. [37].

(3) from Bloom and Paige [14].

(4) from Cleaveland and Steffen [26].

Table 5.1: Worst-case running times of some immediately derived algorithms. The local counterparts would have all the running times multiplied with $\log(ss')$. Ready bisimulation is also often called 2/3-*bisimulation*.

relations' like weak bisimulation, congruence, simulation preorder, and weak prebisimulation example 5.3 shows that they have reduced boolean assertions of size $O(tt')$ with at most ss' references from the innermost fixed-point to the outermost, and they have alternation depth two, hence the global algorithms will execute in time $O(ss'tt')$.

The results are summarized in table 5.1.

Not surprisingly the algorithms we get from the μ-calculus cannot in general compete with the carefully constructed algorithms for bisimulation equivalence, which take full advantage of the knowledge of the relation being an equivalence allowing a representation by its equivalence classes. However, for the preorders the difference is minor. For ready simulation the difference is down to the difference between $(t+t')(s+s')$ and tt', i.e. if the transition system is sparse there is no real difference. For the prebisimulation we achieve the same complexity as Cleaveland and Steffen; and for the remaining preorders we have found no non-trivial bounds – only the trivial algorithms computing approximants, which has running time $O(ss'tt')$ for the relations of alternation depth one and $O((ss')^2tt')$ for alternation depth two.

What we also have is *local* algorithms for all these relations including the possible benefits from local approaches, and they have worst-case complexities that are at most a logarithmic factor worse than the global algorithms.

5.11 Bibliographic Notes

The idea of model checking was pioneered by Clarke and Emerson [21] who described a model checker for CTL. The group around Clarke has been mainly interested in verification of hardware designs based on the logics CTL and CTL* and has used a method based on compact representations of sets of states ('BDD's') to handle large state-spaces (see for example Burch et. al. [19]).

Emerson and Lei [35] described the first non-trivial model checker for the modal μ-calculus. They essentially exploited the observation from Bekič's theorem used in the normalization of assertions in section 5.7 to give a model checker running in time $O(|A|^{ad+1}|S|^{ad}|T|)$.[4] The global algorithm from section 5.4 (first published in [6]) improves this bound to $O(|A|^{ad}|S|^{ad-1}|T|)$. (Later the same bound has been achieved by Cleaveland, Dreimüller and Steffen [24], though it is not obvious whether they use the stronger definition of alternation depth used in this thesis.) Taking into account that most properties seem to be expressible in alternation depth two, for which the improvement is from $O(|A|^3|S|^2|T|)$ to $O(|A|^2|S||T|)$, this is a considerable difference. For alternation depth one, the improvement is from $O(|A|^2|S|^|T|)$ to $O(|A||T|)$, a result that has independently been achieved by algorithms of Cleaveland and Steffen [27] and Vergauwen and Lewi [85] which like our algorithm all are building on the idea of Arnold and Crubille [9] of using *simple assertions* as an intermediate form. (They all refer to their algorithms as being linear although they are linear in the *product* of the size of the transition system and the size of the assertion — and for Arnold and Crubille the square of the size of the assertion.)

The idea of local model checkers for the modal μ-calculus is due to

[4]Recall that they use a slightly different definition of alternation depth. However, we claim without proof, that their algorithm is really running in the time determined by our stronger measure. If this should not be the case, the improvement from our global algorithm is even bigger.

Larsen [53]. (He refers to the modal μ-calculus as 'Hennessy-Milner logic with recursion'.) Larsen's paper describes a tableau-like method restricted to the case of one-level fixed-points, i.e. a subset of alternation depth one, and was later extended to the full calculus as a method based on tableaux by Stirling and Walker [80] who named the approach *local model checking.* It has been recast as a rewrite method by Winskel [92] which gives a very simple proof of correctness. The motivation of Stirling and Walker was to give a proof system for showing satisfaction, and its implementation as an algorithm is by Cleaveland [23]. The tableau-method has been used as the basis for an implementation in the Edinburgh-Sussex Concurrency Workbench [25] and Larsen's algorithm is used in the TAV System [55].

All these local methods suffer from severe inefficiencies. They have bad worst-case behaviour and have exponential running times in the size of the transition systems – even the optimizations suggested by Cleaveland [23] does not improve on this fact. However, Larsen has recently published an improvement of his original algorithm, which for one fixed-point has polynomial running time. Nevertheless, it is not as efficient as the algorithm of section 5.6 running in time $O(|A||T|\log(|A||S|))$. Xinxin describes in his thesis [96] a possible extension of Larsen's algorithm to the full calculus. His algorithm is rather involved and the complexity measure he gives, although polynomial in the alternation depth, is considerably worse than the bound of $O(|A|^{ad}|S|^{ad-1}|T|\ \log(|A||S|))$ proposed as likely in section 5.7.3 and substantiated by the two-components algorithm running in worst-case time $O(|A|^2|S||T|\ \log(|A||S|))$.

Cleaveland and Steffen presents in [26] ideas of computing preorders very much like the approach outlined in section 5.10, however, whereas they suggest checking $s \leq s'$ for a preorder $\leq$ by generating a characteristic formula C of $\leq$ with respect to s by *ad hoc* means and verify whether s' satisfies it, we get the same effect in a much simpler way by writing down directly the definition of $\leq$ as a formula in the extended modal μ-calculus thereby allowing model checking algorithms to be applied directly. Moreover, we can get characteristic formulas *for free* through the reduction for product and hence also characteristic formulas for all the preorders investigated in Steffen [75].

The complexity bounds we have given are all based on the technique of amortized complexity analysis as explained in for instance Cormen, Leiserson and Rivest [28].

Chapter 6

Computing Fixed-Points in Finite Cpo's and Lattices

The local model-checking algorithms in chapter 5 are based on ideas of *sharing values* and *tracing changes along dependencies*; ideas that are not inherently connected to the problem of model checking. In this chapter we exploit this observation by considering the problem of finding minimum solutions to general monotonic equation systems; present a local algorithm, prove it correct, and discuss possibly applications and the relation to the model-checking algorithms. Except from the discussion on the relation to the model-checking algorithms, which among other things will provide correctness proofs for these algorithms, this chapter is self-contained and has no direct relevance to the verification of concurrent systems and is as such independent of the rest of the thesis.

6.1 Summary

We present a very simple, yet general algorithm for computing simultaneous, minimum fixed-points of monotonic functions, or turning the viewpoint slightly, an algorithm for computing minimum solutions to a system of monotonic equations. The algorithm is local (demand-driven, lazy, ...), i.e. it will try to determine the value of a single component in the simultaneous fixed-point by investigating only certain necessary parts of the description of the monotonic function, or in terms of the equational presentation, it will determine the value of a single variable by investigating only a part of the equational system.

In the worst-case this involves inspecting the complete system, and the algorithm will be a logarithmic factor worse than a global algorithm (computing the values of all variables simultaneously). But despite its simplicity the local algorithm has some advantages which promises much better performance on typical cases. The algorithm should be seen as a schema that for any particular application needs to be refined to achieve better efficiency, but the general mechanism remains the same. As such it seems to achieve performance comparable to, and for some examples improving upon, carefully designed *ad hoc* algorithms, still maintaining the benefits of being local.

We will illustrate this point by tailoring the general algorithm to concrete examples in such (apparently) diverse areas as type inference, model checking, and strictness analysis. Especially in connection with the last example, strictness analysis, and more generally abstract interpretation, it is illustrated how the local algorithm provides a very minimal approach when determining the fixed-points, reminiscent of, but improving upon, what is known as Pending Analysis [97].

6.2 Introduction

Fixed-points arise everywhere in computer science, when giving semantics of programming languages, in program analysis, in program optimization, in program verification, and many other situations. We will present a general algorithm for computing such fixed-points in complete partial orders (cpo's), and hence lattices, of finite height. (Any poset with bottom of finite height is trivially a cpo, but we stick to the term cpo because we will only apply the finiteness property when it is strictly necessary.) The algorithm will be well-suited to situations where the fixed-points belong to large products of cpo's, and examples of type inference problems, strictness analysis, and model checking problems will be shown to fit into the general framework and yield efficient algorithms.

The simultaneous fixed-points will be described as solutions to sets of monotonic equations and the algorithm works on such descriptions in a local fashion, i.e. the algorithm will compute the fixed-point 'demand-driven' or *locally*, assuming that only some of the components of the fixed-points are really of interest, start from one such component and investigate what is necessary to determine the value. In this respect it differs from most fixed-point finding algorithms which tends to *globally* compute the complete fixed-point. A notable example of such an algorithm is due to Kildall [48], which describes the algorithm as solving a problem of dataflow analysis. We show how this, indeed very simple, global algorithm in a version suitable for the present framework, is related to the local, and show that the possible benefits of the local algorithm compared to the global has a cost of a logarithmic factor in the worst-case, but the typical case would out-perform the global algorithm.

The component of interest could be for instance a variable denoting 'error' in the case of type inference, and hence the algorithm would only search locally for an error in the program under consideration, without necessarily assigning types to all program fragments. In the case of strictness analysis, this component will typically be a function applied to one particular argument, which will then be computed without necessarily computing the behaviour of the function on *all* arguments as would the global algorithm.

To be more precise, we will consider systems of equations on the following form:

$$\begin{array}{rcl} x_1 & = & f_1(\chi_1) \\ & \vdots & \\ x_n & = & f_n(\chi_n) \end{array} \tag{6.1}$$

where χ_i is a tuple of variables $(\chi_{i1}, \dots, \chi_{ia_i})$. Associated to each variable x_i is a set of values D_{x_i} which we also refer to as D_i and we require this set to be a complete partial order (cpo) with a bottom element denoted by $\bot_{D_i}$ and of finite height. We use $\mathit{Pred}(x_i) = \{x_i \mid \exists l.\chi_{jl} = x_i\}$ for the set of *predecessors* of the variable x_i. Moreover, for $1 \leq i \leq n$ the function f_i must be monotonic with type

$$f_i : D_{i1} \times \cdots \times D_{ia_i} \to D_i$$

where we have written D_{il} for $D_{\chi il}$. The combined effect of the right-hand sides is a function

$$f_1 \times \cdots \times f_n : (D_{11} \times \cdots D_{1_{a_1}}) \times \cdots \times (D_{n1} \times \cdots \times D_{na_n}) \to D_1 \times \cdots \times D_n$$

which by using the obvious injection

$$G : D_1 \times \cdots \times D_n \rightarrow (D_{11} \times \cdots D_{1_{a_1}}) \times \cdots \times (D_{n1} \times \cdots \times D_{na_n})$$

with $G(m)_{il} = m(\chi_{il})$ gives rise to the function

$$f : D_1 \times \cdots D_n \rightarrow D_1 \times \cdots \times D_n$$

defined by $f = (f_1 \times \cdots \times f_n) \circ G$. The function f is the function one usually thinks of as being induced by the equation system, but we will need the more refined view offered by the product of the individual functions in order to properly describe and reason about the fixed-point algorithm.

As it is well-known a monotonic equation system (6.2) has a minimum solution, the minimum fixed-point of f, given by

$$\mu f = \sqcup_{i \in \omega} f^i(\vec{\perp})$$

with the definition

$$\begin{array}{rcl} f^0(x) & = & x \\ f^{i+1} & = & f(f^i(x)) \end{array}$$

yielding an increasing chain $\vec{\perp} \sqsubseteq f(\vec{\perp}) \sqsubseteq f^2(\vec{\perp}) \sqsubseteq \ldots$ where $\sqsubseteq$ is the ordering on the cpo $D_1 \times \ldots \times D_n$, and $\sqcup$ is the least upper bound of increasing chains.

6.3 Algorithm

Tentatively the algorithm will proceed as follows. We associate with each variable x_j a value $m(x_j)$, which denotes the current value of x_j. Initially the value of all variables will be 'unknown.' We assign the variable of interest, x_i say, marking $\perp$ and puts x_i in a set of *active* variables, for which the right-hand sides will be inspected to verify that the current marking is identical to whatever the right-hand side evaluates to. In evaluating right-hand sides we will always try to inspect as few of the sons as needed, utilizing that the function might be determined by the current marking of only some of the sons. When evaluating a right-hand side it might of course turn out that we do indeed need the value of some sons, which will be assumed to have the value $\perp$ and put on the list of active nodes to be examined. In doing so, we keep track of dependencies between variables, and whenever it turns out that a variable changes its marking (actually, it can only increase) all variables that might depend on this particular variable is put in the active set to be re-examined. At some point the set of active nodes will become empty, and we have actually found (part of) the fixed-point.

This approach has two benefits:

1. Only variables *reachable* from the root variable x_i through the 'sons-of' relation will ever be investigated, a kind of *syntactic dependency analysis.*

2. Moreover, only variables that turns out to be *actually needed* in determining a right-hand side will ever be investigated, a kind of *semantic dependency analysis.*

Of course, in the worst case the set of variables visited might be precisely the set of variables 'syntactically' reachable from the root variable, but potentially much fewer might be needed. Another important property of the sketched algorithm is that all this happens on-the-fly: The complete system does not have to be computed a *priori*, but the right-hand sides can be supplied on demand.

6.3.1 'Unknown' Values

In order to formally present the algorithm we will introduce notation for 'unknown' values. Technically, we will define a special kind of lifting $(\_)_?$ of cpo's, and characterize a class of functions which behaves properly with respect to unknown arguments.[1]

The ?*-lifting* $D_?$ of a poset D is the poset

$$D_? = \{?\} \cup \{\lfloor d \rfloor d \in D\}$$

with ordering $\leq_{D_?}$, defined by

$$x \leq_{D_?} y \Leftrightarrow_{\text{def}} x = ? \text{ or } \exists a, b \in D.\ x = \lfloor a \rfloor \ \&\ y = \lfloor b \rfloor \ \&\ a \leq_D b$$

Hence ? is a 'bottom element' of $D_?$. (The function $\lfloor\ \rfloor$ is any injective function without ? in its image.)

Definition 6.1 For $1 \leq i \leq k$ let D_i and D be posets and suppose that $f : (D_1)_? \times \ldots \times (D_k)_? \to D_?$ is a function. Then f is

[1]This is definition 5.3 in chapter 5.

- ?-*strict*if

$$f(?, \dots, ?) = ?,$$

- ?-*reflecting* if

$$f(x_1, \dots, x_k) = ? \Rightarrow \exists i.\ x_i = ?,$$

- ?-*monotonic* if

$$\vec{x} \leq \vec{y} \Rightarrow f(\vec{x}) \leq f(\vec{y}) \text{ or } f(\vec{y}) = ?,$$

  and finally,

- ?-*faithful* if for all $y \in D$,

$$\begin{gathered} f(x_1, \dots, x_{i-1}, ?, x_{i+1}, \dots, x_k) = \lfloor y \rfloor \Rightarrow \\ \forall x.\ f(x_1, \dots, x_{i-1}, \lfloor x \rfloor, x_{i+1}, \dots, x_k) = \lfloor y \rfloor. \end{gathered}$$

A function fulfilling all four is said to be ?-*nice*. □

These four notions are intended to capture some intuitive understanding of unknown values: f must depend on its arguments, f can only yield an 'unknown' result if one of its arguments are unknown, f is monotonic in the usual sense, except that sometimes, increasing the arguments can cause f to yield an unknown value, and finally, if f yields a known value with some argument unknown, it must be independent of that particular argument (with the other arguments fixed).

We will say that a ?-nice function $f' : (D_1)_? \times \cdots \times (D_k)_? \to D_?$ is a ?-*nice extension* of $f : D_1 \times \cdots \times D_k \to D$ if

$$\forall x_i \in D_i.\ f'(\lfloor x_1 \rfloor, \cdots, \lfloor x_k \rfloor) \ = \ \lfloor f(x_1, \dots, x_k) \rfloor \tag{6.2}$$

i.e. on the lifted elements f' agrees with f. Notice, that for a ?-nice extension f' of f we have

$$\mu x.\ f'(x \sqcup \Omega) \ = \ \lfloor \mu x. f(x) \rfloor \tag{6.3}$$

where $\Omega = (\lfloor \bot_{D_1} \rfloor, \dots, \lfloor \bot_{D_n} \rfloor) \in (D_1)_? \times \ldots \times (D_n)_?$, which follows directly by induction on the approximants, utilizing (6.2).

In a trivial manner, all monotonic functions $D_1 \times \ldots \times D_k \to D$ can be extended to ?-nice functions by mapping any vector of arguments which includes a ? to ?. However, this will result in an algorithm which does not fully exploit the possibility of minimizing the search because it makes the semantic dependency analysis void; all functions will require the presence of all arguments before giving a value.

As an example of how to choose better ?-nice functions, we consider the case of two-point domains.

Example 6.1 Consider the situation where all D_i's are identical to the two-point cpo $\mathbb{O} = \{0, 1\}$ with $0 < 1$. We can define ?-nice extensions of the usual boolean connectives $\vee$ and $\wedge$ as shown in the two tables.

$\wedge$	?	0	1
?	?	0	?
0	0	0	0
1	?	0	1

$\vee$	?	0	1
?	?	?	1
0	?	0	1
1	1	1	1

These extension are non-trivial, e.g. $0 \wedge ? = 0$. Notice also the ?-monotonic behaviour showing that $\wedge$ is not monotonic in the usual sense: For $(0, ?) < (1, ?)$ we get

$$\wedge(0, ?) = 0 \wedge ? = 0 > ? = 1 \wedge ? = \wedge(1, ?).$$

Hence although at one stage the value of the conjunction can be determined by looking at only the first argument, if this argument increases its value to 1, the second argument must be inspected in order to determine the value of the conjunction.

The ?-nice extensions here capture the well-known facts that sometimes the values of a conjunction/disjunction can be determined by considering only one of the arguments. □

6.3.2 The Local Algorithm

To prove the algorithm correct we will use a lemma, which captures a useful property of fixed-points. First, we define a relativized equivalence and relativized partial order on a product of cpo's. Assume $E = \prod_{i \in I} E_i$ is an I-indexed product of cpo's. For any subset $S \subseteq I$, let the relations $=_S$ and $\leq_S$ on E be defined by

$$u \leq_S v \Leftrightarrow_{\text{def}} \forall i \in I.\ u_i \leq_{E_i} v_i$$

and $u =_S v \Leftrightarrow_{\text{def}} u \leq_S v \ \&\ v \leq_S u$.

We quote from chapter 5:

Partial fixed-point lemma, lemma 5.4 *Let I be an indexing set and for each $i \in I$, E_i a cpo, and let $E = \prod_{i \in I} E_i$ be the product cpo of the E_i 's ordered pointwise. Assume $f : E \to E$ is ω-continuous and that $u \in E$. Then for all $S \subseteq I$, if*

$$\begin{array}{rl} (i) & \forall u'.u =_S u' \Rightarrow u =_S f(u'),\ \textit{and} \\ (ii) & u \leq_S \mu f \end{array}$$

then

$$u =_S \mu f.$$

□

The local algorithm is stated in figure 6.1. The active set of variables mentioned previously is denoted by A, and $c(x_j)$ is a vector of values in $\mathbb{O} = \{0, 1\}$ with the l'th coordinate equal to 1 if the l'th son has previously been 'inspected.' We have used the conditional $\to$: $\mathbb{O}_? \times D_? \to D_?$ defined by

$$b \to x = \begin{cases} ? & \text{if } b =? \text{ or } b = O \\ x & \text{if } b = 1 \end{cases}$$

Notation. Let $\mathcal{D}_L = (D_1)_? \times \cdots \times (D_n)_?$ and

$$\mathcal{D}_R = ((D_{11})_? \times \cdots \times (D_{1_{a_1}})_?) \times \cdots \times ((D_{n1})_? \times \cdots \times (D_{n_{a_n}})_?).$$

The inclusion G used for defining f is easily extended to an inclusion $G : \mathcal{D}_L \to \mathcal{D}_R$ by still taking $G(m)_{il} = m(\chi_{il})$. Hence, for a set of ?-nice extensions f'_i of the f_i's we get a ?-nice function $f^\times = (f'_1 \times \cdots \times f'_n) : \mathcal{D}_R \to \mathcal{D}_L$ and a function $f' = f^\times \circ G : \mathcal{D}_L \to \mathcal{D}_L$.

For a $c \in (\mathbb{O}_?)^{a_1} \times \cdots \times (\mathbb{O}_?)^{a_n}$ we define the monotonic function $F_c : \mathcal{D}_L \to \mathcal{D}_R$ as the pointwise application of the conditional $\to$ to c, i.e. for an $m \in \mathcal{D}_L$,

$$F_c(m)_{il} = c_{il} \to G(m)_{il}$$

meaning that for instance the function

$$f^{\times} \circ F_c$$

is almost like f' except that certain values of the argument are 'filtered out' and replaced by unknown values.

```
Input: Monotonic equational system $x_1 = f_1(\chi_1), \ldots, x_n = f_n(\chi_n)$ ; ?-nice extensions $f_i'$ of the $f_i$'s; and a variable $x_i$.
Output: An element $m \in \mathcal{D}_L$ such that $x_i \in \mathrm{dom}(m)$ and $m =_{\mathrm{dom}(m)} \lfloor \mu f \rfloor$.

1:  for all $x_j$ do $m(x_j) := ?$  $d(x_j) := ?$  $c(x_j) := ?$ od
2:  $A := \{x_i\}$  $m(x_i) := \lfloor \perp \rfloor$  $d(x_i) := \emptyset$  $c(x_i) := \vec{0}$
3:  while $A \neq \emptyset$ do
4:      pick an $x_j \in A$  $A := A \setminus \{x_j\}$
5:      $r := f_j'(c(x_j)_1 \to m(\chi_{j1}), \ldots, c(x_j)_{a_j} \to m(\chi_{ja_j}))$
6:      if $r =?$ then
7:          pick an $l$ s.t. $c(x_j)_l = 0$
8:          $c(x_j)_l := 1$
9:          if $m(\chi_{jl}) =?$ then
10:             $m(\chi_{jl}) := \lfloor \perp \rfloor$  $d(\chi_{jl}) := \{x_j\}$  $c(\chi_{jl}) := \vec{0}$
11:             $A := \{\chi_{jl}, x_j\} \cup A$
            else
12:             $d(\chi_{jl}) := \{x_j\} \cup d(\chi_{jl})$
13:             $A := \{x_j\} \cup A$
            fi
14:     else if $r > m(x_j)$ then
15:         $m(x_j) := r$  $A := d(x_j) \cup A$
        fi
16: od
```

Figure 6.1: The general local algorithm.

We overload notation and use Ω for both the value $(\lfloor \perp_{D_1} \rfloor, \ldots, \lfloor \perp_{D_n} \rfloor) \in \mathcal{D}_L$ and the corresponding value $G(\Omega) \in \mathcal{D}_R$.

In the presentation of the algorithm we have chosen to present elements m of $\mathcal{D}_L$ as functions that take a variable x_j to a value in $(D_j)_?$ in order to emphasize the partiality of these elements; they do not need to exist in their entirety. Only defined components need to be stored in memory — an issue we will return to when discussing implementations details. We denote by $\mathrm{dom}(m)$ the set of variables (or indices) on which m is defined. □

Theorem 6.1 (Correctness) *The algorithm of figure 6.1 correctly computes part of the fixed-point μf, i.e. it terminates with an element $m : \mathcal{D}_L$ such that $x_i \in \mathrm{dom}(m)$ and $m =_{\mathrm{dom}(m)} \lfloor \mu f \rfloor$.*

Proof: The correctness proof is straightforward, exploiting the partial fixed-point lemma (lemma 5.4) and using Hoare Logic with an invariant I for the while-loop which is the conjunction of the following assertions:

$$\begin{array}{rl}
(i) & m \leq \mu u.f^{\times}(F_c(u) \sqcup \Omega) \\
(ii) & \forall j.\ m(x_j) > ? \Rightarrow d(x_j) = \{x_i \mid \exists l.\chi_{il} = x_j \ \&\ c(x_i)_l = 1\} \\
(iii) & \forall j.\ m(x_j) > ? \Rightarrow (c(x_j)_l = 1 \Rightarrow m(\chi_{jl}) > ?) \\
(iv) & \forall j.\ m(x_j) > ? \ \&\ (m(x_j) < f'_{x_j} \circ F_c(m) \text{ or } f'_{x_j} \circ \\
 & F_c(m) = ?) \Rightarrow x_j \in A \\
(v) & m(x_i) > ? \\
(vi) & m \leq f^{\times}(F_c(m)) \sqcup \Omega
\end{array}$$

First, we argue that $(I \ \&\ A = \emptyset)$ is enough to prove the result. From (iv) with $A = \emptyset$ and (vi) it follows that,

$$m(x_j) > ? \text{ implies } m(x_j) = f'_{x_j}(F_c(m)). \tag{6.4}$$

Now, let $u \in \mathcal{D}_L$ satisfy $u =_{\mathrm{dom}(m)} m$, i.e. $u \geq m$. Then for all $x_i \in \mathrm{dom}(m)$,

$$\begin{array}{rl}
? < & m(x_j) \\
= & f'_j(F_c(m)) \\
 & \qquad\qquad \text{by (6.4)} \\
= & f'_j(F_c(u)) \\
 & \qquad\qquad \text{as } f'_j \text{ is ?-faithful, } m \leq u \text{ and } F_c \text{ monotonic.}
\end{array}$$

Hence, since $f^{\times} = f'_1 \times \ldots \times f'_n$ then

$$\forall u \in \mathcal{D}_L.\ u =_{\mathrm{dom}(m)} m \Rightarrow m =_{\mathrm{dom}(m)} f^{\times}(F_c(u))$$

and therefore as $f^{\times}$ is ?-faithful and $F_c(u) \leq u$, we have

$$\forall u \in \mathcal{D}_L.\ u =_{\mathrm{dom}(m)} m \Rightarrow m =_{\mathrm{dom}(m)} f^{\times}(u \sqcup \Omega)$$

From (i) we have

$$m \leq \mu u.f^{\times}(F_c(u) \sqcup \Omega) \leq \mu u.f'(u \sqcup \Omega).$$

Combining these two facts, it follows from the partial fixed-point lemma, that

$$m =_{\mathrm{dom}(m)} \mu u.f'(u \sqcup \Omega)$$

and $\mu u.f'(u \sqcup \Omega) = \lfloor \mu f \rfloor$ as f' is a ?-nice extension of f, proving the theorem.

Secondly, to prove that I is indeed an invariant, we consider each conjunct in turn.

(i) : Holds at line 3 as

$$m \leq \Omega \leq \mu u.f^{\times}(F_c(u) \sqcup \Omega). \tag{6.5}$$

and $f^{\times}$ is ?-nice. Now, assume I holds at line 4. The variables m and c of (i) are changed in lines 8, 10, and 15. At line 8 the fixed-point $\mu u.f^{\times}(F_c(u) \sqcup \Omega)$ is (potentially) increased since $F_c(u)$ is increased, hence (i) also holds after line 8. The change in line 10 does not affect (i) due to (6.5). Finally, in line 15, m gets a new value $\dot{m}$. It only differs from m at x_j where it has value

$$\begin{aligned} r &= f'_{x_j}(c(x_j)_1 \to m(\chi_{j1}), \ldots, c(x_j)_{a_j} \to m(\chi_{j a_j})) \\ &\leq f'_{x_j}(F_c(\mu u.f^{\times}(F_c(u) \sqcup \Omega))) \\ &\qquad\qquad \text{by the assumption that } (i) \text{ holds at line 4} \\ &= f'_{x_j}(F_c(\mu u.f^{\times}(F_c(u) \sqcup \Omega)) \sqcup \Omega) \\ &\qquad\qquad \text{by ?-faithfulness of } f'_{x_j} \text{ and } r > ? \\ &= (\mu u.f^{\times}(F_c(u) \sqcup \Omega))(x_j) \end{aligned}$$

Hence (i) holds after line 15.

(ii) : Straightforward by looking at the changes taking place in lines 10 and 12.

(iii) : Also straightforward from the changes in line 8 and 10.

(iv) :The variables of (iv) are changed at lines 4, 10, 11, 13, and 14. From line 4 it still holds with A replaced by $A \cup \{x_j\}$. At line 10 m is changed at χ_{jl} to the lifted value $\lfloor \perp \rfloor$ but as χ_{jl} is added to A together with $\{x_j\}$ in line 11, (iv) still holds. At line 13 x_j is simply re-entered into A and (iv) still holds. At line 14, call the new value of m for $\dot{m}$ It only differs from m at x_j where $\dot{m}\ (x_j)$ is $f'_{x_j} \circ (F_c)(m) >?$ which justifies removing x_j from the left-hand side of the implication. However, the increase of the value at x_j might affect the value of $f^{\times}(F_c(\dot{m}))$, actually, due to (ii) this can only happen at the points given by $d(x_j)$, and as we add these elements to A, (iv) holds again.

(v) : Trivial, by observing that entries in m are only ever increased.

(vi) : It trivially holds at line 3. Now assume that it holds at line 4. Then the only variable in (vi), m is changed at lines 10 and 15. After line 10 (vi) is trivially satisfied again. At line 15, m is changed. Call the new value of m for $\dot{m}$. The only difference is at x_j where $\dot{m}\ (x_j)$ gets the value

$$f'_j(F_c(m)) >?$$

which is definitely less than $f'_j(m)$ which is less than $f'_j(\dot{m})$ as the value at x_j is *increased* from m to $\dot{m}$. Hence (vi) holds again.

□

The correctness proof is independent of whatever particular implementation is used for the data-structures of the algorithm. These choices will of course have a great impact on the complexity of the algorithm. To illustrate this, let us consider what a general implementation could look like. The set of active nodes A could be implemented as a stack with constant insertion and extraction times, and the algorithm will behave very much as a depth-first traversal (Tarjan [81]); m, d, and b are all partial maps, with ? meaning 'out side domain of definition', that could be implemented by some balanced search tree with search time bounded by log n, n being the number of variables. Each entry in m is a simple value; in d it is a dynamically growing list, and each entry in b could be implemented as a simple counter bc with the understanding that the $c(x_j)_l$ of the algorithm equals 1 if the counter $bc(x_j)$ is bigger than l, i.e. $b(x_j)_l = 1$ iff $bc(x_j) > l$.

To sketch the complexity analysis, let $ht(D)$ be the *height of* D i.e. the length of the longest strictly increasing chain in D minus one, and let $c(f)$ be the maximal cost of evaluating f on any of its arguments. Now, observe that every variable will appear in A at most $\sum_{u\in s(v)}(ht(D_u)+1)$ times, as a variable is only re-entered into A if one of its sons gets an increased value, which for each son only can happen $ht((D_u)_?) = ht(D_u)+1$ times. Hence, the worst-case complexity is, by this informal amortized cost argument:

$$O(\sum_{v\in V}(c(f_v)\sum_{u\in S(v)}(ht(D_u)+1))\log n)$$

Using more uniform bounds on the functions and domains we arrive at:

Theorem 6.2 (Complexity) *If the cost of computing each of the functions of f is bounded by c, the arity bounded by a, and the height of the cpo's bounded by h, then the worst-case complexity is*

$$O(ncah \log n).$$

In practice this bound is *very* pessimistic, it will only be reached in very pathological circumstances: The fixed-point will be identical to the 'highest' element in the cpo and as the algorithm proceeds all the variables change their values in the smallest possible steps. Moreover, all the variables must be reachable from the root variable in the syntactical and the semantical sense.

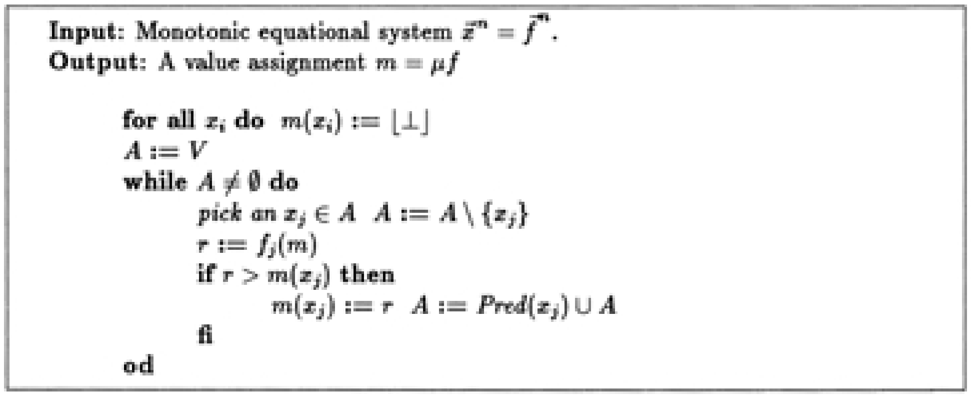
It is, however, very dificult to express anything about the behaviour of the algorithm on typical cases and even to define what a typical case is. However, the examples to be shown later will give some indication of when it is successful.

Before proceeding with the discussion on the local algorithm, let us briefly compare the local algorithm with the *global* algorithm of Kildall [48], shown in figure 6.2, suitable reformulated to fit our framework. In principle it is constructed from the local algorithm by initializing the marking of all variables to $\lfloor\perp\rfloor$, inserting all variables in the active set of variables, and removing the semantic dependency analysis by always taking $d(x_j) = Pred(x_j)$. Of course the branch corresponding to f_x yielding an unknown result is removed. By an argument analogous to the local case, the worst-case complexity is $O(ncah)$, hence the worst-case behaviour of the local algorithm is a logarithmic factor worse, due to the searches in connection with the partial maps. This means that from a strict complexity argument our algorithm is worse, but as it has already been pointed out the local algorithm offers some benefits which the global lacks.

We now turn attention to three examples showing how the local algorithm can be applied. Not all details are included, the general lines are sketched, and emphasis is put on the points of general interest.

6.4 Example: Strictness Analysis

Our first example will be on *strictness analysis* as introduced by Mycroft [60] in a version due to Wadler [86]. However, most of the remarks and constructions apply equally well to abstract interpretation in general.



Input: Monotonic equational system $\vec{x}^n = \vec{f}^n$.
Output: A value assignment $m = \mu f$

```
    for all $x_i$ do  $m(x_i) := \lfloor\bot\rfloor$
    $A := V$
    while $A \neq \emptyset$ do
        pick an $x_j \in A$  $A := A \setminus \{x_j\}$
        $r := f_j(m)$
        if $r > m(x_j)$ then
            $m(x_j) := r$  $A := Pred(x_j) \cup A$
        fi
    od
```

Figure 6.2: Kildall's global algorithm.

We assume that we have given an *abstract program* as set of mutually recursive function declarations:

$$\begin{array}{rcl} f_1(x_{11}, x_{12}, \dots, x_{1_{a1}}) & = & e_1 \\ & \vdots & \\ f_n(x_{n1}, x_{n2}, \dots, x_{n_{an}}) & = & e_n \end{array}$$

where the free variables of the body e_j is included in $\{x_{j1}, \dots, x_{ja_j}, f_1, \dots, f_n\}$. (We will not bother to define any particular syntax for expressions.) Each function has a type

$$f_j : D_{j1} \times \ldots \times D_{ja_j} \to D_j$$

where the D's are cpo's of finite height with bottom (for strictness analysis this will typically be finite lattices) and we assume that all the bodies are indeed well-defined with respect to this typing.

To rephrase this in terms of a monotonic equationa system (6.2) we introduce a variable $v_{f_j;\vec{x}}$ for each pair of function and argument $\vec{x}$ in the ?-lifted product

$$(D_{j1})_? \times \ldots (D_{ja_j})_?.$$

The equation for $v_{f_j;\vec{x}}$ will be a bit special; we will think of the right-hand side $g_{f_j;\vec{x}}$ as a function on *all* variables of the system, i.e. one for each pair of function and possible argument. Although finite, this can be quite a lot of variables!

Now, to evaluate $g_{f_j;\vec{x}}$ we simply proceed, by for instance executing an interpreter for lambda-expressions (if that is the language we have used for the expressions) and when at some point we need the value of a function at a specific argument which is 'unknown', we suspend the evaluation, return the value ?, and proceed with the algorithm, which picks a son (this should of course be the one that made us halt in the first place), assumes it has the value $\lfloor\perp\rfloor$ and proceeds.

To tabulate a function for a range of values U we simply execute the algorithm for each element of U, reusing, of course, earlier stored results.

For this to be valid, we must ensure that the right-hand sides are all monotonic. This could be done by restricting the syntax, but care has to be taken if expressions like $f(f(\ldots),\ldots)$ are to be allowed (cf. the problems with Pending Analysis reported in [32]). We consider this problem to be outside the scope of the current discussion. In the case of higher-order functions, e.g.

$$f : (D \to E) \to E$$

another difficulty arises implicitly: When f is applied to an argument h, we must search for the node $v_{f;h}$, which involves comparing functions. Assuming that D and E are simple domains this is not too bad, the function space will be reasonably small and the task not impossible. Finally, very few functional programs (except perhaps when using continuations) seem to apply functionals on many different functions, so in practice few comparisons will be needed. Moreover, any of the techniques for compactly representing functions could be used to speed up this part.

Considering the second-order case (like with f above) this approach of computing strictness has two major benefits compared to iterative algorithms:

1. Only first order functions will ever have to be compared, no second order functions must be compared to determine stability of the iteration, and in general comparison of n'th order functions for $n + 1$'st order analysis.

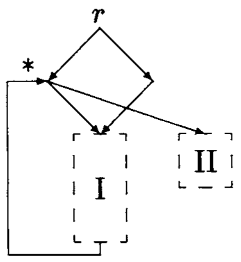
2. Potentially only a very small proportion of the huge number of possible function-argument pairs will be needed.

In this second respect our local algorithm is very similar to Pending Analysis [97], but it can be *exponentially faster*, due to the explicit treatment of dependencies. To see why, we first briefly describe the Pending Analysis:

> As just described the evaluation start with a function applied to one particular argument. If in evaluating such a function application, any previously visited function-argument pair is re-encountered, the value is simply assumed to be bottom. In the case of a lattice of height one this suffices to make sure that the minimum fixed-point will be correctly computed and in the general case the application is re-evaluated until it is stable, every time in a recursive occurrence of a call, using the previously computed value.

To see how this differs from our algorithm, consider the following graph of function calls assumed to occur in the evaluation of a function application. Each arrow indicates a function call.

The pending analysis will traverse this as a tree from the root r, i.e. the root of (I) will be visited twice and so will all nodes in (I). If the structure of (I) is again as above it is not difficult to see that the Pending Analysis will perform exponentially many calls, and this is not merely a problem that can be solved using 'dynamic programming' or 'memoization'. To see why, assume that the Pending Analysis simply stores the computed values (as suggested in [97]) and whenever an application is revisited this stored value



is used. Now, suppose that we first visit the left branch of the diamond, and through the upgoing edge from (I) visit the application (*) a second time, which is then assumed to have the value $\perp$. Then all applications in (I) will be evaluated under the assumption that this is the value of (*), but having visited (I), suppose we finally visit (II) and discover that the value of (*) should have been something bigger than $\perp$. Now, if, as suggested for Pending Analysis, the call in the right branch simply reuses all the values computed for (I) we will end up with a result which is potentially too small! (This is a disaster for strictness analysis — the value will be 'unsafe'.) Surely, the fix is to recompute the values in (I) reflecting the change of (*), which is precisely what our algorithm is doing, moreover it does it in a very minimal fashion by chasing explicit dependencies.

Let us return to a concrete example, the function cat (for 'concatenate') defined in the following program:

```
foldr(f,[],a)     = a
foldr(f,h::t,a)   = f(h,foldr(f,t,a))
append(l,m)       = foldr(cons,l,m)
cat(l)            = foldr(append, 1, nil)
```

with the types

$$\begin{array}{ll}
\texttt{cons} & : \alpha \times \alpha\ \texttt{list} \rightarrow \alpha\ \texttt{list} \\
\texttt{foldr} & : (\alpha \times \gamma \rightarrow \gamma) \times \alpha\ \texttt{list}\ \times \gamma \rightarrow \gamma \\
\texttt{append} & : \alpha\ \texttt{list} \times \alpha\ \texttt{list} \rightarrow \alpha\ \texttt{list} \\
\texttt{cat} & : \alpha\ \texttt{list list} \rightarrow \alpha\ \texttt{list}
\end{array}$$

Let $\mathbf{2} = \mathbb{O} = \{0,1\}, \mathbf{4} = \{0,1,2,3\}$ with $0 < 1 < 2 < 3$ likewise for $\mathbf{6}$. Then

Wadler's analysis [86] suggests the following abstract types, when **cat** is to be instantiated to c `list list` for some ground type c.[2]

$$\begin{array}{ll}
\texttt{cons} & : \mathbf{2} \times \mathbf{4} \to \mathbf{4} \\
\texttt{foldr} & : (\mathbf{4} \times \mathbf{4} \to \mathbf{4}) \times \mathbf{6} \times \mathbf{4} \to \mathbf{4} \\
\texttt{append} & : \mathbf{4} \times \mathbf{4} \to \mathbf{4} \\
\texttt{cat} & : \mathbf{6} \to \mathbf{4}
\end{array}$$

The size of $(\mathbf{4} \times \mathbf{4} \to \mathbf{4}) \times \mathbf{6} \times \mathbf{4}$ is around 10^{11}, so hopefully we do not need to evaluate `foldr` on all its arguments! Actully, in computing `foldr` at one argument the lock algorithm visits at most 24 variables. This case is particularly simple, as the same function is used for the argument of `foldr` in any recursive call, but the general point remains the same. If we look at the functional defining `foldr` this is actually defined on a lattice of $4^{10^{11}}$ elements with height $3 * 10^{11}$, so any attempt of iterating from the bottom inside this huge lattice can be fatal.

6.5 Example: Model Checking

The local algorithm of chapter 5 is a special case of the general local algorithm where all the D_i's are the two-point lattice $\mathbb{O}$ and the functions f_i are either conjunctions and disjunctions. Similarly, the Mu-Component of that chapter is a small refinement of the general local algorithm. These observations makes it possible to prove the correctness of the algorithms without redoing everything from scratch.

Lemma 6.1 (Correctness part of theorem 5.4) *When the local algorithm of figure 5.6 terminates, we have*

$$\begin{array}{ll}
(i) & x_i \in \mathrm{dom}(m) \ \textit{and} \\
(ii) & m =_{\mathrm{dom}(m)} \lfloor \mu f \rfloor
\end{array}$$

Proof: In the local algorithm for the two-point lattices we have for reasons of efficiency realized the effect of $c(x_j)$ by a counter $p(x_j)$. The relationship between the two can be expressed as

[2]Actually `foldr` is needed in two versions corresponding to the two different applications of `foldr`, but the more general of the two has enough information to deduce the strictness information for the other.

$$c(x_j)_l = 1 \Leftrightarrow l \leq p(x_j).$$

Similarly, also for efficiency reason we have used a partial map $h : V \to I\!N_?$ with the property that

$$h(x_i) = |\{l \mid (c(x_i)_l \to m(\chi_{il})) = 1\}|.$$

(This is easily verified to hold invariantly.) However, this does not in any way affect the fact that the local algorithm of figure 5.6 is merely a special case of the general algorithm of figure 6.1, hence (i) and (ii) fo11ow directly from theorem 6.1. □

For the Mu-Component we must slightly alter the invariant, which now should hold after every operation performed on the component under the assumption that the invariant did hold before the call. However, it is a very straightforward change. First, we need a little extra notation.

Notation. For two functions $u : V_x \to \mathbb{O}_?$ and $v : V_y \to \mathbb{O}_?$, with V_x and V_y disjoint, we let $u + v : (V_x \cup V_y) \to \mathbb{O}_?$ be the function defined by

$$u + v(z) = \begin{cases} u(z) & \text{if } z \in V_x \\ v(z) & \text{if } z \in V_y \end{cases}$$

□

Lemma 6.2 (Correctness of Mu-Component, theorem 5.5) *Let K be a Mu-Cormponent for $\vec{x}^n = \vec{b}^n$ with free variables $V_x = \{x_1, \ldots, x_n\}$* and $V_y = \{y_1, \ldots, y_m\}$ *inducing the function $f' : (\mathbb{O}_?)^{V_x \cup V_y} \to (\mathbb{O}_?)^{V_x}$. If, after having performed an init and any sequence of update, $find(x_i)$ and legal $set(y_j, b)$ operations, K is stable, i.e. $A = \emptyset$ then,*

$$\forall v \in (\mathbb{O}_?)^{V_y}.\ v =_R m_y \Rightarrow m_x =_{\mathrm{dom}(m)} \mu u.f'(u + v \sqcup \Omega)$$

Proof: We construct an invariant I from the assertion (i) to (iv) from the proof of theorem 6.1 and add one conjunct to capture the role of the variable R. Hence, define I to be the conjunction of the following assertions:

$$\begin{array}{rl}
(i) & m_x \le \mu u.f^{\times}((F_c(u)+m_y)\sqcup\Omega) \\
(ii) & \forall j.\ m_x(x_j) >? \Rightarrow d(x_j) = \{x_i \mid \exists l.\chi_{il} = x_j \ \&\ c(x_i)_l = 1\} \\
(ii') & \forall j.\ m_y(y_j) >? \Rightarrow d(y_j) = \{x_i \mid \exists l.\chi_{il} = y_j \ \&\ c(x_i)_l = 1\} \\
(iii) & \forall j.\ m_x(x_j) >? \Rightarrow (c(x_j)_l = 1 \Rightarrow (m_x+m_y)(\chi_{jl}) >?) \\
(iv) & \forall j.\ m_x(x_j) >? \ \&\ (m_x(x_j) < f'_{x_j}(F_c(m_x+m_y)) \text{ or} \\
 & \quad f'_{x_j}(F_c(m_x+m_y) =?) \Rightarrow x_j \in A \\
(v) & R = \{y_j \mid \exists i,l.\chi_{il} = y_j \ \&\ c(x_i)_l = 1\} \\
(vi) & m_x \le f^{\times}(F_c(m_x+m_y)\sqcup\Omega)
\end{array}$$

Thus R is the set of y_j's that have been 'needed' by the Mu-Component, i.e. y_j is a son of an x_i which has looked at it.

We have used $m_x + m_y$ for the map $V \to \mathbb{O}_?$ which is constructed by joining together $m_x : V_x \to \mathbb{O}_?$ and $m_y : V_y \to \mathbb{O}_?$; this is the variable m in the data-structure. Similarly, for any pair of maps $u : V_x \to \mathbb{O}_?$ and $v : V_y \to \mathbb{O}_?$, $u+v$ is the map $V \to \mathbb{O}_?$ constructed by joining u and v.

Exactly as in theorem 6.1 we can prove that, after a call to *init*, if we assume that I holds before any call to *find*, *update*, and *set* it also holds after such a call. Hence, when $A = \emptyset$, we have for the same reasons,

$$\forall u \in (\mathbb{O}_?)^{V_x}.\ u =_{\mathrm{dom}(m_x)} m_x \Rightarrow f^{\times}(F_c(u+m_y)) =_{\mathrm{dom}(m_x)} m_x.$$

Consider any $u : V_x \to \mathbb{O}_?$ and any $v : V_y \to \mathbb{O}_?$ with $v =_R m_y$. Then

$$\begin{array}{rcl}
F_c(u+v)(x_j)(l) & = & c(x_j)_l \to (u+v)(\chi_{il}) \\
 & = & \begin{cases} ? & \text{if } c(x_i)_l \in \{?,0\} \\ u(\chi_{il}) & \text{if } c(x_i) = 1, \chi_{il} \in V_x \\ v(\chi_{il}) & \text{if } c(x_i) = 1, \chi_{il} \in V_y \end{cases} \\
 & = & \begin{cases} ? & \text{if } c(x_i)_l \in \{?,0\} \\ u(\chi_{il}) & \text{if } c(x_i) = 1, \chi_{il} \in V_x \\ m_y(\chi_{il}) & \text{if } c(x_i) = 1, \chi_{il} \in V_y \end{cases} \\
 & & \quad \text{as } c(x_i) = 1 \ \&\ \chi_{il} \in V_y \text{ implies } \chi_{il} \in R \\
 & & \quad \text{by } (v) \text{ and } v =_R m_y \\
 & = & c(x_j)_l \to (u+m_y)(\chi_{il}) \\
 & = & F_c(u+m_y)(x_j)(l)
\end{array}$$

Thus for all $u, F_c(u+v) = F_c(u+m_y)$ implying that for all $v =_R m_y$,

$$\forall u \in (\mathbb{O}_?)^{V_x}.\ u =_{\mathrm{dom}(m_x)} m_x \Rightarrow f^\times(F_c(u+v)) =_{\mathrm{dom}(m_x)} m_x.$$

Now, as $f^\times$ is ?-faithful and $F_c(u+v) \leq G(u+v)$, it follows that

$$\forall u \in (\mathbb{O}_?)^{V_x}.\ u =_{\mathrm{dom}(m_x)} m_x \quad \Rightarrow \quad f'(u+v) =_{\mathrm{dom}(m_x)} m_x. \tag{6.6}$$

From (i) we have

$$m_x \leq \mu u.f^\times(F_c(u+m_y) \sqcup \Omega),$$

which using $F_c(u+v) = F_c(u+m_y)$ gives,

$$m_x \leq \mu u.f^\times(F_c(u+v) \sqcup \Omega),$$

and utilizing $F_c(u+v) \leq G(u+v)$ we finally get

$$\begin{aligned} m_x \leq \mu u.f^\times(G(u+v) \sqcup \Omega) &= \mu u.f^\times(G(u+v \sqcup \Omega)) \qquad (6.7) \\ &= \mu u.f'(u+v \sqcup \Omega). \end{aligned}$$

Combining (6.6) and (6.8) we get by the partial fixed-point lemma, that for all $v =_R m_y$,

$$m_x =_{\mathrm{dom}(x_x} \mu u.f'(u+v \sqcup \Omega),$$

6.6 Example: Constraint Systems

Recently, it has become popular to solve various type checking, type inference, and other program analysis related problems, by constructing a set of constraints to be solved. We will show how one of these problems can be solved by the local method with optimal complexity (up to the logarithmic factor).

The particular constraint system we consider is due to Palsberg and Schwartzbach [67], used in performing what they call *safety analysis* — a version of *closure analysis* — but very similar sets of constraints have been used for type inference [66]. For each subterm of the program we will have a variable, which is going to hold information about that particular part of the program (whether this is type information or anything else is irrelevant). In this particular situation, we will think of the information of interest as subsets of what we will call tokens. The set of subsets of tokens is a finite lattice $\mathcal{P}(S)$ ordered by inclusion and S being the set of tokens. Now, the constraint system can be formulated as consisting of a set of conditional and unconditional inequalities

$$c \subseteq x, \quad r \Rightarrow y \subseteq x, \quad x \subseteq c$$

where x, y are token set variables, c a constant token set, and r is a boolean constant defined as $r = e$ where e is an expression built from disjunctions and conjunctions of other boolean constants and the inequalities $c \subseteq x$. Denote by $\mathcal{C}$ the complete set of inequalities and boolean constant definitions. We build a set of monotonic equations with

1. a variable v_x of type $\mathcal{P}(S)$ for each of the token set variables x,
2. a variable b_r of type $\mathbb{O}$ for each of the boolean constants, and
3. a variable $b_{c\subseteq x}/b_{x\subseteq c}$ of type $\mathbb{O}$ for each constraint $c \subseteq x/x \subseteq c$.

We will make use of some auxiliary functions: Let $\rightarrow$ be the conditional of type $\mathbb{O} \times \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ with an obvious definition (using 1 to represent true), $(c\subseteq) : \mathcal{P}(S) \rightarrow \mathbb{O}$ has just as obvious a definition, and finally **viol**$(\subseteq c) : (S) \rightarrow \mathbb{O}$ is defined on $u \in \mathcal{P}(S)$ as the negation of $u \subseteq c$ (to make it monotonic in u).

The equations associated with the variables are now as follows:

$$\begin{array}{rcll}
v_x & = & (\displaystyle\bigcup_{r\Rightarrow y\subseteq x\in\mathcal{C}} b_r \rightarrow y) \cup (\displaystyle\bigcup_{c\subseteq x\in\mathcal{C}} c) & \\
b_r & = & b_e & r = e \in \mathcal{C} \\
 & & \text{where } b_e \text{ is (recursively) defined by} & \\
 & & b_e = \begin{cases} \bigwedge_{i\in I} b_{e_i} & \textit{if } e = \bigwedge_{i\in I} e_i \\ \bigvee_{i\in I} b_{e_i} & \textit{if } e = \bigwedge_{i\in I} e_i \\ b_{r'} & \textit{if } e = r' \end{cases} & \\
b_{c\subseteq x} & = & (c \subseteq) r_x & \\
b_{x\subseteq c} & = & \mathbf{viol}(\subseteq c) r_x &
\end{array}$$

The right-hand sides are all monotonic, and we can easily give ?-nice extensions of the functions involved. For $\bigwedge$ and $\bigvee$ we use the straightforward extension of the binary case from example 6.1 with the efficient implementation discussed in the model checking example. For $\bigcup : \mathcal{P}(S)^k \rightarrow \mathcal{P}(S)$ we take

$$\bigcup(u_1, \ldots, u_k) = \begin{cases} S & \textit{if } \exists i.u_i = S \\ ? & \textit{if } \forall i.u_i \neq S \ \& \ \exists i.u_i = ? \\ u_1 \cup \ldots \cup u_k & \textit{otherwise} \end{cases}$$

The rest are as follows:

if	?	$n \neq ?$
?	?	?
0	$\perp_D$	$\perp_D$
1	?	x

	$(c \subseteq)$
?	?
x	$\begin{cases} 0 & \textit{if } c \not\subseteq x \\ 1 & \textit{if } c \subseteq x \end{cases}$

	viol$(c \subseteq)$
?	?
x	$\begin{cases} 0 & \textit{if } x \subseteq x \\ 1 & \textit{if } x \not\subseteq c \end{cases}$

Now, it is not hard to see, that the set of inequalities $\mathcal{C}$ has a solution, if and only if, the monotonic equation system has a minimum solution in which all the variables $v_{x \subseteq c}$, corresponding to what Palsberg and Schwartzbach calls 'safety constraints', are 0, thus not being violated. The algorithm they suggest has running time $O(n^3)$, n being the size of the program and the number of tokens. They argue that the number of constraints will be $O(n^2)$.

A straightforward implementation of the union operator and the tests for inclusion gives us a local algorithm with the following running time

$$\begin{aligned} &\textstyle\sum_{v \in V}(c(f_v) \sum_{u \in S(v)}(ht(D_u) + 1)) \log n \\ &\quad = \textstyle\sum_{v_x}(a_{v_x} n a_{v_x}(n+1)) \log n + \textit{`cheaper stuff'} \\ &\qquad\quad a_{v_x} \text{ being the arity of the right-hand side of } v_x \\ &\quad = n^2 \log n (\textstyle\sum_{v_x} a_{v_x}^2) \\ &\quad \in O(n^4 \log n) \end{aligned}$$

The way to improve on this shows as a general point how the general algorithm can be used as the backbone of more specific and efficient algorithms, while maintaining the overall structure. Here, the expensive part is the repeated computation of unions of sets, the size of which is bounded by n. But the only thing that happens in the algorithm is 'small' changes in the arguments, so by altering the way changes are propagated we will improve the bound to $O(n^3\ log\ n)$ (and Kildall's algorithm achieves $O(n^3)$, through the same construction).

To each variable we associate a bitvector of length n, the i'th coordinate indicating whether token number i belongs to the set or not. Then, when a variable changes marking and we add the parents to the active set, we will also propagate the *actual change* as a list of tokens, which can then be incorporated in time proportional to the size of the change. In this manner, the *amortized* cost of computing each of the union operations will be bounded by the arity multiplied with $n+1$ (the height of the lattices of the sons). Similarly, the inclusion tests ($c\ \subseteq$) and ($\subseteq\ c$) can be implemented with amortized cost $O(n)$. The total running time is now bounded by

$$(\textit{total cost for } v_x\text{'s} + \textit{total cost for } b_r\text{'s} + \textit{total cost for } b_{c\subseteq x}/b_{x\subseteq c}\text{'s})\ \log n$$

which is

$$O((n^2+n^3+n^3)\ \log n) = O(n^3\ \log n).$$

6.7 Bibliographic Notes and Related Work

The aims of minimizing the number of variables of the equation system investigated when finding a partial minimum solution is shared with the aims of Cousot and Cousot in their "chaotic fixed-point iteration" [29, sec. 4.2.1] and in the refined denotational semantics known as "minimal function graphs" (Jones and Mycroft [47]). However, whereas these papers describe general schemes for computing partial minimum solutions, they are very brief on the subject of when functions "need" the values of other functions applied to specific arguments, and how to incorporate that into an algorithm. Jones and Mycroft leave out this decisions, their description is parameterized by such proper choices, and Cousot and Cousot seems to indicate merely a *syntactic* criterion (corresponding roughly to the part of our algorithm performing syntactic dependency analysis). Contrary to this, we formalize the dependency as the notion of ?-*nice extensions* and show how this together with the explicit presence of a graph representing the semantic dependencies, allowing for efficient sharing and updating of values, makes, even in the worst-case, this local method almost[3] as efficient as the global method — the global method having bad average-case behaviour.

[3]The log-factor.

Similarly, the aims of fast fixed-point finding of functionals illustrated in our example on strictness analysis, is shared by the work of Nielson and Nielson [62, 61]. Their approach is, however, somewhat orthogonal to the algorithms described here. They focus on classifying functions subject to the number of steps needed in computing the fixed-point iteratively and on finding classes for which equality tests of functions can be done without having to consider by brute-force each element in the domains of the functions. (An example is: if the elements of the approximation sequence can guaranteed to be join-preserving, then the functions need only be compared on the join-irreducible elements (often called primes) of the poset constituting the domain of the functions.) Especially, as concerns this last analysis, minimizing the cost of comparing functions, the present algorithm could benefit from their results when applied to higher-order cases.

6.8 Further Work

We have introduced a local fixed-point finder, and shown how it can be used for solving three problems of general interest. The pattern we have used is to reformulate the problems to problems of finding minimum solutions to sets of monotonic equations, and by tailoring the local algorithm achieve an efficient solution to the problem. We expect that this approach can be used on a variety of cases.

An interesting aspect which has not been investigated in this paper, is the potential speed-up coming from decomposing the value domains to smaller domains and thereby adding new variables. A notable example of the success of such an approach is the model checking problem, where the original problem is to compute a fixed-point of a function $f = \lambda X.A$ on a lattice $\mathcal{P}(S)$. This lattice is decomposed to the lattice $\mathbb{O}^{|S|}$ and the corresponding systems has $|S|$ variables with a total size of the right-hand sides of $|A||T|$ (T is the labelled transition system), thereby yielding a significant speed-up (from $|A||T|^2$ to $|A||T|$), which actually corresponds to computing just one approximation to the fixed-point, i.e. one application of f.

The connection to the work from data-flow analysis is intriguing and should be further investigated.

6.9 Acknowledgements

At various points in the work towards this chapter, I have had useful discussions with Chris Hankin, Sebastian Hunt, John Launchbury, Hanne Riis Nielson, Flemming Nielson, Jens Palsberg, Mads Rosendahl, Michael Schwartzbach, Glynn Winskel and others, and last but not least Fritz Henglein who pointed me to Kildall's algorithm.

Chapter 7

Model Checking in Infinite-State Systems

7.1 Introduction

In this chapter we describe a method for performing model checking on infinite state systems in the modal μ-calculus. In contrast to the situation with finite state systems allowing more or less efficient *automatic methods*, we are in general forced to consider only *semi-automatic* or *machine-assisted* methods when considering infinite state systems. This is an obvious fact whenever the class of models and the logic is powerful enough to encode undecidable properties, such as the Halting problem for Turing machines.

The modal μ-calculus is one such powerful logic. It is very straightforward to encode the behaviour of a Turing machine TM as an infinite-state system with states coding the internal state of the Turing machine as well as the contents of the tape; and find an assertion H, s.t.

$$TM(i, w) \models H$$

is valid, if and only if, the Turing machine *TM* when started in the initial state i with tape contents w, halts. So for the general case, any hope of finding an algorithm actuary deciding the model-checking problem is of course domed to failure.

We describe a general *method*, which can assist in proving that subsets of states of infinite labelled transition systems satisfy formulae in the modal μ-calculus. Success of using the method in one particular situation will depend on proper *choices* in certain steps of applying the method, and on the ability to show properties of infinite sets of states by induction. The actual inductive proof takes place as part of the method but depends on a well-founded relation being suppled. The undecidability can now be viewed as a combination of the impossibility of making these choices algorithmically and of the impossibility of algorithmically supplying the 'right' well-founded relation. The method will be sound in the sense that, whenever a model is shown to satisfy an assertion of the logic using the method, this is certainly a valid conclusion, and it will be complete in the sense that, whenever a model satisfies an assertion it is possible to make correct choices, and provide well-founded relations such that in a finite number of steps of the method this fact will be proven.

The method raises some interesting questions. One is a question of 'relative completeness', i.e. in analogy with Hoare Logic, whether proper notations for infinite sets can be found, making the method complete under the assumption that the mathematical reasoning within this notation of infinite sets can be performed. Another issue is whether non-trivial subclasses of the models and perhaps subsets of the logic yields decidable systems.

It is also of great importance to find reasonable notations for subsets of infinite state systems, which, although not necessarily 'relative complete' at least yields convenient frameworks for application of the method. We show how this might be done for *Bounded Processes*, a subset of Milner's CCS [59] where precesses do not have unlimited evolving, but bounded structure. Another example for Petri-Nets can be found in the work of Bradfield [16], [15], which employs a related method due to Bradfield and Stirling [17]. To some extent the present method can be seen as a recast of their method inspired by the work of Winskel [92] for the finite-state case. The relation between their method and the method described here, will be considered in the concluding section.

An interesting point manifest in the method, is the commonly accepted dogma that reasoning about maximum fixed-points is 'easy', like 'partial correctness' in Hoare-Floyd Logic allowing non-termination, and bisimulation equivalence of Process Algebras, whereas reasoning about minimum fixed-points is often more involved, as when showing termination of programs in Hoare-Floyd Logic. The analogy with Hoare-Floyd Logic can actually be made quite precise, see e.g. Bradfield [15, sec. 3.7]. In the method described here, these parallels manifest themselves, as reasoning about minimum fixed-points requires a well-founded relation to be supplied, whereas no such thing is required for the maximum fixed-point.

7.2 Fixed-Points

Winskel [92] has shown that a slightly modified unfolding of a maximum fixed-point can be used as the key step in the development of a model checker for finite-state systems. This property of maximum fixed-points will be introduced as the *second reduction lemma*:

Lemma 7.1 (The second reduction lemma, Kozen [49], Winskel [92]) *Let ψ be a monotonic function on $\mathcal{P}(S)$. For $V \subseteq S$ we have*

$$V \subseteq \nu\psi \ \Leftrightarrow\ V \subseteq \psi(\nu U.V \cup \psi(U)).$$

Winskel uses this lemma in the situation where V is a singleton $\{p\}$. He defines a relation which in a precise sense makes the right-hand side smaller, thus 'simpler' to verify, and because he works with finite-state systems, this relation turns out to be well-founded, ensuring termination of the algorithm. As we consider infinite state systems, termination is no longer guaranteed. Moreover, following Bradfield and Stirling [17] we will try to verify that (possibly infinite) sets of states satisfy an assertion, not only singletons. This seems more appropriate for infinite-state systems; although initially we might only want to know whether one particular state satisfies an assertion, this state can quickly lead to considering whether an infinite number of states satisfy an assertion (an example of this is provided later). So we will be involved in deciding judgements like $V \subseteq U$, where V is a (possibly infinite) set of states and U is a property expressed in our assertional language. We will use lemma 7.1 to give a rule for the maximum fixed-points, but what about the minimum fixed-points? The Duality Principle for Complete Lattices yields an immediate corollary.

Corollary 7.1 *Let ψ be a monotonic function on $\mathcal{P}(S)$. For $V \subseteq S$ we have*

$$V \supseteq \mu\psi \ \Leftrightarrow\ V \supseteq \psi(\mu U.V \cap \psi(U)).$$

This, however, is not very useful. Being interested in determining whether sets of states satisfy a property corresponds to determining whether $V \subseteq \mu\psi$ and *not* $V \supseteq \mu\psi$. So we must find another formulation. Notice, however, that for *singletons* we can derive a useful bi-implication like the one in the reduction lemma:

$$\begin{array}{rcl} p \in \mu U.\psi(U) & \Leftrightarrow & S \setminus \{p\} \not\supseteq \mu U.\psi(U) \\ & & \quad \text{by simple set theory} \\ & \Leftrightarrow & S \setminus \{p\} \not\supseteq \psi(\mu U.(S \setminus \{p\}) \cap \psi(U)) \\ & & \quad \text{by corollary 7.1} \\ & \Leftrightarrow & p \in \psi(\mu U.(\psi(U) \setminus \{p\})). \end{array}$$

(The first and last bi-implication fail for arbitrary sets). Hence, the minimum fixed-point on the right-hand side is now slightly 'smaller' as the state p has been excluded. For finite-state systems this is actually enough to ensure termination as the exclusion of states from a fixed-point cannot go on forever; eventually we will find out that a state p belongs to minimum fixed-point, or we will be involved with deciding whether a state p belongs to a minimum fixed-point from which it has previously been explicitly excluded. (See also the discussion in section 5.2.)

However, for infinite-state systems, excluding singletons are not enough to guarantee termination; we could go on unfolding the fixed-point forever without ever reaching a conclusion. Instead we will use a principle of *well-founded induction* based on the lemma below. Recall, that a relation $\sqsubset$ on the set U is a *well-founded relation* (abbreviated *w.f.r.*) if there does not exist an infinitely decreasing chain $u_0 \sqsupset u_1 \sqsupset \cdots \sqsupset u_n \sqsupset \cdots$. Moreover, we extend a relation $\sqsubset$ on U to a relation on $\mathcal{P}(U)$ by defining

$$V \sqsubset W \Leftrightarrow_{\text{def}} \forall v \in V,\ w \in W.\ v \sqsubset w$$

and we let $(\sqsubset W)$ be the set of elements of U less than *all* elements of W, i.e.

$$(\sqsubset W) =_{\text{def}} \{v \in U \mid \forall w \in W.\ v \sqsubset w\}$$

To state the lemma we need the notion of a covering: A *covering* of U is a collection of sets $\{U_i\}_{i \in I}$ s.t $\bigcup_{i \in I} U_i = U$.

Lemma 7.2 (Well-founded induction on minimum fixed-points)
Let ψ be a monotonic function on $\mathcal{P}(S)$. For a set $U \subseteq S$, the following holds:

If there exists a w.f.r. $\sqsubset$ *on* U *and a covering* $\{U_i\}_{i \in I}$ *of* U *such that*
$\forall i \in I.\ U_i \subseteq \psi(\mu V.(\sqsubset U_i) \cup \psi(V))$
then $U \subseteq \mu\psi$

Proof: Recall, the principle of well-founded induction for a predicate Q on a set U with w.f.r. $\sqsubset$:

$$\text{If } \forall u \in U.\ (\forall u' \sqsubset u. Q(u')) \Rightarrow Q(u) \text{ then } \forall u \in U.\ Q(u).$$

Hence, take any $u \in U$. As $\{U_i\}_{i \in I}$ covers U, there exists a U_i containing u. We now deduce as follows:

$$\begin{array}{rcl}
\forall u' \sqsubset u.\ u' \in \mu\psi & \Rightarrow & \forall u' \sqsubset U_i.\ u' \in \mu\psi \\
& & \quad \text{since } u \in U_i \\
& \Rightarrow & \sqsubset U_i \subseteq \mu\psi \\
& \Rightarrow & \mu V.(\sqsubset U_i) \cup \psi(V) = \mu\psi \\
& & \quad \text{by lemma 5.1} \\
& \Rightarrow & u \in U_i \subseteq \psi(\mu V.(\sqsubset U_i) \cup \psi(V)) = \psi(\mu\psi) = \mu\psi \\
& & \quad \text{by assumption}
\end{array}$$

From the principle of well-founded induction it follows that

$$\forall u \in U.\ u \in \mu\psi$$

proving the lemma. $\Box$

The other direction of the implication holds in a trivially way. Take $I = \{1\}, U_1 = U$, and $\sqsubset$ any w.f.r. for instance the empty relation. Then as $(\sqsubset U) = \emptyset$, the requirement to this trivial covering degenerates to the validity of unfolding of fixed-points. However, also more interesting choices of covering and well-founded relation exist, indeed in showing completeness of the method we will argue that a certain *canonical* covering and relation can be found such that the minimum fixed-point will never be unfolded more than once.

7.3 Logic

We will use a version of the modal μ-calculus, which is essentially the standard calculus (in positive, normal form) extended with constants, sets of actions in the modalities, and annotations on the fixed-points expressing states 'assumed to satisfy' the fixed-point. The syntax is described by the following grammar:

$$A ::= Q \mid A_0 \vee A_1 \mid A_0 \wedge A_1 \mid \langle\kappa\rangle A \mid [\kappa]A \mid X \mid \mu X\{U\}A \mid \nu X\{U\}A$$

In the modalities κ is a (possibly infinite) set of labels, hence the modality $\langle\kappa\rangle A$ is an abbreviation for $\exists a.(a \in \kappa) \wedge \langle a\rangle A$.

The denotation of the modalities are

$$\begin{array}{rcl} [\![\langle\kappa\rangle A]\!]_{T,V}\rho & = & \{s \in S \mid \exists s' \in S\ \exists a \in \kappa.\ s \xrightarrow{a} s' \ \&\ s' \in [\![A]\!]\rho\} \\ [\![[\kappa]A]\!]_{T,V}\rho & = & \{s \in S \mid \forall s' \in S\ \forall a \in \kappa.\ s \xrightarrow{a} s' \Rightarrow s' \in [\![A]\!]\rho\} \end{array}$$

and for the fixed-points, let $\psi : \mathcal{P}(S) \to \mathcal{P}(S)$ be the function $\psi(U) = V \cup [\![A]\!]\rho[U/X]$ and define

$$\begin{array}{rcl} [\![\mu X\{V\}A]\!]_{T,V}\rho & = & \mu\psi \\ [\![\nu X\{V\}A]\!]_{T,V}\rho & = & \nu\psi \end{array}$$

This means that the usual $\mu X.A$ is an abbreviation for $\mu X\{\emptyset\}A$. For closed assertions define $[\![A]\!]_{T,V} = [\![A]\!]_{T,V}\rho$ for any environment ρ. When there is no risk of confusion we will even leave out T and V.

We define the satisfaction predicate $\models$ on sets of states as follows: For a closed assertion A and a set $U \subseteq S$ let

$$\models_{T,V} U : A \Leftrightarrow_{\mathrm{def}} U \subseteq [\![A]\!]_{T,V}.$$

7.4 The Model Checking Method

In this section we will introduce a syntactic counterpart $\vdash$ of the satisfaction relation $\models$ and give a set of rules that allow us to verify that correctness assertions $\vdash U : A$ belongs to $\vdash$. Let $CorrAssn^{cl}$ be the set of closed correctness assertions. We give a binary relation $\longrightarrow \subseteq CorrAssn^{cl} \to \mathcal{P}(CorrAssn^{cl})$ between correctness assertions and sets of correctness aisertions. The intuition is that if $(U : A) \longrightarrow \Gamma$ then to prove that $(U : A)$ is valid, we can prove each of the correctness assertions in the set Γ. However, as the minimum fixed-points can result in infinite sets of correctness assertions — all of the same 'form' — we will describe a 'schematic relation'

$\Longrightarrow \subseteq \mathcal{P}(CorrAssn^{cl}) \to \mathcal{P}(CorrAssn^{cl})$ which will allow sets of correctness assertions to be rewritten.

The rules for $\longrightarrow$ is defined by structural induction on assertions in figure 7.1.

(R1)	$U : Q$	$\longrightarrow$	$\emptyset$	if $U \subseteq V(Q)$
(R2)	$U : A \wedge B$	$\longrightarrow$	$\{(U : A),\ (U : B)\}$	
(R3)	$U : A \vee B$	$\longrightarrow$	$\{(U_0 : A),\ (U_1 : B)\}$	if $U_0 \cup U_1 = U$
(R4)	$U : \langle\kappa\rangle A$	$\longrightarrow$	$\{U' : A\}$	if $U \subseteq (\xrightarrow{\kappa} U')$
(R5)	$U : [\kappa]A$	$\longrightarrow$	$\{(U \xrightarrow{\kappa}) : A\}$	
(R6)	$U : \nu X\{V\}A$	$\longrightarrow$	$\emptyset$	if $U \subseteq V$
(R7)	$U : \nu X\{V\}A$	$\longrightarrow$	$\{U : A[\nu X\{V \cup U\}A/X]\}$	if $U \not\subseteq V$
(R8)	$U : \mu X\{V\}A$	$\longrightarrow$	$\emptyset$	if $U \subseteq V$
(R9)	$U : \mu X\{V\}A$	$\longrightarrow$	$\{U_i : A[\mu X\{V \cup (\sqsubset U_i)\}A/X]\}_{i\in I}$	if $U \not\subseteq V$
				$\bigcup U_i = U$
				$\sqsubset$ w.f.r. on U
(W)	$U : A$	$\longrightarrow$	$\{U' \cup U : A\}$	
($\emptyset$)	$\emptyset : A$	$\longrightarrow$	$\emptyset$	
(I)	$U : A$	$\longrightarrow$	$\{U : A\}$	

Figure 7.1: The rules.

We notice that the rules **(R1)**, **(R2)**, **(R5)**, **(R6)**, **(R7)**, **(R8)**, **($\emptyset$)**, and **(I)** all are deterministic, in the sense that, given an instantiation of the left-hand side there is only one instantiation of the right-hand side, whereas **(R3)**, **(R4)**, **(R9)**, and **(W)** all involve choices, and as there in general will be more than one proper choice, give rise to several possible instantiations of the right-hand sides, thus introducing 'non-determinacy'. For the method to be successful in showing validity of a correctness assertion these choices must all be made correctly. Let us consider the rules in more detail.

(R1), (R2), (R3). All are quite obvious. Only **(R3)** involves a choice.

(R4), (R5). In rule **(R5)** $(U \xrightarrow{\kappa})$ denotes the set of states that can be reached through an action in κ from a state in U, i.e.

$$(U \xrightarrow{\kappa}) = \{s \in S \mid \exists u \in U\ \exists a \in \kappa . u \xrightarrow{a} s\}.$$

$$\boxed{\quad \frac{\forall\gamma\in\Gamma.\ \gamma\longrightarrow\Delta_\gamma}{\Gamma\Longrightarrow\bigcup_{\gamma\in\Gamma}\Delta_\gamma}\quad(\Longrightarrow) \quad}$$

Figure 7.2: The simultaneous rewrite relation.

The importance of this operator is that

$$U \subseteq [\![[\kappa]A]\!] \Leftrightarrow (U \xrightarrow{\kappa}) \subseteq [\![A]\!].$$

It is, however, not possible to define a similar operation for the diamond-modality, which inevitably involves some choices. To see this consider the simple three-state transition system $(\{p,q,r\},\{a\},\rightarrow)$ with $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$. Now, if $\{p\} : \langle a\rangle A$ is to be valid, then *either* $\{q\} : A$ *or* $\{r\} : A$ *or both* must be valid, but it is not possible to tell whether we should insist on this being $\{q\}$, $\{r\}$ or perhaps $\{q,r\}$. We have chosen to present this choice in a way which also allows for weakening, hence in **(R4)** U' is any set which satisfies $U \subseteq (\xrightarrow{\kappa})U'$, where

$$\xrightarrow{\kappa} U' = \{s \in S \mid \exists u \in U'\ \exists a \in \kappa.\ s \xrightarrow{a} u\}.$$

Notice, that **(R5)** could have been given in a completely analogous fashion, but we keep the current presentation because it is deterministic and the analogue of **(R4)** for the box-modality can be achieved as a derived rule through the weakening rule **(W)**.

(R6), (R7), (R8), (R9). The ν-rule **(R7)** expresses the reduction lemma and **(R6)** an easy consequence of the semantics of the ν-operator. The μ-rule **(R9)** is inspired by lemma 7.2.

(W). The weakening rule allows for very many choices! It is essential to the completeness of the system.

($\emptyset$). Included for convenience. It is derivable from the other rules.

(I). The identity rule making $\longrightarrow$ reflexive, is used later when defining $\Longrightarrow$.

The μ-rule **(R9)** might give rise to infinite sets of correctness assertions being generated. However, they all have the same form and we can expect that they to a large extent can be rewritten simultaneously, considering the index i merely as a parameterization of the correctness assertions. To formalize this idea we introduce a rewriting relation between (possibly infinite) sets of correctness assertions $\Longrightarrow$. It has one defining rule given in figure 7.4. As $\longrightarrow$ by **(I)** is reflexive the rule allows one to select some of the correctness assertions in Γ to be rewritten according to $\longrightarrow$ and leave others unchanged.

Let $\Longrightarrow^{*} = (\bigcup_{n\in\omega} \Longrightarrow^{n})$ where $\Longrightarrow^{0} = Id$, $\Longrightarrow^{n+1} = (\Longrightarrow \circ \Longrightarrow^{n})$. A correctness assertion $U : A$ is now provable on a transition system T with valuation V, written $\vdash_{T,V}\ U : A$ if this fact can be derived using $\Longrightarrow^{*}$, i.e.

$$\vdash_{T,V} U : A \Leftrightarrow_{\mathrm{def}} \{U : A\} \Longrightarrow^{*} \emptyset.$$

With this definition of a provably correct assertion, the rules are sound:

Theorem 7.1 (Soundness) *Suppose T is a labelled transition system with valuation V and A is a closed assertion. If $\vdash_{T,V} U : A$ then $\models_{T,V} U : A$.*

And complete:

Theorem 7.2 (Completeness) *Suppose T is a labelled transition system with valuation V and A is a closed assertion. If $\models_{T,V} U : A$ then $\vdash_{T,V} U : A$.*

The proof of these two theorems can be found in section 7.7.

7.5 Examples

In this section we will show how to apply the method to two small examples. We extend WPA with *value-passing* and suggest a notation for infinite sets of states which seems to be particularly useful for a class of *bounded processes*, processes which do not have arbitrarily, unboundedly evolving structure.

First, we assume that $\mathcal{A}$ is a set of neutral actions or channel names, and assume that $\mathbb{V}$ is a set of values. Then the set of basic actions is $Act = \mathcal{A} \cup \{a?v \mid a \in \mathcal{A}, v \in \mathbb{V}\} \cup \{a!v \mid a \in \mathcal{A}, v \in \mathbb{V}\}$. Prefixes π are now either input, output, or neutral prefixes:

$$\pi ::= a?v \mid a!e \mid a,$$

where $a \in \mathcal{A}$, v is a value variable, and e a value expression. Moreover, we add process constants parameterized by value expressions, such that the extended syntax for process terms is

$$t ::= \ldots \mid \pi.t \mid (\psi)t \mid C(e_1, \ldots, e_n),$$

where C denotes a process constant with arity n defined through an equation

$$C(v_1, \ldots, v_n) = t$$

where the free value variables of t are among $v_1, \ldots, v_n$ (we often abbreviate this as $\vec{v}$). Constant definitions can be mutually recursive (cf. remark 2.2 on page 22). Value expressions e are build from a set of operators, value variables $v \in \mathit{var}$, and constants $c \in \mathit{const}$. Guards, (ψ), are boolean expressions over predicates on the value expressions. Now, states are identified with closed process expressions, so sets of states are sets of processes, which we suggest can be described by

$$\vec{t}; \vec{\psi}(\vec{v})$$

where $\vec{t}$ is a list of process expressions, $\vec{\psi}$ a list of predicates, and $\vec{v}$ a list of free value-variables, which are implicitly assumed to be universally quantified. That is, tentatively the semantics of $\vec{t}; \vec{\psi}(\vec{v})$ is the set

$$[\![\vec{t}; \vec{\psi}(\vec{v})]\!] = \{\vec{t}[\vec{c}/\vec{v}] \mid \vec{\psi}[\vec{c}/\vec{v}], c_i \in \mathbb{V}\},$$

where $\vec{v}$ includes all the free variables of $\vec{t}$ and $\vec{\psi}$. An example when the values are natural numbers is

$$P, Q(n); n > 0, n \leq 3(n)$$

i.e. the set $\{P, Q(1), Q(2), Q(3)\}$. We will simply write $\vec{t}(\vec{v})$ instead of $\vec{t}; \vec{\psi}(\vec{v})$ when $\vec{\psi}$ is empty. Now, entailments will be on the form $\vdash \vec{t}; \vec{\psi}(\vec{v}) : A$ or $\vdash \vec{t}(\vec{v}) : A$.

The operational semantics is as before, with the following rules for the prefixes and guards:

$$a!v.t \overset{a!v}{\rightarrow} t$$

$$a?x.t \overset{a?v}{\rightarrow} t[v/x] \qquad v \in \mathbb{V}$$

$$\frac{t \overset{a}{\rightarrow} t}{(\psi)t \overset{a}{\rightarrow} t} \qquad \psi \text{ true}$$

Example 7.1 This is a classic example. It has been used to show that on a very simple transition system, $\mu X\{\}[.]X$ cannot be found as the ω-limit of its approximants, $F, [.]F, [.][.]F, \ldots$ the ordinal $\omega + 1$ is necessary. Define P and $Q(n)$ as follows:

$$\begin{array}{rcl} P & = & a?n.Q(n) \\ Q(n) & = & (n > 0)\tau.Q(n-1) \end{array}$$

So P inputs a number n on the channel a, and then proceeds by making n τ's. We will show that P always terminates, i.e. that all execution sequences are finite. This is expressed in the modal μ-calculus as $\mu X\{\}[.]X$. We rewrite as follows:

$$\begin{array}{lcl} P : \mu X\{\}[.]X & \overset{(\mathbf{R9})}{\longrightarrow} & P : [.]\mu X\{\}[.]X \\ & & \quad \text{with trivial singleton covering, arbitrary w.fr.} \\ & \overset{(\mathbf{R5})}{\longrightarrow} & Q(n)\ (n) : \mu X\{\}[.]X \\ & \overset{(\mathbf{R9})}{\longrightarrow} & \{Q(n) : [.]\mu X\{\sqsubset Q(n)\}[.]X\}_{n\in\omega} \\ & & \quad \text{with covering } \{Q(n)\}_{n\in\omega}\} \text{ and w.f.r.} \\ & & \quad\ Q(m) \sqsubset Q(n) \Leftrightarrow_{\text{def}} m < n. \\ & = & (Q(0) : [.]\mu X\{\sqsubset Q(0)\}[.]X), \{Q(n) : [.]\mu X \end{array}$$

$$\{\sqsubset Q(n)\}[.]X\}_{n>0}$$

$$\overset{(\mathbf{R5})}{\Longrightarrow} \quad (\emptyset : \mu X\{\sqsubset Q(0)\}[.]X), \{Q(n)\ (n) : [.]\dots\}_{n>0}$$

since $Q(0) \not\rightarrow$, hence $(Q(0)\ :\ [.]\mu\dots$
$\longrightarrow (\emptyset : \mu\dots)$

$$\overset{(\mathbf{\emptyset})}{\Longrightarrow} \quad \{Q(n) : [.]\mu X\{\sqsubset Q(0)\}[.]X\}_{n>0}$$

$$\overset{(\mathbf{R5})}{\Longrightarrow} \quad \{Q(n)\ :\ [.]\mu X\{\sqsubset Q(0)\}[.]X\}_{n>0}$$

$$\overset{(\mathbf{R8})}{\Longrightarrow} \quad \emptyset \quad \text{as } n-1<n$$

Notice, that the splitting of the ω-set of correctness assertions after the third step was strongly suggested to us by the guard $n > 0$ in the definition of $Q(n)$.

It is also worthwhile to observe that although we used a covering of singleton sets here, it is not always necessary to fall back on singletons. If we instead had the definition

$$\begin{array}{rcl} P & = & a?n.b?m.Q(n,m) \\ Q(n,m) & = & (n>0)c!m.b?m.Q(n-1,m) \end{array}$$

we could use the covering

$$\{\{Q(n,m) \mid m \in \omega\}\}_{n\in\omega}$$

and the w.f.r. $Q(n',m') \sqsubset Q(n,m) \Leftrightarrow_{\text{def}} n' < n$. □

Example 7.2 This is an example from Bradfield [16, p. 6].

Consider the following definition of a process M:

$$\begin{array}{rcl} M(A,B,C) & = & (A \geq 1)a.M(A,B+1,C) \\ & + & (A \geq 1)b.M(A-1,B,C+1) \\ & + & (B \geq 1 \wedge C \geq 1)c.M(A,B-1,C) \end{array}$$

The process $M(l,m,n)$ is really describing the firing sequence of a certain *Petri net* with l tokens on the *place A,* m tokens on place B, and n tokens on the place C, and the actions a, b, and c are *transitions* of the Petri net. (See Bradfield [16] for details.)

Using the previously defined notation, sets of states will now be described by

$$M(A,B,C);\vec{\psi}$$

For convenience, we will omit $M(A,B,C)$ and just write $\vec{\psi}$. The initial marking we consider is $A=1, B=0, C=0$ and we will show that c only happens finitely often, expressed as the assertion $\mu X\{\}\nu Y\{\}[c]X \wedge [a,b]Y$. Intuitively this is obvious: Either a fires indefinitely, increasing the number of tokens on B, or at some point b fires, and then only c can fire. As there is only a finite number of tokens on B when this happens and c removes one token whenever fired, it must eventually stop.

Formally, we show:

$$(A=1, B=0, C=0 : \mu X\{\}\nu Y\{\}[c]X \wedge [a,b]Y) \Longrightarrow^* \emptyset.$$

Let $X_0 = \mu X\{\}\nu Y\{\}[c]X \wedge [a,b]Y$ and rewrite as follows:

$(A=1, B=0, C=0 : X_0)$

$\xrightarrow{(\mathbf{W})}$ $A+C=1 : X_0$

$\xrightarrow{(\mathbf{R9})}$ $\{A+C=1, B=n \ : \ \nu Y\{\}[c]X_1 \wedge [a,b]Y\}_{n\in\omega}$

where $X_1 = \mu X\{A+C=1, B<n\}\nu Y\{\}[c]X \wedge [a,b]Y$

$\xRightarrow{(\mathbf{R7})}$ $\{A+C=1, B=n : [c]X_1 \wedge [a,b]Y_0\}_{n\in\omega}$

where $Y_0 = \nu Y\{A+C=1\}[c]X_1 \wedge [a,b]Y$

$\xRightarrow{(\mathbf{R2})}$ $\{A+C=1, B=n : [c]X_1,\ A+C=1, B=n : [a,b]Y_0\}_{n\in\omega}$

$\xRightarrow{(\mathbf{R5})}$ $\{A+C=1, B=n : [c]X_1\}_{n\in\omega}$,

$\{A+C=1, B=n, n=0 \Rightarrow C=1 : Y_0\}_{n\in\omega}$

$\xRightarrow{(\mathbf{R6})}$ $\{A+C=1, B=n : [c]X_1\}_{n>\omega}$

$\xRightarrow{(\mathbf{R5})}$ $\{A=0, B=n-1\ C=1 : X_1\}_{n\in\omega}$

$\xRightarrow{(\mathbf{R8})}$ $\emptyset$

It is essential to extend the sets of markings in the first weakening step in order to make the later application of rule **(R9)** successful. □

In the previous two examples, the processes involved were of a particular simple kind, they did not have 'evolving structure'. To be precise about this, let us define an operation ˆ which maps WPA process expressions with values to WPA process expressions without:

$$\begin{array}{rcl} \hat{nil} & = & nil \\ \widehat{\pi.t} & = & \hat{\pi}.\hat{t} \\ \widehat{(\psi)t} & = & \hat{t} \\ \widehat{t \upharpoonright \Lambda} & = & \hat{t} \upharpoonright \Lambda \\ \widehat{t\{\Xi\}} & = & \hat{t}\{\Xi\} \\ \widehat{t_0 + t_1} & = & \hat{t_0} + \hat{t_1} \\ \widehat{t_0 \times t_1} & = & \hat{t_0} \times \hat{t_1} \\ \widehat{C(\vec{e})} & = & C \end{array}$$

where for action prefixes: $\widehat{a?v} = a, \widehat{a!v} = \bar{a}, \hat{a} = a$.

Definition 7.1 A process P is *bounded* if the set

$$\{(\widehat{Q} \mid \exists n \exists a_1, \ldots, a_n .\ P \xrightarrow{a_1}\xrightarrow{a_2} \ldots \xrightarrow{a_n} Q\}$$

is finite. □

The notation we have used seems to be particularly well-suited for bounded processes, as all the reachable states can be described by a finite number of process expressions, together with a collection of constraints on the free value-variables. We claim that it is not difficult to see that each particular *state* can actually be described by a process expression and a finite number of constraints, but whether any *set of states* expressible in the modal μ-calculus can actually be described by finitely many constraints, yielding a relative completeness result, is another issue not addressed here.

7.6 Relation to the Tableau Method of Bradfield and Stirling

We have chosen to present the method as a set of rewrite rules. However, it is not difficult to give a presentation of the rewrite rules as 'goal-oriented'

$$\frac{U : \langle\kappa\rangle A}{U' : A} \qquad (U \subseteq (\overset{\kappa}{\to} U'))$$

$$\frac{U : [\kappa]A}{(U \overset{\kappa}{\to}) : A}$$

$$\frac{U : \mu X\{V\}A}{} \qquad (U \subseteq V)$$

$$\frac{U : \mu X\{V\}A}{\{U_i : A[\mu X\{V \cup (\sqsubset U_i)\}A/X]\}_{i\in I}} \qquad \begin{pmatrix} U \not\subseteq V \\ \bigcup U_i = U \\ \sqsubset \text{ w.f.r. on } U \end{pmatrix}$$

$$\frac{U : \nu X\{V\}A}{} \qquad (U \subseteq V)$$

$$\frac{U : \nu X\{V\}A}{U : A[\nu X\{V \cup U\}A/X]} \qquad (U \not\subseteq V)$$

Figure 7.3: Some of the rewrite rules presented as 'goal-oriented' proof rules.

proof rules (see figure 7.5). In general, a rewrite rule of the form

$$(U : A) \longrightarrow \Gamma \text{ if } C$$

gives rise to a proof rule

$$\frac{U : A}{\Gamma} \quad (C)$$

which has side condition C.

Besides the annotations on fixed-points which localizes validity, i.e. makes it independent of the proof tree, the main difference to the tableau method of Bradfield and Stirling [17, 15] is in the treatment of the minimum fixed-points. Whereas Bradfield and Stirling constructs a finite proof tree with certain non-trivial success criteria — a tableau — which for the minimum fixed-point involves determining, outside the system, well-foundedness of a relation *induced by the tableau*, we supply a well-founded relation on the states which is independent of the proof being constructed; and carry out the inductive reasoning *inside the system* as we proceed with the rules.

For the present method, building a proof tree, showing how rules are applied, is not an essential ingredient, but it could be used as an organizational trick that makes explicit where choices were taken and perhaps could be altered.

Another apparent difference is that the tableau method of Bradfield and Stirling constructs a finite proof tree, whereas the application of $\Longrightarrow$ seems to have an inherent infinite nature. However, the appealing feature of generating finite proof trees has the cost of pushing the infinite reasoning into the reasoning involved in showing well-foundedness of the relation induced by the tableau. Moreover, the infinite nature of $\Longrightarrow$ is only apparent. As the examples show the infinite reasoning performed with $\Longrightarrow$ is rather innocent; the correctness assertions all have the same form, so the proof proceeds in the same manner for each correctness assertion, and is thus more a means of proving 'parameterized' correctness assertions.

7.7 Proofs of Soundness and Completeness

In this section we show soundness (theorem 7.1) and completeness (theorem 7.2) of the method.

7.7.1 Soundness

In order to show soundness we assume that $\vdash U : A$, i.e. $\{U : A\} \Longrightarrow^* \emptyset$ and argue that $\models U : A$.

Proof (Soundness): Let the predicate Q be defined by

$$Q(n) \Leftrightarrow_{\text{def}} (\Gamma \Longrightarrow^n \emptyset) \Rightarrow \text{for all } (U : A) \in \Gamma.\ \models U : A.$$

We prove by induction on $n \in \omega$ that $Q(n)$ holds for all n, from which the theorem follows. The base case is trivial. As the only clause defining $\Longrightarrow$ is

$$\frac{\forall \gamma \in \Gamma.\ \gamma \longrightarrow \Delta_\gamma}{\Gamma \Longrightarrow \bigcup_{\gamma \in \Gamma} \Delta_\gamma} \quad (\Longrightarrow)$$

the inductive step amounts to showing that if $(U : A) \longrightarrow \Delta$ and $\Delta \longrightarrow^{n-1} \emptyset$ then $\models U : A$. By the induction hypothesis $\Delta \Longrightarrow^{n-1} \emptyset$ implies that for all $(U' : A') \in \Delta$ we have $\models U' : A'$, hence we must argue that if

$$(U : A) \longrightarrow \Delta \ \&\ \forall (U' : A') \in \Delta.\ \models U' : A'$$

then

$$\models U : A.$$

We consider each rule in turn.

(R1), (R2), (R3), (R4), (R5). Straightforward.

(R6), (R7). From the semantics of the annotated maximum fixed-point we deduce as follows

$$[\![\nu X\{V\}A]\!]\rho = \nu W.V \cup [\![A]\!]\rho[W/X] = V \cup [\![A]\!]\rho[\nu W..../X] \supseteq V.$$

Hence, certainly if $U \subseteq V$, we have $U \subseteq [\![\nu X\{V\}A]\!]\rho$ and therefore $\models U : \nu X\{V\}A$ proving soundness of **(R6)**. 0therwise, if $U \not\subseteq V$, the soundness of **(R7)** follows from lemma 7.1.

(R8), (R9). Rule **(R8)** is like for the minimum fixed-point above. The rule **(R9)** is sound by lemma 7.2.

(W), (∅), (I). Trivial.

□

7.7.2 Completeness

In order to show completeness we will need some facts about the ordinals, *On*. Let $<$ be the well-founded relation on *On*. Define for a monotone function $f : \mathcal{P}(S) \to \mathcal{P}(S)$ the set $\mu^\alpha f$ inductively as follows:

$$\begin{array}{rcll} \mu^0 f & = & \emptyset & \\ \mu^{\alpha+1} f & = & f(\mu^\alpha f) & \\ \mu^\lambda f & = & \bigcup_{\alpha<\Lambda} \mu^\alpha f & \text{for } \lambda \text{ a limit ordinal.} \end{array}$$

The following proposition shows, that the minimum fixed-point of a monotonic function f on a powerset can be found as the least upper bound of all the approximants $\mu^\alpha f$.

Proposition 7.1 *Let $\mathcal{P}(S)$ be a powerset, and assume $f : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ is a monotonic function. Then $\{\mu^{\alpha}\ f\}_{\alpha \in O_n}$ is an increasing sequence with*

$$\mu f = \bigcup_{\alpha \in On} \mu^{\alpha} f,$$

and there is a least ordinal β (the closure ordinal), *such that $\mu^{\beta} f = \mu^{\beta+1} f$ and*

$$\mu f = \mu^{\beta} f,$$

We denote this ordinal $cl(f)$.

Proof: The proposition holds in all complete lattices, consult e.g. Aczel [2] for a proof. □

In the completeness proof we construct a canonical proof which only need to unfold each fixed-point once. A simple property of the annotations on fixed-points that make this possible is captured by the following lemma.

Lemma 7.3 *Assume that V is a valuation with $V(Q_X) = W$. If A is an assertion with only one free variable X and*

$$\vdash_V U : A[Q_X/X]$$

then

$$\vdash_V U : A[\mu X\{W\}B/X]$$

and

$$\vdash_V U : A[\nu X\{W\}B/X].$$

Proof: Simple structural induction on A. Any application of the **(R1)**-rule is replaced by an application of **(R6)** or **(R8)**. □
Define the relation $\prec$ on closed assertions by $A' \prec A$ if and only if, $A' = A''[\vec{Q}/\vec{X}]$ for some proper subassertion A'' of A where $\vec{Q}$ is a vector of constants and $\vec{X}$ are the free variables of A''. Surely, $\prec$ is well-founded. We can now prove the completeness:

Proof (Completeness): We show that the following predicate $P(A)$ holds for all closed assertions A by induction on $\prec$:

$$P(A) \ \Leftrightarrow_{\text{def}} \vdash_V [\![A]\!] : A$$

Given an A, let $U = [\![A]\!]$ and consider the possible forms of A.

$A \equiv \nu X\{V\}B$. If $U \subseteq V$ the claim follows from rule **(R6)**. Otherwise, assume given a constant Q with valuation $V(Q) = U$. Then by the induction hypothesis

$$\vdash_V [\![B[Q/X]]\!] : B[Q/X]$$

which by lemma 7.3 implies

$$\vdash_V [\![B[Q/X]]\!] : B[\nu X\{V \cup U\}B/X]$$

Hence, as $[\![B[Q/X]]\!] = [\![B[\nu X\{V \cup U\}B/X]]\!] = U$ then

$$\vdash_V U : B[\nu X\{V \cup U\}B/X]$$

and the result follows from rule **(R7)**.

$A \equiv \mu X\{V\}B$. If $U \subseteq V$ the claim follows from rule **(R8)**. Otherwise, we use rule **(R9)** in the following way. Let $\psi(Z) = [\![B]\!]\rho[Z/X] \cup V$ for an arbitrary environment ρ. Let β be the closure ordinal of ψ, and let for all $u \in U$, α_u be the ordinal such that $\alpha_u + 1$ is the least ordinal with

$$u \in \mu^{\alpha_u+1}\psi$$

(Notice, that this *must* be a successor ordinal, as for a limit ordinal λ,

$$u \in \bigcup_{\gamma<\lambda} \mu^{\gamma}\psi$$

implies that there exists a $\gamma < \lambda$ such that $u \in \mu^{\gamma}\psi$. Moreover, $\alpha_u + 1$ can be no bigger than the closure ordinal β of ψ.)

Define the relation $\sqsubset$ on elements of $U = \mu\psi$ by $u' \sqsubset u$ iff $\alpha_{u'} < \alpha_u$. By the well-foundedness of the ordinals, $\sqsubset$ is a well-founded relation. Notice, that $\mu^{\alpha_u}\psi = (\sqsubset u)$.

Assume given a set of constants Q_u with valuation $V(Q_u) = V \cup (\sqsubset u)$. Then since $B[Q_u/X] \prec \mu X\{V\}B$ the induction hypothesis yields

$$\vdash_V [\![B[Q_u/X]]\!] : B[Q_u/X]$$

which by lemma 7.3 implies

$$\vdash_V [\![B[Q_u/X]]\!] \quad : \quad B[\mu X\{V \cup (\sqsubset u)\}B/X]. \tag{7.1}$$

Observe, that $u \in \mu^{\alpha_u+1}\psi = \psi(\mu^{\alpha_u}\psi) = \psi(\sqsubset u) \subseteq [\![B[Q_u/X]]\!]$ assuming $V(Q_u) = V \cup (\sqsubset u)$.

We can now proceed rewriting with $\Longrightarrow$ as follows:

$$\begin{array}{rcl}
[\![\mu X\{V\}B]\!] : \mu X\{V\}B & \longrightarrow & \{u : B[\mu X\{V \cup (\sqsubset u)\}B/X]\}_{u \in [\![\mu X\{V\}B]\!]} \\
& & \quad \text{by } (\mathbf{R9}) \\
& \Longrightarrow & \{[\![B[Q_u/X]]\!] : B[\mu X\{V \cup (\sqsubset u)\} \\
& & B/X]\}_{u \in [\![\mu X\{V\}B]\!]} \\
& & \quad \text{by } (\mathbf{W}) \text{ since } u \in [\![B[Q_u/X]]\!] \\
& \Longrightarrow & \emptyset \\
& & \quad \text{by } (7.1).
\end{array}$$

Remaining cases. They are all very straightforward.

We can now prove the completeness: For any $U \subseteq [\![A]\!]$ we use the weakening rule to rewrite as follows

$$(U : A) \longrightarrow ([\![A]\!] : A)$$

and then by the inductive proof above,

$$([\![A]\!] : A) \Longrightarrow^* \emptyset$$

□

7.8 Conclusion

When restricting ourselves to finite-state systems and using only singletons in the correctness assertions, we can replace the few choices that remains by finite disjunctions, thereby rediscovering the model checker of Winskel – in a version without negations, but with an explicit rule for the minimum fixed-point. Note, however that for the finite case, the algorithms in chapter

5 are more efficient. One short-coming of the method presented so far, is the inability to show that $\models U : A$ does *not hold*. The rules are not very appropriate for this; one has to show that all the possible choices lead to false expressions. An attempt to remedy this, could be through making explicit the choices as – rather large – disjunctions and then appealing to external methods for showing that all the disjuncts rewrite to false. This is certainly not appealing, especially because some of the (wrong) choices could lead to infinite rewriting sequences, making the task almost impossible, and at least requiring very strong external reasoning.

A more obvious attempt would be to simply try to show that U satisfies another assertion making $\models U : A$ impossible. If U is a singleton $\{u\}$, this is quite easy as

$$\not\models \{u\} : A \Leftrightarrow \models \{u\} : \neg A$$

where we have introduced negation with semantics $[\![\neg A]\!] = S\ [\![A]\!]$, i.e. the complement of A. This is not the case for general U, but we instead observe that

$$\not\models U : A \Leftrightarrow \exists U'(\emptyset \neq U' \subseteq U). \models U' : \neg A.$$

Instead of introducing a new rule for negation – which is just as difficult as to cope with showing non-validity – we consider $\neg A$ to be simply an operation on assertions that dualizes every operator in A (taking constants to new constants denoting their complement, $\langle\kappa\rangle$ to $[\kappa]$, μ to ν etc.), thereby making the method applicable as it is.

Chapter 8

Categorical Models for an Intuitionistic Modal μ-Calculus

8.1 Introduction

One of the major open problems in connection with the modal μ-calculus, is the question of whether a finite axiomatization exists. Kozen [49] gave a finite axiomatization for a sublogic consisting of 'aconjunctive' assertions, but for the full calculus the problem remains open.

The aim of this chapter is to give a completeness result using methods from categorical logic and show how to recast the problem of finding a complete axiomatization with respect to Kripke-like models as a problem of cutting down the set of models allowed by the categorical framework. We define a class of *monotone transition systems*, which seems to be a suitable candidate for a non-trivial completeness result, and for which we can find concrete instances from work in the process algebraic community.

We will use an intuitionistic version of the calculus, although as we point out in the concluding section everything should carry through to the classical calculus. The choice of considering an intuitionistic version is somewhat arbitrary, but emphasizes the generality of the categorical approach.

To stress the difference to the standard and extended calculus considered in the rest of the thesis, we have chosen to use lowercase greek letters for assertion in the intuitionistic modal μ-calculus, and use lowercase letters $p, q, \ldots$ for assertion variables.

8.2 Logic

For simplicity of presentation we will only consider unrelativized modalities $\Diamond$ and $\Box$. We claim that using relativized modalities $\langle\alpha\rangle$ and $[\alpha]$ would not change the fundamental issues.

The syntax of the intuitionistic modal logic μIK is defined by the following grammar:

$$\psi ::= p \mid \bot \mid \top \mid \psi \wedge \psi \mid \psi \vee \psi \mid \Diamond\psi \mid \Box\psi \mid \mu p.\psi \mid \nu p.\psi$$

The set of propositional variables ranged over by p is assumed to be a countable set $\{p_0, p_1, \dots\}$. The two fixed-point operators $\mu p.\psi$ and $\nu p.\psi$ bind the variable p and denote minimum and maximum fixed-points of the function of p described by the body ψ. Let *IAssn* be the set of assertions.

We will present a theory for μIK such that the fragment of μIK without the fixed-points is equivalent to an implication-free fragment of the minimum modal logic IK given by Plotkin and Stirling [72] and the fragment without fixed-points and modalities is a usual implication-free first order intuitionistic propositional logic. The theory for μIK will be presented as sequents closed under rules of deduction. The sequents have the general form

$$\Gamma \vdash \varphi$$

where Γ is a finite set of formulas, hence $\vdash \subseteq$ *Fin(IAssn)* $\times$ *IAssn* is the least relation between finite sets of assertions and assertions, closed under the rules of figure 8.1, 8.2, and 8.3. (The rules for the intuitionistic propositional fragment is taken from [70], with a rule added for $\top$ necessary due to the absence of implication.) The rules are presented in natural deduction style.

We will now define a class of categorical models, called *μIK-categories*, for the logic. A μIK-category will be a category with finite products, an internal distributive lattice object, K-modalities, and families of operators μ_X and ν_X.

Definition 8.1 Assume $\mathbf{C}$ is a category with finite products. An *internal distributive lattice object* of $\mathbf{C}$ is an object D of $\mathbf{C}$ equipped with the morphisms

$$\begin{aligned} \bot, \top &: \mathbf{1} \to D, \\ \wedge, \vee &: d \times D \times D, \end{aligned}$$

$$\frac{\Gamma \vdash \psi}{\Gamma\varphi \vdash \psi}\ (W) \qquad \frac{}{\psi \vdash \psi}\ (Id)$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \psi}\ (\bot E) \qquad \frac{}{\Gamma \vdash \top}\ (\top I)$$

$$\frac{\Gamma \vdash \psi \quad \Delta \vdash \varphi}{\Gamma\Delta \vdash \psi \wedge \varphi}\ (\wedge I) \qquad \frac{\Gamma \vdash \psi \wedge \varphi}{\Gamma \vdash \psi}\ (\wedge E_1) \qquad \frac{\Gamma \vdash \psi \wedge \varphi}{\Gamma \vdash \varphi}\ (\wedge E_2)$$

$$\frac{\Gamma \vdash \psi}{\Gamma \vdash \psi \vee \varphi}\ (\vee I_1) \qquad \frac{\Gamma \vdash \varphi}{\Gamma \vdash \psi \vee \varphi}\ (\vee I_2)$$

$$\frac{\Gamma\psi \vdash \theta \quad \Delta\varphi \vdash \theta \quad \Theta \vdash \psi \vee \varphi}{\Gamma\Delta\Theta \vdash \theta}\ (\vee E)$$

Figure 8.1: Rules for the intuitionistic propositional fragment.

$$\frac{\psi \vdash \varphi}{\Diamond\psi \vdash \Diamond\varphi}\ (\Diamond I) \qquad \frac{\Gamma \vdash \Diamond\bot}{\Gamma \vdash \bot}\ (\Diamond E)$$

$$\frac{}{\Gamma \vdash \Box\top}\ (\Box I_1) \qquad \frac{\psi \vdash \varphi}{\Box\psi \vdash \Box\varphi}\ (\Box I_2)$$

$$\frac{\Gamma \vdash \Diamond(\psi \vee \varphi)}{\Gamma \vdash \Diamond\psi \vee \Diamond\varphi}\ (\Diamond\vee) \qquad \frac{\Gamma \vdash \Box\psi \wedge \Box\varphi}{\Gamma \vdash \Box(\psi \wedge \varphi)}\ (\Box\wedge)$$

$$\frac{\Gamma \vdash \Box\psi \wedge \Diamond\varphi}{\Gamma \vdash \Diamond(\psi \wedge \varphi)}\ (\Box\Diamond)$$

Figure 8.2: Rules for the modalities.

such that the following equations hold for all morphisms $a, b, c : X \longrightarrow D$,

(i)	$(a \vee b) \vee c = a \vee (b \vee c)$	(i')	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$
(ii)	$a \vee b = b \vee a$	(ii')	$a \wedge b = b \wedge a$
(iii)	$a \vee a = a$	(iii')	$a \wedge a = a$
(iv)	$a \vee (a \wedge b) = a$	(iv')	$a \wedge (a \vee b) = a$
(v)	$a \vee \bot = a$	(v')	$a \wedge \top = a$

Usually in algebra these equations are called the *associative laws* (i), (i'), *the commutative laws* (ii), (ii'), *the idempotency laws* (iii), (iii'), *the absorption laws* (iv), (iv'), and *the unit laws* (v), (v').

To be more categorically precise all the equations should be stated as

$$\frac{}{\varphi[\mu p.\varphi/p] \vdash \mu p.\varphi}\ (\mu 1) \qquad \frac{\varphi[\psi/p] \vdash \psi}{\mu p.\varphi \vdash \psi}\ (\mu 2)$$

$$\frac{}{\nu p.\varphi \vdash \varphi[\nu p.\varphi/p]}\ (\nu 1) \qquad \frac{\psi \vdash \varphi[\psi/p]}{\psi \vdash \nu p.\varphi}\ (\nu 2)$$

Figure 8.3: Rules for the fixed-points.

commutative diagrams in **C**. There is, however, an obvious way of reading the equations as diagrams, which we indicate by two examples. Equation (i) corresponds to: For all $a, b, c : X \to D$ the following diagram commutes,

$$\begin{tikzcd}
X \arrow[r, "{\langle a,\langle b,c\rangle\rangle}"] \arrow[d, "{\langle\langle a,b\rangle,c\rangle}"'] & D \times (D \times D) \arrow[r, "id \times \vee"] & D \times D \arrow[d, "\vee"] \\
(D \times D) \times D \arrow[r, "\vee \times id"] & D \times D \arrow[r, "\vee"] & D
\end{tikzcd}$$

Equation (iv) corresponds to the diagram:

$$\begin{tikzcd}
X \arrow[d, "{\langle a,\langle a,b\rangle\rangle}"'] \arrow[rrd, "a"] & & \\
D \times (D \times D) \arrow[r, "id \times \wedge"] & D \times D \arrow[r, "\vee"] & D
\end{tikzcd}$$

Notice, that we can define a partial order $\leq$ as $a \leq b \Leftrightarrow_{\text{def}} a \vee b = b$ or equivalently (by the absorption laws) as $a \leq b \Leftrightarrow_{\text{def}} a \wedge b = a$.

Definition 8.2 A category **C** with an internal distributive lattice object D has K-modalities, if there exists morphisms $\Box, \Diamond : D \to D$ satisfying the following equations:

$$\begin{array}{llcl}
(x) & \Box\psi \wedge \Box\varphi & = & \Box(\psi \wedge \varphi) \\
(xi) & \Box\top & = & \top \\
(xii) & \Diamond\psi \vee \Diamond\varphi & = & \Diamond(\psi \vee \varphi) \\
(xiii) & \Diamond\bot & = & \bot \\
(xiv) & \Box\psi \wedge \Diamond\varphi & = & \Box\psi \wedge \Diamond(\psi \wedge \varphi)
\end{array}$$

Again these should be expressed as diagrams to be truly categorical. Writing out equation (x) in more detail suggests how:

$$\wedge \circ \langle \Box \circ \psi, \Box \circ \varphi \rangle = \Box \circ \wedge \circ \langle \psi, \varphi \rangle$$

$\Box$

Instead of (xiv) often the inequality

$$\Box\psi \wedge \Diamond\varphi \leq \Diamond(\psi \wedge \varphi),$$

is used for expressing the relationship between the two modalities. These two formulations can easily be shown to be equivalent using the rules in definition 8.1. Notice, that we do not include the dual equation

$$\Diamond\psi \vee \varphi = \Diamond\psi \vee \Box(\psi \vee \varphi),$$

simply because it is not valid for the monotone transition system models we are going to consider in section 8.4, and therefore would require more restricted models to be sound. It could be added without complications for the categorical models.

It is useful to observe that (x) and (xii) imply that the modalities are monotonic with respect to $\leq$. For $\Box$ the argument is as follows:

$$\begin{array}{lcll}
\psi \leq \varphi & \Rightarrow & \psi \wedge \varphi = \psi & \text{by def.} \\
& \Rightarrow & \Box(\psi \wedge \varphi) = \Box\psi & \\
& \Rightarrow & \Box\psi \wedge \Box\varphi = \Box\psi & \text{by eqn. } (x) \\
& \Rightarrow & \Box\psi \leq \Box\varphi & \text{by def.}
\end{array}$$

Now, let **C** be a category with an internal distributive lattice object, and define $\leq_X$ on morphisms of **C** by

$$f \leq_X g \Leftrightarrow_{\text{def}} \wedge \circ \langle f, g \rangle = f,$$

where $f, g : X \to D$. With this ordering the rules of definition 8.1 essentially turn each hom-set $\mathbf{C}(X, D)$ into a distributive lattice. Moreover, each morphism $f : X' \to X$ induces a homomorphism of distributive lattices

$$f^* : \mathbf{C}(X, D) \to \mathbf{C}(X', D)$$

by $f^* = \_ \circ f$.

Definition 8.3 In a category $\mathbf{C}$ with finite products and internal distributive lattice object D we define: A morphism $f : X \times D \to D$ is *monotonic* if for all $g : Y \to X$ and $h, k : Y \to D$,

$$h \leq_Y k \Rightarrow f \circ \langle g, h \rangle \leq_D f \circ \langle g, k \rangle.$$

The morphism g is needed to take care of the 'context' X, and hence this definition requires the inequality to hold in all contexts.

A morphism $x : X \to D$ is a *pre-fixed point* of a monotonic morphism $f : X \times D \to D$ if

$$f \circ \langle id, x \rangle \leq_X x.$$

□

We are now ready to state the requirements to the fixed-point operators in the category. It will be given as an indexed family of operators μ_X and ν_X.

Definition 8.4 A *family of minimum fixed-point finders* is a family of operators μ_X indexed by the objects X of $\mathbf{C}$, which to each monotonic morphism $f : X \times D \to D$ associates a morphism $\mu_X(f) : X \to D$ such that:

- μ_X is natural in X, that is for all morphisms $f : X \times D \to D, g : Y \to X$:

  $$\mu_Y(f \circ (g \times id)) = \mu_X(f) \circ g$$

  (This is required to ensure that μ_X commutes with substitution.)

- For all monotonic morphisms $f : X \times D \to D$, $\mu_X(f)$ is a pre-fixed point and it is the least such pre-fixed point, that is for all $x : X \to D$,

$$f \circ \langle id, x\rangle \leq_X x \Rightarrow \mu_X(f) \leq_X x$$

$\Box$

A completely dual definition gives a notion of family of maximum fixed-point finders: A morphism $x : X \to D$ is a *post-fixed point* of a monotonic morphism $f : X \times D \to D$ if

$$x \leq_X f \circ \langle id, x\rangle.$$

Hence ν_X must be natural in X and for all monotonic $f : X \times D \to X$, $\nu_X(f)$ is a post-fixed point and it is the greatest such post-fixed point, that is for all $x : X \to D$,

$$x \leq_X f \circ \langle id, x\rangle \Rightarrow x \leq_X \nu_X(f).$$

Definition 8.5 A *μIK-category* is a category $\mathbf{C}$ with finite products, an internal distributive lattice object, K-modalities, and families of minimum and maximum fixed-point operators. $\Box$

Given a μIK-category $\mathbf{C}$ with the internal distributive lattice object D, we can interpret the logic μIK in $\mathbf{C}$, thereby giving a rather general class of models which are both sound and complete for the logic. To define the interpretation we will need to consider *assertions in context* $\psi\ (\vec{p})$, where $\vec{p} = (p_1, \dots, p_n)$ is a tuple of distinct variables. For an assertion in context to be well-formed we require all the free variables of ψ to appear among the variables of $\vec{p}$. For such an assertion $\psi\ (\vec{p})$ we define a morphism

$$[\![\psi\ (\vec{p})]\!] : D^n \to D$$

inductively on the structure of ψ:

$$\begin{array}{rcl}
[\![p_i(\vec{p})]\!] & = & \pi_i : D^n \to D \\
[\![\bot(\vec{p})]\!] & = & \bot\circ\ !_{D_n} : D^n \to \mathbf{1} \to D \\
[\![\top(\vec{p})]\!] & = & \top\circ\ !_{D_n} : D^n \to \mathbf{1} \to D \\
[\![\psi \wedge \varphi(\vec{p})]\!] & = & \wedge \circ \langle [\![\psi\ (\vec{p})]\!], [\![\varphi\ (\vec{p})]\!]\rangle : D^n \to D \times D \to D \\
[\![\psi \vee \varphi(\vec{p})]\!] & = & \vee \circ \langle [\![\psi\ (\vec{p})]\!], [\![\varphi\ (\vec{p})]\!]\rangle : D^n \to D \times D \to D \\
[\![\Diamond\psi(\vec{p})]\!] & = & \Diamond \circ [\![\psi\ (\vec{p})]\!] : D^n \to D \to D \\
[\![\Box\psi(\vec{p})]\!] & = & \Box \circ [\![\psi\ (\vec{p})]\!] : D^n \to D \to D \\
[\![\mu q.\psi(\vec{p})]\!] & = & \mu_{D^n}([\![\psi\ (\vec{p}q)]\!]) : D^n \to D \\
[\![\nu q.\psi(\vec{p})]\!] & = & \nu_{D^n}([\![\psi\ (\vec{p}q)]\!]) : D^n \to D
\end{array}$$

For this interpretation to be well-defined on the fixed-points we must argue that $[\![\psi\ (\vec{p}q)]\!] : D^n \times D \to D$ is monotonic, however, as no implication is present all operators are monotonic so this follows easily by structural induction on ψ.

Note in particular that for each closed assertion ψ we can use the empty context to get a well-formed assertion in context and obtain the global element $[\![\psi]\!] =_{\text{def}} [\![\psi\ ()]\!] \in \mathbf{C}(\mathbf{1}, D)$. In general for a *sequent in context* $\Gamma \vdash \psi\ (\vec{p})$, subject to the well-formedness criterion that all free variables of Γ and ψ must be in $\vec{p}$, we define a semantic entailment relation by

$$\Gamma \models_{\mathbf{C}} \psi\ (\vec{p}) \Leftrightarrow_{\text{def}} \bigvee_{\gamma\in\Gamma} [\![\gamma\ (\vec{p})]\!]_{\mathbf{C}} \leq_{D^n} [\![\psi(\vec{p})]\!]_{\mathbf{C}}.$$

With this interpretation the class of μIK-categories provides sound and complete models for the logic μIK.

Theorem 8.1 (Soundness of μIK wrt. μIK-categories)
If $\Gamma \vdash \psi\ (\vec{p})$ then for all μIK-categories $\mathbf{C}$, $\Gamma \models_{\mathbf{C}} \psi\ (\vec{p})$.

Theorem 8.2 (Completeness of μIK wrt. μIK-categories)
If for all μIK-categories $\mathbf{C}$, $\Gamma \models_{\mathbf{C}} \psi\ (\vec{p})$ then $\Gamma \vdash \psi\ (\vec{p})$.

The next section will be devoted to showing these theorems.

8.3 Proofs

We will often make use of the property captured by the lemma below that composition in the categorical model **C** corresponds to syntactic substitution in the logic.

Lemma 8.1 (Substitution lemma) *Let* **C** *be a μIK-category with internal distributive lattice object D. Then*

$$[\![\psi[\varphi/q](\vec{p})]\!]_{\mathbf{C}} = [\![\psi(\vec{p}q)]\!]_{\mathbf{C}} \circ \langle id_{D_n}, [\![\varphi(\vec{p})]\!]_{\mathbf{C}} \rangle$$

where q is not in $\vec{p}$ (and therefore not free in φ).

Proof: Straightforward by structural induction on ψ. $\Box$

8.3.1 Soundness

The proof of soundness is standard. Each rule is verified separately.

Proof (Soundness, Theorem 8.1): We consider a few characteristic cases. The remaining rules are all similar or simpler.

Rule ($\Box\Diamond$). We assume that $\Gamma \models_{\mathbf{C}} \Box\psi \wedge \Diamond\varphi$ and must argue that $\Gamma \models_{\mathbf{C}} \Diamond(\psi \wedge \varphi)$. Assume $\vec{p}$ is a context making the sequent well-defined. From the definition of $\models_{\mathbf{C}}$ we have:

$$\Gamma \models_{\mathbf{C}} \Box\psi \wedge \Diamond\varphi\ (\vec{p}) \Leftrightarrow \textstyle\bigwedge_{\gamma\in\Gamma}[\![\gamma\ (\vec{p})]\!]_{\mathbf{C}} \leq_{D^n} [\![\Box\psi \wedge \Diamond\varphi\ (\vec{p})]\!]_{\mathbf{C}}$$

We now use the equation (xiv) to rewrite the right-hand side:

$$\begin{array}{rcll} [\![\Box\psi \wedge \varphi\ (\vec{p})]\!]_{\mathbf{C}} & = & \Box[\![\psi\ (\vec{p})]\!]_{\mathbf{C}} \wedge \Diamond[\![\varphi\ (\vec{p})]\!]_{\mathbf{C}} & \text{by def.} \\ & = & \Box[\![\psi\ (\vec{p})]\!]_{\mathbf{C}} \wedge \Diamond([\![\psi\ (\vec{p})]\!]_{\mathbf{C}} \wedge [\![\varphi\ (\vec{p})]\!]_{\mathbf{C}}) & \text{by } (xiv) \\ & \leq_{D^n} & \Diamond([\![\psi\ (\vec{p})]\!]_{\mathbf{C}} \wedge [\![\varphi\ (\vec{p})]\!]_{\mathbf{C}}) & \\ & = & [\![\Diamond(\psi \wedge \varphi\ (\vec{p})]\!]_{\mathbf{C}}) & \text{by def.} \end{array}$$

hence by definition of $\models_{\mathbf{C}}$,

$$\bigwedge_{\gamma\in\Gamma}[\![\gamma\ (\vec{p})]\!]_{\mathbf{C}} \leq_{D^n} [\![\Diamond(\psi\wedge\varphi)\ (\vec{p})]\!]_{\mathbf{C}} \Rightarrow \Gamma \models_{\mathbf{C}} \Diamond(\psi\wedge\varphi)\ (\vec{p}).$$

Rule ($\mu 1$). We will show that $\varphi[\mu p.\varphi/p] \models_{\mathbf{C}} \mu p.\varphi$, i.e.

$$[\![\varphi[\mu p.\varphi/p]\ (\vec{p})]\!]_{\mathbf{C}} \leq_{D^n} [\![\mu p.\varphi\ (\vec{p})]\!]_{\mathbf{C}}$$

where $\vec{p} = (p_1, \ldots, p_n)$ is a context making the assertions well-formed. Now,

$$\begin{array}{lcl}
[\![\varphi[\mu p.\varphi p](\vec{p})]\!]_{\mathbf{C}} & = & [\![\varphi(\vec{p}p)]\!]_{\mathbf{C}} \circ \langle id_{D^n}[\![\mu p.\varphi(\vec{p}p)]\!]_{\mathbf{C}}\rangle \\
 & & \quad \text{by the substitution lemma} \\
 & = & [\![\varphi(\vec{p}p)]\!]_{\mathbf{C}} \circ \langle id_{D^n}, \mu_{D^n}([\![\varphi(\vec{p})]\!]_{\mathbf{C}})\rangle \\
 & & \quad \text{by definition} \\
 & \leq_{D^n} & \mu_{D^n}([\![\varphi(\vec{p}p)]\!]_{\mathbf{C}}) \\
 & & \quad \text{as } \mu_{D^n}(f) \text{ is a pre-fixed point of } f \text{ in } \mathbf{C} \\
 & = & [\![\mu p.\varphi\ (\vec{p})]\!]_{\mathbf{C}}
\end{array}$$

Rule ($\mu 2$). Assume that $\varphi[\psi/p] \models_{\mathbf{C}} \psi$. Hence,

$$\begin{array}{lcl}
[\![\varphi[\psi/p]\ (\vec{p})]\!]_{\mathbf{C}} \leq_{H^n} [\![\psi\ (\vec{p})]\!]_{\mathbf{C}} & \Rightarrow & [\![\varphi\ (\vec{p}p)]\!]_{\mathbf{C}} \circ \langle id_{H^n}[\![\psi\ (\vec{p})]\!]_{\mathbf{C}} \leq_{H^n} [\![\psi\ (\vec{p})]\!]_{\mathbf{C}}\rangle \\
 & & \quad \text{by the substitution lemma} \\
 & \Rightarrow & (\mu_{H^n}[\![\varphi(\vec{p}p)]\!]_{\mathbf{C}}) \leq_{H^n} [\![\psi(\vec{p})]\!]_{\mathbf{C}} \\
 & & \quad \text{as } \mu_{H^n} \text{ yields least fixed-points} \\
 & \Rightarrow & \mu p.\varphi \models_{\mathbf{C}} \psi \\
 & & \quad \text{by definition of } \models_{\mathbf{C}}
\end{array}$$

□

8.3.2 Completeness

Proof (Completeness, Theorem 8.2): We use a construction similar to the construction of a Lindenbaum algebra for the case of boolean algebra. First, define the equivalence $\sim$ on assertions by

$$\psi \sim \varphi \Leftrightarrow_{\text{def}} \psi \vdash \varphi \text{ and } \varphi \vdash \psi$$

Let $D(n)$ be the set of equivalence classes $[\psi]$ under $\sim$ of assertions with free variables in $\{p_1, \ldots, p_n\}$. On $D(n)$ we define the partial order $\precsim$ as $[\psi] \precsim [\varphi] \iff \psi \vdash \varphi$ which makes $D(n)$ into (at least) a distributive lattice, with operators induced by the syntactic counterparts, i.e. bottom is $[\bot]$, top is $[\top]$, meet of $[\psi]$ and $[\varphi]$ is $[\psi \wedge \varphi]$ etc. We can now define a category **D** as follows.

Objects are numbers $n \in \omega$ written U^n.

Morphisms $U^n \to U^m$ are m-tuples of elements of $D(n)$, i.e. equivalence classes of lists $\Psi_1, \ldots, \Psi_m$ where each formula Ψ_i is a formula with free variables contained in $\{p_1, \ldots, p_n\}$ under the equivalence relation $(\Psi_1, \ldots, \Psi_m) \sim (\Psi'_1, \ldots, \Psi'_m)$ iff $\Psi_i \sim \Psi'_i$: for $1 \leq i \leq m$.

Composition is substitution on representative formulas:

$$[\vec{\Psi}] \circ [\vec{\Phi}] =_{\text{def}} [\vec{\Theta}]$$

where $\Theta_i = \Psi_i[\Phi_1/p_1, \ldots, \Phi_n/p_n]$ i.e. Θ is constructed from Ψ by simultaneously substituting all the Φ_i's for the p_i's.

Identity is

$$[(p_1, \ldots, p_n)] : U^n \to U^n$$

We observe that **D** has finite products: The terminal object $\mathbf{1}$ is U^0, and the unique map $! : U^n \to U^0$ is the empty tuple. The product $U^n \times U^m$ is U^{n+m} where $\pi_1 = [p_1, \ldots, p_n]$, $\pi_2 = [p_{n+1}, \ldots, p_{n+m}]$ and for morphisms $[\vec{\Phi}] : U^k \to U^n$, $[\vec{\Psi}] : U^k \to U^m$ we take

$$\langle [\vec{\Phi}], [\vec{\Psi}] \rangle = [(\Phi_1, \ldots, \Phi_n, \Psi_1, \ldots, \Psi_m)] : U^k \to U^{n+m}.$$

In **D** U is an internal distributive lattice object with K-modalities: $\top = [\top] : \mathbf{1} \to U$, $\wedge = [p_1 \wedge p_2] : U \times U \to U$, etc. Commutativity of the diagrams follow straightforward from the rules of the logic; it amounts to showing that the equational presentation is sound for the sequent presentation. We consider a few typical cases.

Equation (xxi)**.** We must argue that $\Diamond a \vee \Diamond b = \Diamond(a \vee b)$ for all morphisms $a, b : U^n \to U$. Now $a = [\psi], b = [\varphi]$ for some $\psi, \varphi \in D(n)$ and

$$\begin{array}{rcl} \Diamond a \vee \Diamond b &=& \Diamond[\psi] \vee \Diamond[\psi] \\ &=& [p_1 \vee p_2] \circ \langle [\Diamond p_1] \circ [\psi], [\Diamond p_1] \circ [\varphi] \rangle \\ && \quad \text{by def. of } \vee : U \times U \to U, \text{ and } \Diamond : U \times U \\ &=& [\Diamond\psi \vee \Diamond\varphi] \\ && \quad \text{by definition of composition in } \mathbf{D} \end{array}$$

Similarly, $\Diamond(a \vee b) = [\Diamond(\psi \vee \varphi)]$ so we must argue that $\Diamond\psi \vee \Diamond\varphi \sim \Diamond(\psi \vee \varphi)$. The direction $\Diamond(\psi \vee \varphi) \vdash \Diamond\psi \vee \Diamond\varphi$ follows directly from (Id) and $(\Diamond\vee)$. For the other direction we have the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\psi \vdash \psi}\,(Id)}{\psi \vdash \psi \vee \varphi}\,(\vee I_1)}{\Diamond\psi \vdash \Diamond(\psi \vee \varphi)}\,(\Diamond I) \quad \dfrac{\vdots}{\Diamond\varphi \vdash \Diamond(\psi \vee \varphi)}\,(\Diamond I) \quad \dfrac{}{\Diamond\psi \vee \Diamond\varphi \vdash \psi \vee \Diamond\varphi}\,(Id)}{\Diamond\psi \vee \Diamond\varphi \vdash \Diamond(\psi \vee \varphi)}\,(\vee E)$$

Equation (xiv) . As above this amounts to showing that $\Box\psi\Diamond\varphi \vdash \Diamond(\psi\wedge\varphi)$ and $\Box\psi\Diamond(\psi \wedge \varphi) \vdash \Box\psi \wedge \Diamond\varphi$. We consider the first, the second follows easily from monotonicity of $\Diamond$:

$$\dfrac{\dfrac{}{\Box\psi \vdash \Box\psi}\,(Id) \quad \dfrac{\dfrac{\dfrac{}{\Box\psi \vdash \Box\psi}\,(Id) \quad \dfrac{}{\Diamond\varphi \vdash \Diamond\varphi}\,(Id)}{\Box\psi\Diamond\varphi \vdash \Box\psi \vee \Diamond\varphi}\,(\wedge I)}{\Box\psi\Diamond\varphi \vdash \Diamond(\psi \wedge \varphi)}\,(\Box\Diamond)}{\Box\psi\Diamond\varphi \vdash \Box\psi \wedge \Diamond(\psi \wedge \varphi) \wedge I}\,(\wedge I)$$

Moreover, $\mathbf{D}$ has families of minimum and maximum fixed-point finders. Let $[\psi] : U^n \times U \to U$ be a monotonic morphism in $\mathbf{D}$. Define $\mu_{U^n}([\psi]) : U^n \to U$ by $\mu_{U^n}([\psi]) = [\mu p_{n+1}.\psi]$. We must check that this family of operators μ_X is natural in X and indeed yields least fixed-points in $\mathbf{D}$.

Naturality of μ_X. We must show that for all $[\vec{\Phi}] : U^n \to U^m$, $[\psi] : U^m \times U \to U$,

$$\mu_{U^n}([\psi] \circ ([\vec{\Phi}] \times id_U)) = \mu_{U^m}([\psi]) \circ [\vec{\Phi}].$$

We rewrite as follows:

$$\begin{array}{rcl} \mu_{U^n}([\psi]\circ([\vec{\Phi}]\times id_U)) & = & [\mu p_{n+1}.(\psi/[\vec{\Phi}/(p_1,\dots,p_m),p_{n+1}/p_{m+1}])] \\ & = & [(\mu p_{m+1}.\psi)[\Phi/(p_1,\dots,p_m)]] \\ & & \quad \text{as } p_{n+1} \text{ is not free in } \vec{\Phi} \\ & = & \mu_{U^m}([\psi])\circ[\vec{\Phi}] \end{array}$$

Fixed-point property of μ_X. Assume that $[\psi] : U^n \times U \to U$ is a monotonic morphism, we must argue that $\mu_{U^n}[\psi] = [\mu p_{n+1}.\psi]$ is a pre-fixed point, hence that

$$[\psi]\circ\langle id_{U^n},[\mu p_{n+1}.\psi]\rangle \leq_{U^n} [\mu p_{n+1}.\psi]$$

which by the substitution lemma is equivalent to showing that

$$[\psi[\mu p_{n+1}.\psi/p_{n+1}]] \precsim [\mu p_{n+1}.\psi]$$

But by (μ1) we have

$$\psi[\mu p_{n+1}.\psi/p_{n+1}] \vdash \mu p_{n+1}.\psi$$

from which the result follows.

Hence $\mathbf{D}$ is a μIK-category, and the interpretation of the logic given by $[\![\ ]\!]_{\mathbf{D}}$ coincides with $[\ ]$, i.e. for a context $\vec{p} = (p_1,\dots,p_n)$,

$$[\![\psi(\vec{p})]\!]_{\mathbf{D}} = [\psi] : U^n \to U,$$

which can easily be shown by structural induction on ψ. Furthermore, $\precsim$ coincides with $\leq_{D^n}$:

$$\begin{array}{rcll} [\psi]\leq_{U^n}[\varphi] & \Leftrightarrow & [\psi]\wedge[\varphi]=[\psi] & \text{by definition of } \leq_{U^n} \\ & \Leftrightarrow & [p_1\wedge p_2]\circ\langle[\psi],[\varphi]\rangle=[\psi] & \text{by definition of } \wedge : U\times U\to U \\ & \Leftrightarrow & [\psi\wedge\varphi]=[\psi] & \text{by definition of composition in } \mathbf{D} \\ & \Leftrightarrow & \psi\wedge\varphi\sim\psi & \\ & \Leftrightarrow & \psi\wedge\varphi\vdash\psi \text{ and } \psi\vdash\psi\wedge\varphi & \text{by definition of } \sim \\ & \Leftrightarrow & \psi\vdash\varphi & (^*) \\ & \Leftrightarrow & [\psi]\precsim[\varphi] & \end{array}$$

For the bi-implication (*) we observe that $\psi \vdash \psi \wedge \varphi$ implies that $\psi \vdash \varphi$ by $(\wedge E_2)$. For the other direction by $(\wedge I)$ and (Id) we see that $\psi \vdash \varphi$ implies $\psi \vdash \psi \wedge \varphi$, and by (Id) and $(\wedge E_1)$ we also have $\psi \wedge \varphi \vdash \psi$.

Now, we deduce

$$\begin{array}{rcll}
\Gamma \models_{\mathbf{D}} \psi & \Leftrightarrow & \bigwedge_{\gamma\in\Gamma} [\![\gamma\ (\vec{p})]\!]_{\mathbf{D}} \leq_{U^n} [\![\psi\ (\vec{p})]\!]_{\mathbf{D}} & \text{for a proper context } \vec{p} = (p_1, \ldots, p_n) \\
& \Leftrightarrow & [\![\bigwedge_{\gamma\in\Gamma} \gamma\ (\vec{p})]\!]_{\mathbf{D}} \leq_{U^n} [\![\psi\ (\vec{p})]\!]_{\mathbf{D}} & \text{by def. of } [\![\ ]\!]_{\mathbf{D}} \\
& \Leftrightarrow & [\bigwedge_{\gamma\in\Gamma} \gamma] \leq_{U^n} [\psi] & \\
& \Leftrightarrow & [\bigwedge_{\gamma\in\Gamma} \gamma] \precsim [\psi] & \\
& \Leftrightarrow & \bigwedge_{\gamma\in\Gamma} \gamma \vdash \psi & \text{by def. of } \precsim \\
& \Rightarrow & \Gamma \vdash \psi &
\end{array}$$

For the last implication we have used the rule $(\vee E)$ for performing a 'cut' as follows: From $\bigwedge_{\gamma\in\Gamma} \gamma \vdash \psi$ we deduce $\Gamma, \bigwedge_{\gamma\in\Gamma} \gamma \vdash \psi$ by weakening with (W), and by repeated application of $(\wedge I)$ we can prove $\Gamma \vdash \bigwedge_{\gamma\in\Gamma} \gamma$ and hence we apply $(\vee E)$ with the premises $(\Gamma, \bigwedge_{\gamma\in\Gamma} \gamma \vdash \psi), (\Gamma, \bigwedge_{\gamma\in\Gamma} \gamma \vdash \psi), (\Gamma \vdash \bigwedge_{\gamma\in\Gamma} \gamma)$ and get $\Gamma \vdash \psi$.

Hence in particular if $\Gamma \not\vdash \psi$ then $\Gamma \not\models_{\mathbf{D}} \psi\ (\vec{p})$ from which the completeness follows. □

8.4 Monotone Transition Systems

We now turn our attention to a more restricted and intuitively appealing class of models, which we call *monotone transition systems*. They are variants of the Kripke models used as models for various modal and intuitionistic logics, including the modal μ-calculus. (Kripke models are essentially what we elsewhere have called labelled transition systems.) We will later describe a particular instance of monotone transition systems arising in the process algebraic world. But as pointed out by Plotkin and Stirling [72] one might also expect monotone transition systems to appear in many situations when using domain theory (our monotone transition systems are precisely their Kripke frames with 'Frame Condition 1').

Definition 8.6 A *monotone transition system* T is a triple $(S, \sqsubseteq, \dot{\rightarrow})$ where S is a set of states, $\sqsubseteq$ is a partial order on S and $\dot{\rightarrow} \subseteq S \times S$ is a transition relation between states. We require that $\sqsubseteq$ and $\dot{\rightarrow}$ fulfill the following condition

$$\forall s, s', t \in S.\ s \sqsubseteq s' \ \&\ s \overset{\cdot}{\to} t \quad \Rightarrow \quad \exists t' \in S.\ t \sqsubseteq t' \ \&\ s' \overset{\cdot}{\to} t'. \tag{8.1}$$

□

The partial order $\sqsubseteq$ should be thought of as representing an information ordering. The condition relating the partial order and the transition system can be viewed as a monotonicity requirement on the transition relation; in the degenerate case where $\overset{\cdot}{\to}$ is actually a total function, it is nothing else than the usual definition of monotonicity. Another interesting degenerate case is when $\overset{\cdot}{\to}$ is the identity relation on S, the monotone transition system is then a normal (transitive, reflexive, anti-symmetric) Kripke model as investigated by van Dalen [84] and Hughes and Cresswell [44]. When $\sqsubseteq$ is discrete we arrive at the normal transition systems.

As a diagram the monotonicity condition becomes:

$$\begin{tikzcd}
s' \arrow[r, dashed, "\cdot"] & t' \\
s \arrow[u, "\sqsubseteq"] \arrow[r, "\cdot"] & t \arrow[u, dashed, "\sqsubseteq"']
\end{tikzcd}$$

We will show that a monotone transition system T induces a μIK-category $\mathbf{T}$, thus provides models for the logic μIK. When restricting the interpretation to the fixed-point free fragment IK this interpretation will coincide with the satisfaction relation of Plotkin and Stirling [72]. First, however, we will show that T induces a complete lattice $\mathcal{T}$ given by the upper (upwards closed) subsets of S with respect to $\sqsubseteq$, which is known as the *Alexandrou topology* on $(S, \sqsubseteq)$, see for instance Johnstone [46]. The upper sets are going to be the properties of the underlying monotone transition systems.

We recall that a subset U of the partial order $(S, \sqsubseteq)$ is an *upper set* if for $u, v, u \in U, u \sqsubseteq v$ implies $v \in U$. We now have:

Proposition 8.1 *Given a partial order $(S, \sqsubseteq)$. Then the collection of upper* *subsets of S, $\mathcal{U}(S, \sqsubseteq)$ ordered by inclusion is a complete distributive lattice with meet given by intersection and join given by union.*

Proof: It is straightforward to verify that $\mathcal{U}(S, \sqsubseteq)$ is closed under arbitrary intersections and unions. $\square$

For the modalities we will need a little more structure. Define the two operations $\Diamond_T$ and $\Box_T$ on subsets of S as follows:

$$\begin{aligned}\Diamond_T U &= \{v \in S \mid \exists u \in U. v \dot{\rightarrow} u\}\\ \Box_T U &= \{v \in S \mid \forall v' \in S, \forall w \in S. v \sqsubseteq v' \ \&\ v' \dot{\rightarrow} w \Rightarrow w \in U\}\end{aligned}$$

Upper sets are closed under these operations.

Proposition 8.2 *If U is an upper set, so is $\Diamond_T U$ and $\Box_T U$.*

Proof: Straightforward, for $\Diamond_T$ the monotonicity condition (8.1) is needed. $\square$

Recall, that on a complete lattice $(D, \leq)$ a monotonic function $f : D \rightarrow D$ has by Tarski's theorem [82] minimum and maximum fixed-points given by

$$\mu f = \bigwedge\{x \in D \mid fx \leq x\}$$

and

$$\nu f = \bigvee\{x \in D \mid x \leq fx\}.$$

This means in particular than on the lattice $\mathcal{U}(S, \sqsubseteq)$ we have minimum and maximum fixed-points of monotonic functions. We now have enough structure to give the category $\mathbf{T}$ as a particular subcategory of the category of sets. Let $U = \mathcal{U}(S, \sqsubseteq)$.

Objects are finite Cartesian products of U.

Morphisms $U^n \rightarrow U^m$ are functions from U^n to U^m, i.e. m-tuples of functions from U^n to U.

Composition and identity is as usual.

Now, we make U into an internal distributive lattice by the following definition of operators.

$$\begin{array}{rcl}\bot &=& \lambda x \in \mathbf{1}.\emptyset \\ \top &=& \lambda x \in \mathbf{1}.S \\ \wedge &=& \lambda\langle x, y\rangle \in U \times U.x \cap y \\ \vee &=& \lambda\langle x, y\rangle \in U \times U.x \cup y\end{array}$$

It is trivial to verify that all the diagrams commute (recall that they represent an equational presentation of a distributive lattice). Moreover, U has K-modalities $\Box_T$,$\Diamond_T$ and fixed-point operators by the following definitions.

$$\begin{array}{rcl}\mu_{U^n}(f) &=& \lambda\langle x_1, \dots, x_n\rangle \in U^n\ .\mu(\lambda y \in U.f(x_1, \dots, x_n, y)) \\ \nu_{U^n}(f) &=& \lambda\langle x_1, \dots, x_n\rangle \in U^n\ .\nu(\lambda y \in U.f(x_1, \dots, x_n, y))\end{array}$$

It is straightforward to verify that $\Box_T$ and $\Diamond_T$ satisfy the equations (x) to (xiv). Naturality of μ_{U^n} and ν_{U^n} and the fixed-point properties are just as easy. Hence $\mathbf{T}$ is a μIK-category.

The usual notion of satisfaction $s \models \psi$ between a state s and a closed assertion ψ can now be obtained as

$$s \models \psi \Leftrightarrow_{\text{def}} \uparrow s \subseteq [\![\psi]\!]_{\mathbf{T}}(*),$$

where $*$ denote the single element of the terminal object $\mathbf{1}$ and $\uparrow s$ is the upwards closure of s with respect to $\sqsubseteq$: $\uparrow s = \{q \in S \mid s \sqsubseteq q\}$. It is not hard to prove that this definition coincides with the deinition of Plotkin ind Stirling [72] (using 'frame condition 1'). We just write out some of the consequences for IK:

$$\begin{array}{rcl}s \models \Diamond\psi &\Leftrightarrow& \exists q.s \dot{\rightarrow} q \ \&\ q \models \psi \\ s \models \Box\psi &\Leftrightarrow& \forall s', q.s \sqsubseteq s' \ \&\ s' \dot{\rightarrow} q \Rightarrow q \models \psi\end{array}$$

Now, from the general result of soundness for μIK-categories, it follows that this interpretation is sound. Whether it is complete is a much more difficult question. For the classical version of the modal μ-calculus with classical Kripke models/transition systems this is a problem which has been open since the partial solution given by Kozen in [49]. Thus we might expect it to be even harder to give a completeness result for the intuitionistic version.

Stirling has shown that a finite axiomatization exists for the fixed-point free fragment of μIK [78] (with models almost identical to monotone transition systems), so the problem is whether the axioms for the fixed-points are strong enough to achieve completeness.

$$\begin{tikzcd}
\mu IK \arrow[rr, bend left, "\mathbf{D}"] \arrow[drr, bend right, "universal\ model?"'] & & \mu IK\text{-}categories \arrow[d, bend left, "\mathcal{M}"] \\
& & \mathcal{MTS} \arrow[u, bend left, "`upper\ sets'"] \\
logic & & models
\end{tikzcd}$$

Figure 8.4: One way of attacking completeness: Search for a map $\mathcal{M}$ 'extracting' monotone transition systems from μIK-categories, or at least a universal $\mathcal{MTS}$-model from the universal model $\mathbf{D}$.

Figure 8.4 indicates how completeness can be solved by finding a proper way of generating monotone transition systems from (at least some, including $\mathcal{U}$) μIK-categories; having such a validity-preserving construction would immediately yield a universal falsifying model for μIK. An idea of how this could be done can be found by studying the fixed-point free fragment, i.e. the intuitionistic modal logic IK. The traditional way of constructing a universal model for intuitionistic logic (see Fitting [39]) is like the construction of a Lindenbaum algebra for showing completeness of propositional logic; states of the model are particularly well-behaved (infinite) sets of assertions and the reachability relation is set inclusion. This can be extended to intuitionistic modal logic by defining a transition relation on the states by

$$\Gamma \to \Delta \Leftrightarrow_{\text{def}} \begin{array}{l} \forall\phi.\ \Box\phi \in \Gamma \Rightarrow \phi \in \Delta \\ \forall\phi.\ \phi \in \Delta \Rightarrow \Diamond\phi \in \Gamma \end{array}$$

and taking the information ordering to be set inclusion, as done by Stirling in [78].

However, preliminary studies seem to indicate that using this approach on IK involves work in constructing a universal model from the categorical universal IK-model that is no simpler than going directly from the logic to a monotone transition system. The construction sketched above can be reused, but no great benefits seem to be easily obtainable.

A big obstacle in getting a completeness proof is the failure of compactness for the full modal μ-calculus.[1]

8.5 Adding Implication

When adding implication to the logic, a problem due to a mismatch between syntactic and semantical monotonicity arise. As implication $\rightarrow$ is contravariant in its first argument with respect to the order $\leq$, minimum and maximum fixed-points must be formed subject to the syntactic monotonicity criterion that the variable being bound must only occur positively in the assertion. This causes difficulties in the construction of the categorical universal model as we need a family of operators μ_{U^n} with the property that for *all* monotonic *morphisms*, $[\psi] : U^n \times U \rightarrow U$, we get a morphism $\mu_{U^n}([\psi]) : U^n \rightarrow U$. We define this by taking $\mu_{U^n}([\psi]) = [\mu p_{n+1}.\psi]$; however, in order for this to be well-defined, ψ must be syntactically monotone in p_{n+1}, which is not necessarily the case even though the morphism $[\psi]$ is semantically monotone. One way around this problem is to show that there always exists a provably equal assertion ψ^* which is indeed syntactically monotone and use this in the definition. This is a rather unpleasant approach as it requires a proof which will be very dependent on the actual rules – and it is not even obvious how to actually prove it.

A more thorough treatment should take proper account of the monotonicity of assertions in the categorical models, by having an object U^{op} 'with the opposite ordering' and letting $\rightarrow$ be a morphism $\rightarrow: U^{op} \times U \rightarrow U$. Although such an approach solves the problems about constructing the operators μ_X in the completeness proof, it raises other difficulties. How is one going to define satisfaction and entailment such that the models are sound?

[1]In the standard calculus an example showing non-compactness (suggested to me by Glynn Winskel) is: The infinite set of assertions $\{\mu X.Q \vee \langle a\rangle X, \neg Q, [a]\neg Q, [a][a]\neg Q, \ldots, [a]\ldots[a]\neg Q, \ldots\}$ entails false, but no finite subset of the assertions do so.

To interpret assertions we have to annotate the positive and negative occurrences of free variables ψ^+ and get a morphism $[\psi^+] : (U^{op})^m \times U^n \to U$. However, this makes vaiables occurring in negative and positive positions different and how do we reflect that they are 'the same'? For instance, how can we ensure that the assertion $p \to p$ gives rise to the same morphism as $\top$, although $p \to p$ has both a positive and a negative occurrence of p considered to be different, but still $p \to q$ must be different from $\top$?

8.6 Conclusion

The way in which we introduced the categorical models follows a very general pattern known from work on categorical logic, that could be used whenever equivalent sequent and equational presentations of a logic is present. The fixed-point operators called for some special constructions, which required *ad hoc* ingenuity, but the task seemed very straightforward. As mentioned in the introduction everything will carry through to the classical situation by introducing sequent rules for negation, and rules making the two modalities and the two fixed-points interdefinable. Instead of categories with internal distributive lattice objects, we would be involved with categories with internal Boolean algebra objects. Everything should carry through.

We have introduced a new logic, an intuitionistic (implication-free) version of the modal μ-calculus. Apart from the problem of completeness, this automatically raise a lot of questions like: Is validity decidable, for the μIK-categories? For the monotone transition systems ? Does the logic have the 'finite model property'? With the satisfaction relation defined in section 8.4, is the model checking problem decidable?

Chapter 9

Conclusion and Further Work

In the preceding chapters various aspects of the verification of temporal properties of concurrent systems have been addressed. In this final chapter we will summarize the achievements and point to remaining unsolved problems and possible further work.

9.1 Compositionality

The first contribution of the thesis was the compositional method for deciding satisfaction in chapter 3. The method, building upon earlier work by Winskel[89, 94], was based on the notion of a *reduction*. The main difference from Winskel's previous work is the completely new reductions for recursion and product and the introduction of the propositional logic $\mathcal{L}$ with correctness assertions as atoms, allowing for reductions quite different from and simpler than the previous attempts.

The reduction for product incidentally turned out to be very similar to what Larsen and Xinxin have achieved with their 'operational contexts' [57], but we avoid the complication of going through the notions of operational contexts and we also provide a syntactic reduction, showing how the reduction for product fits together with the other reductions. The adaptation of (some of) the reductions to the extended calculus, showed how it might be expected that these reductions can be an integral part of a general framework for reasoning about concurrent systems using a powerful specification language. (There should be no difficulties in extending the remaining reductions as well.)

The compositional technique also had some unexpected applications in verifying equivalences and preorders by providing algorithms and characteristic formulas as investigated in chapter 4.

The reductions can be seen as running a proof system backwards, for instance the reduction for sum express that $\models (p+g : \langle a \rangle A) \leftrightarrow (p : \langle a \rangle A) \vee (q : \langle a \rangle A)$, ie. to prove $\models p + q : \langle a \rangle A$ it is sufficient (and necessary) to prove either $\models p : \langle a \rangle A$ or $\models q : \langle a \rangle A$. For this simple example it is not hard to supply the two forwards rules

$$\frac{\vdash p : \langle a \rangle A}{\vdash p + q : \langle a \rangle A} \qquad \frac{\vdash q : \langle a \rangle A}{\vdash p + q : \langle a \rangle A},$$

but for some of the other operators, in particular the parallel operator and the recursion operator, things are more complicated. Also the fixed-point assertions crucial for the expressiveness of the logic cause severe difficulties. An indication of how such a *compositional proof system* could look can be found in the work of Stirling [77, 76] and Winskel [90, 93] (which contain versions of the above rules). They do not address the problem of the recursive operator and they solve the problems with the parallel operator in different ways. Stirling suggests using a satisfaction predicate relativized with respect to one part of a parallel composition. However, it requires guessing proper assertions for these components in order to be successful. Winskel factorizes all fixed-point-free assertions into finite disjunctions of products of assertions and shows how it is always possible to decompose such assertions into – rather large – assertions for the individual components of a product of transition systems.

For the fixed-point assertions they put a bound on the number of needed unfoldings determined by the size of the process in question thereby solving the problem by removing the fixed-point operators altogether. This is not a very satisfactory solution because it requires knowledge about the size of the process under consideration, and when reasoning compositionally this should be of no concern to the reasoning – after all part of the process might still be missing.

In contrast to this, it is central to the reductions presented in this thesis that the fixed-points are kept and the main contribution of the reductions is actually to show that this can be done for all operators. The decompositional nature of the reductions can be 'turned around' and used to generate – a rather unwieldy – forwards system for assertions with fixed-points. However, due to the complexity of the rules, it seems to be of little use and we have not yet succeeded in getting a satisfactory proof system in this way.

9.2 Model-Checking Algorithms

Another indirect application of the compositional method was in transforming the problem of deciding satisfaction into a problem of determining the value of a boolean fixed-point expression. Based on this idea we presented global model checkers improving on the bounds of Emerson and Lei [35]. (A recent paper by Cleaveland, Dreimüller, and Steffen [24] presents another global algorithm for the full calculus with a similar complexity bound.) And we presented local model checkers improving on the local algorithms of Larsen [53], Cleaveland [23] (based on the proof system of Stirling and Walker [80]) and Winskel [92]. (Larsen [54] has recently improved on his local algorithm for non-alternating fixed-points giving a polynomial-time algorithm, but it is still not as efficient as the algorithm of chapter 5.)

These algorithms all required (for the local algorithm only in the worst-case) the computation of the complete transition system for a process term; something that might generate transition systems exponentially bigger than the original process term – a problem sometimes referred to as the 'state explosion problem.' Now, are the presented algorithms then 'good enough'? A partial answer can be found by analyzing the complexity of performing model checking on static processes.

Using the observations of chapter 4 it is quite immediate that model checking of the extended calculus (for the subsets that are decidable, cf. the discussions in section 2.7 and section 5.10) is at least as difficult as the checking of bisimulation equivalence and the checking of any of the preorders in that chapter. Hence, using the result of Rabinovich [73] that deciding bisimulation equivalence for what we refer to as static processes is PSPACE-hard, it follows that model checking of the extended calculus is PSPACE-hard for static processes.

A more direct proof strengthening this result to the standard calculus can be given based on Rabinovich's construction.

Theorem 9.1 *Deciding*

$$\models p : A$$

for a static process p

$$p = op(p_1, \ldots, p_n),$$

and a closed assertion A in the standard calculus of alternation depth one is PSPACE-hard.

Proof: (Sketch) Take A to be the assertion $EEven(\tau, \langle ✓ \rangle T) = \mu X.\langle \tau \rangle X \vee \langle ✓ \rangle T$. Let M be any Turing machine for a PSPACE-hard problem with space bound s and time bound t. Define for each input x a process $p(x)$ which simulates the Turing machine, and terminates by performing the action ✓ if the input is accepted. This can be done as described by Rabinovich [73] by taking for an input of size n, $s(n)$ cells $p_1, \ldots, p_n$ each simulating one location on the tape, and defining a process q simulating the finite state control of the Turing machine. Then the process

$$(q \mid p'_1 \mid \cdots \mid p'_n \mid p_{n+1} \mid \cdots \mid p_{s(n)}) \restriction L$$

where $L = \{\tau, ✓\}, p'_i$ initially contains the i'th bit of the input x, and p_i contains a blank symbol, will eventually perform a ✓-action if and only if the Turing machine M accepts x. This translation is easily performed within polynomial time. □

The algorithm for deciding satisfaction for static processes we get by simply computing the induced transition system and using for example the global algorithm, gives for any fixed alternation depth an algorithm in EXPTIME. Hence we have a gap between the lower bound of PSPACE supplied by the theorem and the upper bound of EXPTIME. However, recent preliminary results on checking bisimulation equivalence giving a lower bound of EXPTIME implies that model checking of static processes in even very restricted subsets of the extended calculus is EXPTIME-hard and probably (depending on the construction used in the proof) also implies that for the standard calculus the problem is EXPTIME-hard. (Cf. Rabinovich [73, p.706] which conjectures that bisimulation checking is EXPTIME-hard and refers to private communications with Stockmeyer proving this fact.) Hence, if this is correct the algorithm that first computes the induced transition system and then applies the global algorithm is *optimal for static processes and assertions of alternation depth one*.

If we allow more than static processes we can go even further. Let CCS_{fin} be the set of finite CCS-processes (recall, from section 2.2 that this set is not recursive, only recursively enumerable). Using a construction similar in spirit to Rabinovich's, but using two stacks as when Milner shows Turing strength of CCS [59], we can actually that the satisfaction problem for CCS_{fin} is *arbitrarily hard* more precisely, for any time (or space) bound t, there exists an assertion and infinitely many processes for which it requires at least time (or space) t to decide the satisfaction problem.

This is proven by simply constructing for a Turing machine M and input x in linear time a process term $p(M, x)$ that when M is total induces a finite transition system and hence belongs to CCS_{fin}. This is very straightforward using Milner's idea; we simply construct two stacks simulating what is to the left and respectively to the right of the head of the Turing machine, and encode the finite control as a process. Initially the stack to the left will contain the input x and the stack to the right is empty. (See Milner [59] section 6.1.) We again indicate that we have reached an accepting state by performing the action ✓, hence the only possible actions of the encoding $p(M, x)$ will be τ and ✓. This translation takes polynomial time and linear space.

Theorem 9.2 *For any time bound t at least exponential (and for any space bound t at least polynomial), there exists a closed assertion A in the standard calculus of alternation depth one and a Turing machine M such that deciding*

$$\models p(M, x) : A$$

for inputs x of length n takes time (or space) at least $t(n)$ for infinitely many x. Hence, deciding satisfaction for processes in CCS_{fin} and assertions of alternation depth one in the standard calculus takes time (or space) at least proportional to t.[1]

Proof: (Sketch) This is a direct consequence of the space and time hierarchy theorems stating, intuitively, that there exists problems with arbitrarily complex time (and space) requirements. (See for example Hopcroft and

[1]The time and space hierarchy theorems used in the proof require t to satisfy a technical criterion of being time (respectively space) constructible, which we will not describe in detail; we just use the term of being 'a time (or space) bound t' to capture this fact.

Ullman [43].) Hence, for any t these theorems supply a Turing machine which requires at least time (or space) t to determine whether inputs should be accepted or not. Taking again A to be the assertion $EEuen(\tau\langle \checkmark \rangle T)$, the construction of $p(M,x)$ in polynomial time and linear space proves the theorem. $\square$

9.3 Other Issues

State explosion. One way of attacking the problem of state explosion is offered by the reduction for product. Consider the situation where we have a static process $p = p_1 \mid \cdots \mid p_n$ and we want to decide whether p satisfies an assertion A. We could use the reduction for product (or rather the reduction for $|$) once to get

$$p_1 \mid \ldots \mid p_{n-1} : \mathrm{red}_{|p_n} A.$$

Repeating this $n-1$ times we end up with the answer. But in each step we are increasing the assertion considerably – by a factor of the size of the process we are dividing out. However, if these intermediate assertions could somehow be kept small, by for instance some assertion-minimizing algorithm, this would provide a means for avoiding the state-explosion problem. At present no non-trivial algorithm for minimizing modal μ-calculus assertions exists, but perhaps one could be constructed, at least for the non-alternating part, using rewrite techniques from one of the existing finite axiomatizations of the fixed-point free fragment (also known as Hennessy-Milner logic, or the minimal modal logic K).

It is easy to construct simple examples where the above approach even with a naive minimization algorithm gives good results.[2] But whether it can provide a good heuristic depends on whether minimization algorithms that work well on non-trivial subsets can be found.

The extended calculus. The extended calculus was introduced mainly for convenience of expressing properties, but the interplay between the

[2]Here is on e. If the assertion is: 'There exists an infinite sequences of a-actions', and the process p_n can perform these by itself, the first division will result in an assertion which is quite easily seen to be true, irrespective of what the remaining processes are.

fixed-point operators and the predicate logic on actions seems to have features that makes it interesting enough to deserve a study on its own. Such a study was initiated in section 2.7 where it was discussed briefly when model checking is decidable and how assertions in the extended calculus under certain circumstances could be translated to assertions in the standard calculus, but many questions concerning for example the expressive power of the logic and the hardness of the model checking problems have been left open.

The elaborate logical structure on actions given by the action predicates and quantifiers could also turn out to be useful in the context of processes with value passing where synchronizations also include passing of values and the action structure is more complex.

The local algorithms. A feature of the local algorithms that have not been investigated here is their ability to offer *explanations* (an observation used in the TAV system [55].) If for instance we use the local algorithm for maximum fixed-points in determining whether two states p and q are bisimilar using the encoding as a maximum fixed-point $\mathcal{B}$ from chapter 4 and we get the result that this is not the case; then in the boolean fixed-point expression we can find a tree of boolean variables with value 0 and a root corresponding to the satisfaction problem $p \times q : \mathcal{B}$, such that any variable with right-hand side a conjunction has one successor (and only one as the local algorithm stops when the first is found) in the tree, and if the variable has a disjunction on the right-hand side all the successors of that variable are in the tree. No cycles will be present.[3] As each boolean variable corresponds to a pair of states $p' \times q'$ and subassertion A of $\mathcal{B}$, this information can be used to explain why p and q are not bisimilar.

Similarly, if a process turns out not to satisfy a minimum fixed-point assertion, a subgraph (now possibly with cyclic dependencies) can be extracted that explains why.

The general local fixed-point finding algorithm. The ideas of sharing and chasing dependencies that made the local model checker efficient turned out to be of more general applicability and in chapter 6 a general local algorithm for computing fixed-points in finite cpo's and lattices

[3]Such a cycle could never get assigned value 0 to all its variables by the algorithm!

where given. It seems to have applications in for instance abstract interpretations, where other local approaches as Pending Analysis has already turned out to be quite successful. It is of major interest and an important area for future work to find out whether the local algorithm we presented can be used in giving an efficient algorithm for higher-order abstract interpretation, a problem which currently, despite several attempts, has no good and efficient solution.

The infinite-state method. The infinite-state method of chapter 7, which we consider as being a recasting of the method of Bradfield and Stirling [17], could be implemented as a tool along the lines of Bradfield [16]. As Bradfield points out the major difficulty is in supplying a useful notation for the infinite-state processes. We believe that for bounded processes the notation suggested in chapter 7 provides a promising attempt.

Fundamental theoretical problems. The major open problem with the standard calculus, is the problem of finding a finite axiomatization; a problem which was solved by Kozen [49] for a restricted 'aconjunctive' calculus, but which for the general calculus remains open. Kozen has given an axiomatization with one infinitary rule for the minimum fixed-point operator in [50]. We offered, modestly, a new way of attack by providing a class of categorical models for which completeness is easily shown. But the problem of restricting these to Kripke-like or transition system-like models does so fa not seem to have any simple solution.

The logic is known to be decidable, a result due to Kozen and Parikh [51] and improved upon by Emerson and Jutla [33] which give the best known, exponential time algorithm. Kozen [50] showed that it has the finite model property, i.e. if an assertion is satisfiable it is satisfiable in a finite model.

Another open problem concerns the hierarchy of sublogics that arise from bounding the number of alternations of minimum and maximum fixed-points. Niwiński [63, 64] has shown that for a term-interpretation of the μ-calculus with the operators – including the modal operators – being simple term-constructors, the hierarchy is indeed strict. But this does not imply that the hierarchy is strict for the more traditional Kripke models; it merely says that it cannot be ruled out that the hierarchy is indeed strict.

Bibliography

[1] *Proceedings of the 4th Workshop on Computer Aided Verification, June 29 - July 1, 1992, Montreal, Quebec, Canada,* 1992. Forthcoming.

[2] P. Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic.* North-Holland, 1983.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[4] Henrik Reif Andersen. Local computation of alternating fixed-points. Technical Report No. 260, Computer Laboratory, University of Cambridge, June 1992.

[5] Henrik Reif Andersen. Local computation of simultaneous fixed-points. Technical Report PB-420, Computer Science Department, Aarhus University, October 1992.

[6] Henrik Reif Andersen. Model checking and boolean graphs (extended abstract). In B. Krieg-Brückner, editor, *Proceedings of 4'th European Symposium on Programming, ESOP'92, Rennes, France,* volume 582 of *LNCS*. Springer-Verlag, 1992.

[7] Henrik Reif Andersen and Glynn Winskel. Compositional checking of satisfaction. *Formal Methods In System Design,* 1(4), December 1992.

[8] Henrik Reif Andersen and Glynn Winskel. Compositional checking of satisfaction (extended abstract). In Larsen and Skou [56].

[9] André Arnold and Paul Crubille. A linear algorithm to solve fixed-point equations on transitions systems. *Information Processing Letters,* 29:57–66, 1988.

[10] G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors. *Proceedings of ICALP,* volume 372 of *LNCS,* 1989.

[11] J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR '90,* volume 458 of *LNCS.* Springer-Verlag, 1990.

[12] H. Bekič. Definable operations in general algebras, and the theory of automata and flow charts. In C. B. Jones, editor, *Hans Bekič: Programming Languages and Their Definition,* volume 177, pages 30–55. Springer-Verlag, 1984.

[13] Bard Bloom. *Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages.* PhD thesis, Massachusetts Institute of Technology, August 1989.

[14] Bard Bloom and Robert Paige. Computing ready simulations efficiently. Draft, July 1992.

[15] Julian C. Bradfield. *Verifying Temporal Properties of Systems with Applications to Petri Nets.* PhD thesis, Laboratory for Foundations of Computer Science, University of Edinburgh, July 1991.

[16] Julian C. Bradfield. A proof assistant for symbolic model-checking. Technical Report ECS-LFCS-92-199, Laboratory for Foundations of Computer Science, University of Edinburgh, March 1992.

[17] Julian C. Bradfield and Colin P. Stirling. Verifying temporal properties of processes. In Baeten and Klop [11], pages 115–125.

[18] Glenn Bruns. A case study in safety-critical design. Technical Report ECS-LFCS-92-239, Laboratory for Foundations of Computer Science, University of Edinburgh, September 1992.

[19] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science,* pages 428–439. IEEE Computer Society Press, 1990.

[20] Jiazhen Cai and Robert Paige. Program derivation by fixed point computation. *Theoretical Computer Science,* 11:197–261, 1989.

[21] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981,* volume 131 of *Lecture Notes in Computer Science,* pages 52–71. Springer-Verlag, 1981.

[22] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science,* pages 353–362, Asilomar Conference Center, Pacific Grove, California, June 5–8 1989. IEEE Computer Society Press.

[23] Rance Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica,* 27:725–747, 1990.

[24] Rance Cleaveland, Marion Dreimüller, and Bernhard Steffen. Faster model checking for the modal mu-calculus. In *Proceedings of the 4th Workshop on Computer Aided Verification, June 29 - July 1, 1992, Montreal, Quebec, Canada* [1], pages 383–394. Forthcoming.

[25] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The Concurrency Work-bench: A semantics based tool for the verification of concurrent systems. Technical Report ECS-LFCS-89-83, Laboratory for Foundations of Computer Science, Uni. of Edinburgh, August 1989.

[26] Rance Cleaveland and Bernhard Steffen. Computing behavioural relations, logically. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Proceedings of ICALP,* volume 510 of *LNCS,* pages 127–138. Springer-Verlag, July 1991.

[27] Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In Larsen and Skou [56].

[28] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.

[29] Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of recursive procedures. In Erich J. Neuhold, editor, *Formal Description of Programming Concepts,* pages 237–277. North-Holland, 1978. Proceedings of the IFIP Working Conference on Formal Description of Programming Concepts, St. Andrews, N.B., Canada, August 1–5, 1977.

[30] Mads Dam. Translating CTL* into the modal μ-calculus. Technical Report ECS-LFCS-90-123, Laboratory for Foundations of Computer Science, University of Edinburgh, November 1990.

[31] Mads Dam. CTL* and ECTL* as fragments of the modal μ-calculus. In Raoult [74], pages 145–164.

[32] Alan J. Dix. Finding fixed points in non-trivial domains: Proofs of pending analysis and related algorithms. Technical Report YCS 107, University of York, Department of Computer Science, 1988.

[33] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *IEEE Foundations of Computer Science,* 29:328–337, 1988.

[34] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J.W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming. ICALP'80,* volume 85 of *Lecture Notes in Computer Science,* pages 169–181. Springer-Verlag, 1980.

[35] E. Allen Emerson and Chin-Luang Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science, Proceedings,* pages 267–278. IEEE, 1986.

[36] E.A. Emerson and J. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time. *Journal of the ACM,* 33, 1986.

[37] Klaus Estenfeld, Hans-Albert Schneider, Dirk Taubner, and Erik Tidén. Computer aided verification of parallel processes. In A.

Pfitzmann and E. Raubold, editors, *VIS '91 Verlässliche Informationssysteme,* volume 271 of *Informatik Fachberichte,* pages 208–226, Darmstadt, 1991. Springer.

[38] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *JCSS,* 18:194–211, 1979.

[39] Melvin Fitting. *Intuitionistic Logic. Model Theory and Forcing.* North-Holland, 1969.

[40] N. De Francesco and P. Inverardi. A semantic driven method to check the finiteness of CCS processes. In Larsen and Skou [56].

[41] Carl A. Gunter. *Semantics of Progamming Languages: Structures and Techniques.* MIT Press, 1992.

[42] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[43] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[44] G. Hughes and M. Cresswell. *An Introduction to Modal Logic.* Methuen, 1968.

[45] IEEE Computer Society. *Proceedings, Third Annual Symposium on Logic in Computer Science,* Edinburgh, Scotland, July 5–8 1988.

[46] Peter T. Johnstone. *Stone spaces.* Cambridge studies in advanced mathematics 3. Cambridge University Press, 1982.

[47] Neil D. Jones and Alan Mycroft. Data flow analysis of applicative programs using minimal function graphs: Abridged version. In *Proceedings of 13th Annual ACM Symp. on Principles of Programming Languages,* 1986.

[48] Gary A. Kildall. A unified approach to global program optimization. In *Proceedings of POPL'73,* pages 194–206. ACM, 1973.

[49] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science,* 27, 1983.

[50] Dexter Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica,* 47:233–242, September 1988.

[51] Dexter Kozen and Rohit Parikh. A decision procedure for the propositional mu-calculus. In E. Clarke and D. Kozen, editors, *Logics of Programs, Proceedings,* volume 164 of *LNCS,* 1983.

[52] Leslie Lamport. "Sometimes" is sometimes "not never". In *Proceedings of 7th Annual ACM Symp. on Principles of Programming Languages,* 1980.

[53] Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In M. Dauchet and M. Nivat, editors, *Proceedings of CAAP, Nancy, Franch,* volume 299 of *Lecture Notes in Computer Science,* pages 215–230, March 1988.

[54] Kim G. Larsen. Efficient local correctness checking. In *Proceedings of the 4th Workshop on Computer Aided Verification, June 29 - July 1, 1992, Montreal, Quebec, Canada* [1]. Forthcoming.

[55] Kim G. Larsen, J.C. Godskesen, and M. Zeeberg. TAV–Tools for Automatic Verification. Technical Report R 89-19, Aalborg Universitetscenter, 1989.

[56] Kim G. Larsen and Arne Skou, editors. *Proceedings of the 3rd Workshop on Computer Aided Verification, July 1991, Aalborg,* volume 575 of *LNCS.* Springer-Verlag, 1992.

[57] Kim G. Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. In M.S. Paterson, editor, *Proceedings of ICALP,* volume 443 of *LNCS,* pages 526–539. Springer-Verlag, 1990.

[58] Robin Milner. *A Calculus of Communicuting Systems,* volume 92 of *LNCS.* Springer-Verlag, 1980.

[59] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[60] Alan Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs.* PhD thesis, Laboratory for

Foundations of Computer Science, University of Edinburgh, December 1981.

[61] Flemming Nielson and Hanne Riis Nielson. Finiteness conditions for fixed point iteration. Manuscript. To appear, November 1991.

[62] Hanne Riis Nielson and Flemming Nielson. Bounded fixed point iteration. In *Proceedings of the 18'th Annual Symposium on Principles of Programming Languages, POPL,* 1991. Also as DAIMI PB-359, Aarhus University, July 1991.

[63] Damian Niwiński. On fixed-point clones. In Laurent Kott, editor, *Proceedings of ICALP,* volume 226 of *LNCS,* pages 464–473, 1986.

[64] Damian Niwiński. Fixed points vs. infinite generation. In *Proceedings, Third Annual Symposium in Logic in Computer Science* [45], pages 402–409.

[65] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journ. Comput,* 16(3):973–989, December 1987.

[66] Jens Palsberg and Michael I. Schwartzbach. Object-oriented type inference. In *Proc. OOPSLA '91, ACM SIGPLAN Sixth Annual Conference on Object-Oriented Programming Systems, Languages and Applications,* pages 146–161, Phoenix, Arizona, October 1991.

[67] Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference. Technical Report PB-389, Computer Science Department, Aarhus University, 1992. Submitted for publication.

[68] David Park. Fixpoint induction and proofs of program properties. *Machine Intelligence,* 5, 1969.

[69] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Proceedings of Theoretical Computer Science, 5th GI-Conference, Karlsruhe, March 23-25, 1981,* volume 104 of *Lecture Notes in Computer Science,* pages 167–183. Springer-Verlag, 1981.

[70] Andrew M. Pitts. On an interpretation of second order quantification in first order intuitionistic propositional logic. *Journal of Symbolic Logic.* To appear.

[71] Gordon Plotkin. Unpublished notes on domain theory ('The Pisa Notes').

[72] Gordon Plotkin and Colin Stirling. A framework for intuitionistic modal logics. In J. Y. Halpern, editor, *Theoretical Aspects of Reasoning About Knowledge,* 1986.

[73] Alexander Rabinovich. Checking equivalences between concurrent systems of finite agents (extended abstract). In *Proceedings of ICALP,* volume 623 of *LNCS,* pages 696–707. Springer-Verlag, 1992.

[74] J.-C. Raoult, editor. *Proceedings of 17'th Colloquium on Trees in Algebra and Programming, CAAP'92, Rennes, France,* volume 581 of *LNCS.* Springer-Verlag, 1992.

[75] Bernhard Steffen. Characteristic formulae for CCS with divergence. In Ausiello et al. [10], pages 723–733.

[76] Colin Stirling. A complete compositional modal proof system for a subset of CCS. volume 194 of *Lecture Notes in Computer Science,* pages 475–486. Springer-Verlag, 1985.

[77] Colin Stirling. A complete modal proof system for a subset of SCCS. volume 185 of *Lecture Notes in Computer Science,* pages 253–266. Springer-Verlag, 1985.

[78] Colin Stirling. Modal logits for communicating systems. Technical Report CSR-193-85, Department of Computer Science, University of Edinburgh, October 1985.

[79] Colin Stirling. Modal and Temporal Logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science.* Oxford University Press, 1991.

[80] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. In J. Díaz and F. Orejas, editors, *Proceedings of TAPSCFT, Barcelona, Spain,* volume 351 of *Lecture Notes in Computer Science,* pages 369–383, March 1989.

[81] R. Tarjan. Depth-first search and linear graph algorihtms. *SIAM J. Comput.,* 2(1), June 1972.

[82] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics,* 5:285–309, 1955.

[83] Dirk Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets,* volume 369 of *Lecture Notes in Computer Science.* Springer-Verlag, 1989.

[84] Dirk van Dalen. *Logic and Structure.* Springer-Verlag, second edition, 1983.

[85] Bart Vergauwen and Johan Lewi. A linear algorithm for solving fixed-point equations on transition systems. In Raoult [74], pages 322–341.

[86] Philip Wadler. Strictness analysis on non-flat domains. In Samson Abramsky and Chris Hankin, editors, *Abstract interpretation of declarative languages,* chapter 12. Ellis Horwood, 1987.

[87] David J. Walker. Automated analysis of mutual exclusion algorithms using CCS. Technical Report ECS-LFCS-89-91, Laboratory for Foundations of Computer Science, University of Edinburgh, August 1989.

[88] Glynn Winskel. Synchronisation trees. *Theoretical Computer Science,* 34:33, 1984.

[89] Glynn Winskel. On the composition and decomposition of assertions. Technical Report TR-59, Computer Laboratory, University of Cambridge, 1985.

[90] Glynn Winskel. A complete proof system for SCCS with modal assertions. *Fundamenta Informaticae,* IX:401–420, 1986.

[91] Glynn Winskel. A category of labelled Petri Nets and compositional proof system (extended abstract). In *Proceedings, Third Annual Symposium on Logic in Computer Science* [45], pages 142–153.

[92] Glynn Winskel. A note on model checking the modal ν-calculus. In Ausiello et al. [10], pages 761–772.

[93] Glynn Winskel. A compositional proof system on a category of labelled transition systems. *Information and Computation,* 87, 1990.

[94] Glynn Winskel. On the compositional checking of validity. In Baeten and Klop [11], pages 481–501.

[95] Glynn Winskel. *The Formal Semantics of Programming Languages.* MIT Press, 1993.

[96] Liu Xinxin. *Specification and Decomposition in Concurrency.* PhD thesis, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, Denmark, April 1992.

[97] Jonathan Young and Paul Hudak. Finding fixpoints on function spaces. Technical Report YALEU/DCS/RR-505, Yale University, Department of Computer Science, December 1986.

Appendix A

Proofs of Theorems of Chapter 3

This section contains the proofs of all major results in chapter 3.

A.1 Proof of Rooting Lemma

Rooting lemma, lemma 3.1 *Given a pointed transition system,* $T = (S_T, i, L, \rightarrow_T)$*, where* S_T *is countable and with the rooting* $\underline{T}$*. Let* $r : \mathcal{P}(S_T) \rightarrow \mathcal{P}(S_T \cup \{\underline{i}\})$ *be the map on properties that take the initial state of* T *to the two copies of it in* $\underline{T}$ *and take all other states to their obvious counterparts. Assume* A *is a pure assertion. Let* $\rho : AssnVar \rightarrow \mathcal{P}(S_T)$ *be an environment of assertions which since* A *is pure, respects the types of* A*. Then*

$$r([\![A]\!]_T \rho) = [\![A]\!]_{\underline{T}}(r \circ \rho).$$

□

Proof: The proof is by structural induction on A.

$A \equiv X$. By the definition of $[\![X]\!]_T\ \phi$ we immediately get:

$$r([\![X]\!]_T\ \phi) = r(\phi(X)) = r \circ \phi(X) = [\![X]\!]_{\underline{T}}(r \circ \phi).$$

$A \equiv \mu X.B$. By definition we have

$$r([\![\mu X.B]\!]_T\ \phi) \;\; = \;\; r(\mu\theta) \qquad \text{(A.1)}$$

where $\theta : \mathcal{P}(S_T) \to \mathcal{P}(S_T)$ is defined by $\theta(U) = [\![B]\!]_T \phi[U/X]$. Taking $\psi(V) = [\![B]\!]_{\underline{T}}(r \circ \phi)[V/X]$ we obtain:

$$\begin{aligned} r \circ \theta(U) &= r([\![B]\!]_T\ \phi[U/X])) \\ &= [\![B]\!]_{\underline{T}}(r \circ \phi)[r(U)/X] \\ &\qquad \text{by the induction hypothesis} \\ &= \psi(r(U)) = \psi \circ r(U) \end{aligned}$$

Furthermore r is easily seen to be strict and ω-continuous, and as we assume S_T to be countable, the reduction lemma yields

$$r(\mu\theta) = \mu\psi$$

which by expanding ψ and (A.1) gives the result:

$$\begin{aligned} r([\![\mu X.B]\!]_T\ \phi &= \mu V \subseteq S_{\underline{T}}.[\![B]\!]_{\underline{T}}(r \circ \phi)[V/X] \\ &= [\![\mu X.B]\!]_{\underline{T}}(r \circ \phi). \end{aligned}$$

$A \equiv \langle\alpha\rangle B, \alpha \neq *$. We proceed by rewriting the left-hand side:

$$\begin{aligned} r([\![\langle\alpha\rangle B]\!]_T\ \phi) &= r(\{s \in S_T \mid \exists s' \in S_T.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B]\!]_T\ \phi\}) \\ &\qquad \text{by definition} \\ &= r(\{s \in S_T \mid \exists s' \in S_T.s \xrightarrow{\alpha} s' \ \&\ s' \in r([\![B]\!]_T\ \phi)\}) \\ &\qquad \text{by def. of rooting} \\ &= r(\{s \in S_T \mid \exists s' \in S_T \cup \{\underline{i}\}.s \xrightarrow{\alpha} s' \ \&\ s' \in r([\![B]\!]_T\ \phi)\}) \\ &\qquad \text{as no transitions enter } \underline{i} \\ &= r(\{s \in S_T \mid \exists s' \in S_T \cup \{\underline{i}\}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B]\!]_{\underline{T}}(r \circ \phi)\}) \\ &\qquad \text{by the induction hypothesis} \\ &= \{i, \underline{i} \mid \exists s' \in S_{\underline{T}}.i \xrightarrow{\alpha} s' \ \&\ s' \in [\![B]\!]_{\underline{T}}(r \circ \phi)\} \\ &\quad \cup \{s' \in S_{\underline{T}} \setminus \{i, \underline{i}\} \mid \exists s' \in S_{\underline{T}}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B]\!]_{\underline{T}}(r \circ \phi)\} \\ &\qquad \text{by applying } r \\ &\qquad [\![\langle\alpha\rangle B]\!]_{\underline{T}}(r \circ \phi). \end{aligned}$$

$A \equiv A_0 \vee A_1$ and $A \equiv \neg B$. Immediate as r distributes over disjunction and negation.

□

A.2 Proof of Reduction Lemma

We prove the reduction lemma as a corollary of a more general result, for which we need the notion of *height* of a partial order.

Definition A.1 Define the *height* of a partial order $(D, \leq)$ to be the smallest cardinal κ such that for any $T \subseteq D$, where $(T, \leq \cap T \times T)$ is totally ordered, $|T| \leq \kappa$. Say a partial order has *countable height* if its height is countable. □

Note that if X is a countable set, then the partial orders $(\mathcal{P}(X), \subseteq)$ and $(\mathcal{P}(X), \supseteq)$ have countable height.

Lemma A.1 *Let D, E be complete lattices of mountable height. Let $in : D \to E$ be an ω-continuous function such that $in(\perp_D) = \perp_E$. Suppose $\varphi : E \to E$ and $\theta : D \to D$ are monotonic functions such that*

$$in \circ \theta = \varphi \circ in.$$

Then

$$in(\mu\theta) = \mu\varphi.$$

Proof: The following facts are well–known (see e.g. [2]):
For a monotonic function $\theta : D \to D$,

1. $\mu\theta = \bigvee_{\alpha \in O_n} \theta^\alpha(\perp_D)$,
   where
   $\theta^0(x) =_{def} x$,
   $\theta^{\alpha+1}(x) =_{def} (\theta^\alpha(x))$, and
   $\theta^\lambda(x) =_{def} \bigvee_{\alpha<\lambda} \theta^\alpha(x)$ for λ a limit-ordinal,
   are such that $\alpha \leq \alpha' \Rightarrow \theta^\alpha(\perp_D) \leq \theta^{\alpha'}(\perp_D)$.

2. In addition, there is a least ordinal β (the closure ordinal) such that $\theta^\beta(\perp_D) = \theta^{\beta+1}(\perp_D)$. Then $\mu\theta = \theta^\beta(\perp_D)$.

Further:

3. If D is of height ω then β is a countable ordinal: The function $\alpha \longmapsto \theta^\beta(\perp_D)$ for $\alpha \in \beta$ is 1-1 and has range a total order in D; hence because D has height ω theordinal β is countable. It follows that when D has height ω then

$$\mu\theta = \bigvee_{\alpha \in Con} \theta^{\alpha}(\bot_D),$$

where Con is the set of countable ordinals.

Now, we proceed to the main proof. Under the assumptions stated in the lemma we see

$$\begin{aligned} in(\mu\theta) &= in(\textstyle\bigvee_{\alpha \in Con} \theta^{\alpha}(\bot_D)) \\ &= in(\theta^{\beta}(\bot_D)) \text{ where } \beta \text{ is the closure ordinal as in (3) above} \\ &= \textstyle\bigvee_{\alpha \in Con} in(\theta^{\alpha}(\bot_D)) \text{ by } \omega\text{-continuity of } in. \qquad \text{(A)} \end{aligned}$$

By ordinal induction we show:

$$in(\theta^{\alpha}(\bot_D)) = \varphi^{\alpha}(in(\bot_D)) \qquad \text{(B)}$$

for all $\alpha \in Con$:

When $\alpha = 0$ then $in(\theta^{\alpha}(\bot_D)) = in(\bot_D) = \varphi^{\alpha}(in(\bot_D))$.

For a successor ordinal,

$$\begin{array}{rcll} in(\theta^{\alpha+1}(\bot_D)) & = & in(\theta(\theta^{\alpha(\bot_D)})) & \text{by definition} \\ & = & \varphi(in(\theta^{\beta}(\bot_D))) & \text{as } in \circ \theta = \varphi \circ in \\ & = & \varphi(\varphi^{\alpha}(in(\bot_D))) & \text{by induction} \\ & = & \varphi^{\alpha+1}(in(\bot_D)). & \end{array}$$

Assume λ is a countable limit ordinal. Then λ is co-final with ω in the sense that there is an ω-sequence of elements of λ

$$\beta_0, \beta_1, \ldots, \beta_n, \ldots$$

such that for all $\alpha \in \lambda$ there is some $n \in \omega$ such that $\alpha \leq \beta_n$: With respect to $\alpha_0, \alpha_1, \ldots, \alpha_n, \ldots$ a countable enumeration of elements of λ, take $\beta_0 = \alpha_0$ and inductively take β_{n+1} to be the maximum of β_n and α_{n+1}.

Now we argue:

$$\begin{array}{rcll} in(\theta^{\lambda}(\bot_D)) & = & in(\bigvee_{\alpha<\lambda} \theta^{\alpha}(\bot_D)) & \\ & = & in(\bigvee_{n\in\omega} \theta^{\beta_n}(\bot_D)) & \text{by cofinality} \\ & = & \bigvee_{n\in\omega} in(\theta^{\beta_n}(\bot_D)) & \text{by } \omega\text{-continuity of } in \\ & = & \bigvee_{n\in\omega} \varphi^{\beta_n}(in(\bot_D)) & \text{by induction} \\ & = & \bigvee_{\alpha<\lambda} \varphi^{\alpha}(in(\bot_D)) & \text{by iconfinality} \\ & = & \varphi^{\lambda}(in(\bot_D)). & \end{array}$$

This completes the inductive proof of (B).

Recalling $in(\perp_D) = \perp_E$, we conclude:

$$\begin{array}{rcll} in(\mu\theta) & = & \bigvee_{\alpha \in Con} in\ \theta^\alpha(\perp_D)) & \text{by (A)} \\ & = & \bigvee_{\alpha \in Con} \varphi^\alpha in(\perp_D)) & \text{by (B)} \\ & = & \bigvee_{\alpha \in Con} \varphi^\alpha(\perp_E)) & \\ & = & \mu\varphi & \text{by (3).} \end{array}$$

□

We remark that the ω-continuity of in is necessary, as the following example shows.

Example A.1 Let E consist of $\perp < \top$ and D be the ordinal $\omega + 1$ ordered by the usual ordering on ordinals. Let $in : D \to E$ be the monotonic (but not continuous) function such that

$$\begin{array}{rcll} in(n) & = & \perp & \text{for } n \in \omega, \\ in(\omega) & = & \top. & \end{array}$$

Take $\varphi : E \to E$ to be the identity on E, and $\theta : D \to D$ to act so

$$\begin{array}{rcll} \theta(n) & = & n+1 & \text{for } n \in \omega, \\ \theta(\omega) & = & \omega. & \end{array}$$

Then $\mu\varphi = \perp$ and $\mu\theta = \omega$. Hence in this case where in is monotonic and not continuous we have $\mu\varphi = \perp$ and $in(\mu\theta) = \top$ so $\mu\varphi \neq in(\mu\theta)$. (Monotonicity of in guarantees $\mu\varphi \leq in(\mu\theta)$.) □

If D, E are powersets of countable sets then they are complete lattices of height ω and so meet the conditions required by the reduction lemma, yielding the special case, lemma 3.2, used in chapter 3.

A.3 Proof of Reduction for Prefix

Reduction for prefix, theorem 3. *Given a closed, pure assertion A, a change of variables σ which is fresh for A, and an arbitrary process term t, then*

$$\models (at : A) \leftrightarrow \mathrm{red}^1(at : A; \sigma).$$

□

Proof: We prove by structural induction on A that for a change of variables σ which is fresh for A, we have for all environments ρ:

$$[\![A[\sigma]]\!]_{\underline{at}}\ \rho = \mathit{in}([\![\mathrm{red}^0(at : A; \sigma)]\!]_t\ \rho, [\![\mathrm{red}^1(at : A; \sigma)]\!]_\bullet\ \rho). \tag{1}$$

The result then follows from the discussion preceding theorem 3.1.

$A \equiv X$. Assuming that $\sigma(X) = \mathit{IN}(X_0, X_1)$ we get

$$\begin{array}{rcl} [\![X[\sigma]]\!]_{\underline{at}}\ \rho & = & \mathit{in}(\rho(X_0), \rho(X_1)) \\ & = & \mathit{in}([\![X_0]\!]_t\ \rho, [\![X_1]\!]_\bullet\ \rho) \\ & = & \mathit{in}([\![\mathrm{red}^0(at : X; \sigma)]\!]_t\ \rho, [\![\mathrm{red}^1(at : X; \sigma)]\!]_\bullet\ \rho) \end{array}$$

$A \equiv \mu X.B$ By definition we have

$$[\![(\mu X.B)[\sigma]]\!]_{\underline{at}}\ \rho = \mu\psi,$$

where ψ is defined by

$$\psi(U) = [\![B[\sigma \setminus X]]\!]_{\underline{at}}\ \rho[U/X].$$

Taking as abbreviations $B^0 = \mathrm{red}^0(at : B; \sigma)$ and $B^1 = \mathrm{red}^1(at : B; \sigma)$ and defining

$$\theta(V_0, V_1) = ([\![B^0]\!]_t\ \rho[V_0/X_0, V_1/X_1], [\![B^1]\!]_\bullet\ \rho[V_0/X_0, V_1/X_1])$$

we can show that θ and ψ are related as required by the reduction lemma:

$$\begin{array}{rcl} \mathit{in} \circ \theta(V_0, V_1) & = & \mathit{in}([\![B^0]\!]_t\ \rho[V_0/X_0, V_1/X_1], [\![B^1]\!]_\bullet\ \rho[V_0/X_0, V_1/X_1]) \\ & = & [\![B[\sigma]]\!]_{\underline{at}}\ \rho[V_0/X_0, V_1/X_1] \\ & & \quad \text{by the ind. hyp.} \\ & = & [\![B[\sigma \setminus X]]\!]_{\underline{at}}\ \rho[\mathit{in}(V_0, V_1)/X] \\ & & \quad \text{as } \sigma(X) = \mathit{IN}(X_0, X_1) \text{ and } \sigma \text{ is fresh for } \mu X.B \\ & = & \psi \circ \mathit{in}(V_0, V_1) = \\ & & \quad \text{by def. of } \psi. \end{array}$$

It is easy to see that *in* is strict and ω-continuous, hence the reduction lemma yields

$$in(\nu\theta) = \mu\psi$$

Writing out $\mu\theta$ in full details we can proceed by applying Bekič's theorem:

$$\begin{aligned}
\mu\theta &= \mu(V_0,V_1).([\![B^0]\!]_t\ \rho[V_0/X_0,V_1/X_1],\ [\![B^1]\!]_\bullet\ \rho[V_0/X_0,V_1/X_1]) \\
&= (\mu V_0.[\![B^0]\!]_t\ \rho[V_0/X_0],\ \mu V_1.[\![B^1]\!]_\bullet\ \rho[(\mu V_0.[\![B^0]\!]_t\ \rho[V_0/X_0])/X_0,V_1/X_1]) \\
&\qquad \text{by Beki\v{c}'s theorem and the observation that } X_1 \text{ is not free in } B^0 \\
&= ([\![\mu X_0.B^0]\!]_t\ \rho,\ \mu V_1.[\![B^1]\!]_\bullet\ \rho[[\![\mu X_0.B^0]\!]_t\ \rho/X_0,V_1/X_1]) \\
&\qquad \text{by definition} \\
&= ([\![\mu X_0.B^0]\!]_t\ \rho,\ \mu V_1.[\![B^1[\mu X_0.B^0/X_0]]\!]_\bullet\ \rho[V_1/X_1]) \\
&\qquad \text{by the substitution lemma} \\
&= ([\![\mu X_0.B^0]\!]_t\ \rho,\ [\![B^1[\mu X_0.B^0/X_0]]\!]_\bullet\ \rho[\emptyset/X_1]) \\
&\qquad \text{as } \mathcal{P}(\{\bullet\}) \text{ is just a two} - \text{point lattice with bottom element } \emptyset \\
&= ([\![\mu X_0.B^0]\!]_t\ \rho,\ [\![B^1[\mu X_0.B^0/X_0][F/X_1]]\!]_\bullet\ \rho) \\
&\qquad \text{by the substitution lemma} \\
&= ([\![\mathrm{red}^0(at:\mu X.B;\sigma)]\!]_t\ \rho,\ [\![\mathrm{red}^1(at:\mu X.B;\sigma)]\!]_\bullet\ \rho).
\end{aligned}$$

We have established that

$$[\![(\mu X.B)[\sigma]]\!]_{\underline{at}}\ \rho = in([\![\mathrm{red}^0(at:\mu X.B;\sigma)]\!]_t\ \rho, [\![\mathrm{red}^1(at:\mu X.B;\sigma)]\!]_\bullet\ \rho)$$

as required.

$A \equiv \langle\alpha\rangle B, \alpha \neq *$. We rewrite from the definition:

$$\begin{aligned}
[\![\langle\alpha\rangle B[\sigma]]\!]_{\underline{at}}\ \rho &= \{s \in S_{\underline{at}} \mid \exists s' \in S_{\underline{at}}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B[\sigma]]\!]_{\underline{at}}\ \rho\} \\
&= \{s \in S_{\underline{at}} \mid \exists s' \in S_{\underline{at}}.s \xrightarrow{\alpha} s' \ \&\ s' \in in([\![B^0]\!]_t\ \rho, [\![B^1]\!]_\bullet\ \rho)\} \\
&\qquad \text{by ind. hyp. where } B^0 \text{ abbreviates } \mathrm{red}^0(at:B;\sigma) \\
&\qquad \text{and} B^1 \text{abbreviates } \mathrm{red}^1(at:B;\sigma) \\
&= \{s \in S_{\underline{at}} \mid \exists s' \in S_{\underline{at}} \setminus \{\underline{at}\}.s \xrightarrow{\alpha} s' \ \&\ s' \in in([\![B^0]\!]_t\ \rho, [\![B^1]\!]_\bullet\ \rho)\} \\
&\qquad \text{as no transitions enter } \underline{at} \\
&= \{s \in S_{\underline{at}} \mid \exists s' \in S_{\underline{at}} \setminus \{\underline{at}\}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B^0]\!]_t\ \rho\} \\
&\qquad \text{by definition of } in \\
&= \{\underline{at} \mid \exists s' \in S_{\underline{at}} \setminus \{\underline{at}\}.\underline{at} \xrightarrow{\alpha} s' \ \&\ s' \in [\![B^0]\!]_t\ \rho\} \\
&\qquad \cup\{s \in S_{\underline{at}} \setminus \{\underline{at}\} \mid \exists s' \in S_{\underline{at}} \setminus \{\underline{at}\}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B^0]\!]_t\ \rho\} \\
&\qquad \text{by simple splitting}
\end{aligned}$$

$$\begin{array}{rl}
= & \{\underline{at} \mid \alpha = a \ \& \ t \in [\![B^0]\!]_t \ \rho\} \\
& \cup\{s \in S_t \mid \exists s' \in S_t . s \xrightarrow{\alpha} s' \ \& \ s' \in [\![B^0]\!]_t \ \rho\} \\
& \quad \text{as the only transition from } \underline{at} \text{ is } \underline{at} \xrightarrow{a} t \text{ and} \\
& \quad \text{by observing } S_{\underline{at}} \setminus \{\underline{at}\} = S_t \\
= & \begin{cases} [\![\langle\alpha\rangle B^0]\!]_t \ \rho & \textit{if } \alpha \neq a \\ \{\underline{at} \mid t \in [\![B^0]\!]_t \ \rho\} & \textit{if } \alpha = a \end{cases} \\
& \quad \text{from the definition of } [\![\langle\alpha\rangle B^0]\!]_t \ \rho \\
= & \mathit{in}([\![\mathrm{red}^0(at; \langle\alpha\rangle B; \sigma)]\!]_t \ \rho, [\![\mathrm{red}^1(at; \langle\alpha\rangle B; \sigma)]\!]_\bullet \ \rho \\
& \quad \text{by def. of } \mathrm{red}^0, \mathrm{red}^1, \text{and } \mathit{in}
\end{array}$$

$A \equiv A_0 \vee A_1$ and $A \equiv \neg B$. Straightforward.

□

A.4 Proof of Reduction for Restriction

Reduction for restriction, theorem 3.5 *Assume A closed and pure, a change of variables σ which is fresh for A and an arbitrary process term t, then*

$$\models (t \restriction \Lambda : A) \leftrightarrow (t : \mathrm{red}_{\restriction\Lambda}(A; \sigma)).$$

□

Proof: We show by structural induction on A that for a change of variables σ which is fresh for A we have, for all ρ:

$$[\![A[\sigma]]\!]_{t\restriction\Lambda} \ \rho = \mathit{in}([\![\mathrm{red}(t \restriction \Lambda : A; \sigma)]\!]_t \ \rho). \tag{A.2}$$

From (A.2) and the definition of *in* it follows that

$$t \restriction \Lambda \in [\![A[\sigma]]\!]_{t\restriction\Lambda} \ \rho \ \textit{iff} \ t \in [\![\mathrm{red}(t \restriction \Lambda : A; \sigma)]\!]_t \ \rho$$

hence by the locality lemma

$$\models (t \restriction \Lambda : A) \leftrightarrow (t : \mathrm{red}(t \restriction \Lambda : A; \sigma)),$$

as required.

$A \equiv X$. Assuming that $\sigma(X) = IN(Y)$ we get:

$$\begin{aligned}
[\![X[\sigma]]\!]_{t\restriction\lambda}\rho &= in(\rho(Y)) \\
&= in([\![Y]\!]_t\ \rho) \\
&= in([\![\mathrm{red}(t \restriction \Lambda : X; \sigma]\!]_t\ \rho).
\end{aligned}$$

$A \equiv \mu X.B$. By definition we have

$$[\![(\mu X.B)[\sigma]]\!]_{t\restriction\lambda}\ \rho = \mu\psi$$

where $\psi : \mathcal{P}(S_{t\restriction\Lambda}) \to \mathcal{P}(S_{t\restriction\Lambda})$ is defined by

$$\psi(U) = [\![B[\sigma/X]]\!]_{t\restriction\Lambda}\ \rho[U/X]$$

Defining $\theta : \mathcal{P}(S_t) \to \mathcal{P}(S_t)$ by

$$\theta(V) = [\![\mathrm{red}(t \restriction \Lambda : B; \sigma)]\!]_t\ \rho[V/Y],$$

we show that θ and ψ are related as required by the reduction lemma:

$$\begin{aligned}
in \circ \theta(V) &= in([\![\mathrm{red}(t \restriction \Lambda : B; \sigma)]\!]_t\ \rho[V/Y]) \\
&= [\![B[\sigma]]\!]_t\ \rho[V/Y] \\
&\qquad \text{by the induction hypothesis} \\
&= [\![B[\sigma/X]]\!]_t\ \rho[in(V)/X] \\
&\qquad \text{as } \sigma(X) = IN(Y) \text{ and } \sigma \text{ is fresh for } \mu X.B \\
&= \psi \circ in(V) \\
&\qquad \text{by definition of } \psi.
\end{aligned}$$

It is easy to see that in is strict and ω-continuous, hence the reduction lemma applies, yielding

$$\mu\psi = in(\mu\theta),$$

hence

$$\begin{array}{rcl}
[\![(\mu X.B)[\sigma]]\!]_{t\upharpoonright\Lambda}\ \rho & = & in(\mu V \subseteq S_{t\upharpoonright\Lambda}.[\![\text{red}(t\upharpoonright\Lambda : B;\sigma)]\!]_t\ \rho[V/Y]) \\
& = & in([\![\mu Y.\text{red}(t\upharpoonright\Lambda : B;\sigma)]\!]_t\ \rho) \\
& & \quad \text{by definition of the } \mu\text{−operator} \\
& = & in([\![\text{red}(t\upharpoonright\Lambda : B;\sigma)]\!]_t\ \rho) \\
& & \quad \text{by definition of red}(t\upharpoonright\Lambda : \mu Y.B;\sigma).
\end{array}$$

$A \equiv \langle\alpha\rangle B, \alpha \neq *$. We rewrite the left-hand side:

$$\begin{array}{rcl}
[\![\langle\alpha\rangle B[\sigma]]\!]_{t\upharpoonright\Lambda}\ \rho & = & \{s \in S_{t\upharpoonright\Lambda} \mid \exists s' \in S_{t\upharpoonright\Lambda}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B[\sigma]]\!]_{t\upharpoonright\Lambda}\ \rho\} \\
& & \quad \text{by definition} \\
& = & \{s \in S_{t\upharpoonright\Lambda} \mid \exists s' \in S_{t\upharpoonright\Lambda}.s \xrightarrow{\alpha} s' \ \&\ s' \in in([\![\text{red}(t\upharpoonright\Lambda;\sigma)]\!]_t\ \rho)\} \\
& & \quad \text{by the induction hypothesis } A.2 \\
& = & \left\{ \begin{array}{ll} \{s\upharpoonright\Lambda \mid s \in S_t \ \&\ \exists s' \in S_t.s \xrightarrow{\alpha} s' \ \& & \\ \quad s' \in [\![\text{red}(t\upharpoonright\Lambda : B;\sigma)]\!]_t\ \rho\} \cup S_{t\upharpoonright\Lambda} & \textit{if } \alpha \in \Lambda \\ \emptyset & \textit{if } \alpha \notin \Lambda \end{array} \right. \\
& & \quad \text{by definition of } in \text{ and the restriction operator} \\
& = & \left\{ \begin{array}{ll} in([\![\langle\alpha\rangle\text{red}(t\upharpoonright\Lambda : B;\sigma)]\!]_t\ \rho) & \textit{if } \alpha \in \Lambda \\ in(\emptyset) & \textit{if } \alpha \notin \Lambda \end{array} \right. \\
& & \quad \text{by definition of } in \text{ and } \langle\alpha\rangle \\
& = & in([\![\text{red}(t\upharpoonright\Lambda : \langle\alpha\rangle B;\sigma)]\!]_t\ \rho) \\
& & \quad \text{by definition of red}(t\upharpoonright\Lambda : \langle\alpha\rangle B;\sigma).
\end{array}$$

$A \equiv A_0 \vee A_1$ and $A \equiv \neg B$. Straightforward.

$\square$

A.5 Proof of Reduction for Recursion

In order to show the correctness of the reduction for recursion we will need a small lemma which describe a useful relationship between transitions in $\underline{t}$ and *rec P.t*.

Lemma A.2 *Let j be the function described in the main text. Then for all $s, s' \in S_{\underline{t}}$ and $\alpha \neq *$ we have:*

$$j(s) \xrightarrow{\alpha} j(s')$$

if and only if,

$$\exists s'' \in S_{\underline{t}}.\ j(s'') = j(s')\&\ ((s = P\ \&\ t \xrightarrow{\alpha} s'')\ \mathit{or}\ (s \neq P\ \&\ s \xrightarrow{\alpha} s'')).$$

Proof: Suppose $s = P$. Then $j(s) = \mathit{rec}\ P.t$ and

$$\begin{array}{lll} \mathit{rec}\ P.t \xrightarrow{\alpha} j(s') & \mathit{iff} & t[\mathit{rec}\ P.t/P] \xrightarrow{\alpha} j(s') \\ & & \quad \text{as only the `unfolding rule` apply when } \alpha \neq * \\ & \mathit{iff} & \exists s'' \in S_{\underline{t}}.\ t \xrightarrow{\alpha} s''\ \&\ s''[\mathit{rec}\ P.t/P] = j(s') \\ & & \quad \text{as} P \text{is strongly guarded} \\ & \mathit{iff} & \exists s'' \in S_{\underline{t}}.\ t \xrightarrow{\alpha} s''\ \&\ j(s'') = j(s') \\ & & \quad \text{by definition of } j. \end{array}$$

Now suppose $s \neq P$. We first consider the case where $j(s) \neq \mathit{rec}\ P.t$, i.e. $s \notin \{\underline{t}, \mathit{rec}\ P.t\}$. Then as P is strongly guarded, the first transition from $j(s)$ is independent of whether $\mathit{rec}\ P.t$ is substituted for P or not:

$$j(s) \xrightarrow{\alpha} j(s')\ \mathit{iff}\ \exists s'' \in S_{\underline{t}}.\ s \xrightarrow{\alpha} s''\ \&\ j(s'') = j(s').$$

When $s = \underline{t}$ we get by the same arguments as in the case of $s = P$, that

$$j(s) \xrightarrow{\alpha} j(s')\ \mathit{iff}\ \exists s'' \in S_{\underline{t}}.\ t \xrightarrow{\alpha} s''\ \&\ j(s'') = j(s'),$$

which by definition of rooting is equivalent to

$$\exists s'' \in S_{\underline{t}}.\ s \xrightarrow{\alpha} s''\ \&\ j(s'') = j(s').$$

For $s = \mathit{rec}\ P.t$ the result is trivial as $j(s) = s$. □

In the inductive proof of correctness it turns out that we will need a stronger induction hypothesis than for the other reductions. We will introduce a notion of 'balanced subset', in the sense that if a state $s \in S_{\underline{t}}$ belongs to the subset, then every other state, which under j maps to the same state in $S_{rec\ P.t}$ belongs to the subset. Formally, a subset $U \subseteq S_{\underline{t}}$ is said to be *balanced* if $j^{-1} \circ in(U) = U$. Note that if j is injective all subsets are trivially balanced. An environment ρ is said to be balanced if $\rho(X)$ is balanced for all variables X. It is easily seen that $D = \{U \subseteq S_{\underline{t}} \mid U \text{ is balanced}\}$ is a complete sublattice of $\mathcal{P}(S_{\underline{t}})$.

We are now able to prove theorem 3.6 (reduction for recursion):

Reduction for recursion, theorem 3.6 *Given a closed, pure assertion A, a change of variables σ which is fresh for A, and a regular process term t in which P is strongly guarded then*

$$\models (rec\ P.t : A) \leftrightarrow (t : \mathrm{red}(rec\ P.t : A; \sigma)).$$

Proof: By structural induction on A, we show that $P(A)$ holds for all A, where P is defined by:

$$\begin{aligned} P(A) \Leftrightarrow_{def} \text{for all balanced } \rho. \quad & [\![A[\sigma]]\!]_{rec\ P.t}\ \rho = \\ & in([\![\mathrm{red}(rec\ P.t : A; \sigma)]\!]_{\underline{t}}\rho) \qquad \text{(A.3)} \\ \& \quad & [\![\mathrm{red}(rec\ P.t : A; \sigma)]\!]_{\underline{t}}\ \rho \in D \end{aligned}$$

From this it follows that

$$\models (rec\ P.t : A) \leftrightarrow (t : \mathrm{red}(rec\ P.t : A; \sigma))$$

for all closed, pure A.

$A \equiv X$. Assuming that $\sigma(X) = IN(Y)$ we have by definition,

$$[\![X[\sigma]]\!]_{rec\ P.t}\ \rho = [\![IN(Y)]\!]_{rec\ P.t}\ \rho = in(\rho(Y)) = in([\![Y]\!]_{\underline{t}}\ \rho).$$

From the assumption that ρ is balanced we immediately get $[\![Y]\!]_{\underline{t}}\ \rho \in D$.

$A \equiv \mu X.B$. By definition we have

$$[\![(\mu X.B)[\sigma]]\!]_{rec\ P.t}\ \rho = \mu\psi$$

where $\psi : \mathcal{P}(S_{rec\ P.t}) \rightarrow \mathcal{P}(S_{rec\ P.t})$ is defined by

$$\psi(U) = [\![B[\sigma \setminus X]]\!]_{rec\ P.t}\ \rho[U/X].$$

Defining $\theta : \mathcal{P}(S_{\underline{t}}) \to \mathcal{P}(S_{\underline{t}})$ by

$$\theta(V) = [\![\mathrm{red}(rec\ P.t : B; \sigma)]\!]_{\underline{t}}\ \rho[V/Y],$$

we show that ψ and θ are related as required by the reduction lemma:

$$\begin{array}{rcl} in \circ \theta(V) & = & in([\![\mathrm{red}(rec\ P.t : B; \sigma)]\!]_t\ \rho[V/Y]) \\ & = & in([\![B[\sigma]]\!]_{rec\ P.t}\ \rho[V/Y]) \\ & & \quad \text{by the ind. hyp. (A.3)} \\ & = & [\![B[\sigma/X]]\!]_{rec\ P.t}\rho\ [in(V)/X]) \\ & & \quad \text{as}\sigma(X) = IN(Y) \text{ and } \sigma \text{ is fresh for } \mu X.B \\ & = & \psi \circ in(V) \\ & & \quad \text{by def. of } \psi \end{array}$$

It is easy to see that in is strict and ω-continuous, hence the reduction lemma yields

$$in(\mu\theta) = \mu\psi$$

Writing out θ and ψ and using the definition of the μ-operator we get:

$$[\![(\mu X.B)[\sigma]]\!]_{rec\ P.t}\rho = in([\![\mathrm{red}(rec\ P.t : \mu Y.B; \sigma)]\!]_{\underline{t}}\rho).$$

Moreover, θ restricts to a function θ' on D, as can be seen from the induction hypothesis: For a balanced environment ρ, $P(B)$ states that $\theta(V)$ is balanced for all balanced V, i.e. θ maps balanced sets to balanced sets. Hence, letting in' be the embedding of D into $\mathcal{P}(S_{\underline{t}})$ – easily seen to be strict and ω-continuous – we have that

$$\theta \circ in' = i,' \circ \theta'$$

which by the reduction lemma gives $\mu\theta = in'(\mu\theta')$. In other words $\mu\theta \in D$.

$A \equiv \langle\alpha\rangle B, \alpha \neq *$. We rewrite from the left-hand side:

$$[\![\langle\alpha\rangle B[\sigma]]\!]_{rec\ P.t}\ \rho$$

$= \{s \in S_{rec\ P.t} \mid \exists s' \in S_{rec\ P.t}.s \xrightarrow{\alpha} s' \ \&\ s' \in [\![B[\sigma]]\!]_{rec\ P.t}\ \rho\}$
by definition
$\{s \in S_{rec\ P.t} \mid \exists s' \in S_{rec\ P.t}.s \xrightarrow{\alpha} s' \ \&\ s' \in in[\![B']\!]_{\underline{t}}\ \rho\}$
by the induction hypothesis where $B' = \mathrm{red}(rec\ P.t : B; \sigma)$
$\overset{*}{=} in(\{s \in S_{\underline{t}} \mid \exists s' \in S_{\underline{t}}.j(s) \xrightarrow{\alpha} j(s') \ \&\ j(s') \in in([\![B']\!]_{\underline{t}}\ \rho)\})$
by the fact that j is surjective
$in(\{s \in S_{\underline{t}} \mid \exists s' \in S_{\underline{t}}.j(s') \in in([\![B']\!]_{\underline{t}}\ \rho) \ \&\ ((s = P \ \&\ t \xrightarrow{\alpha} s')$
$or\ s \xrightarrow{\alpha} s')\})$
by lemma A.2
$= in(\{s \in S_{\underline{t}} \mid \exists s' \in S_{\underline{t}}.s' \in [\![B']\!]_{\underline{t}}\ \rho \ \&\ ((s = P \ \&\ t \xrightarrow{\alpha} s')\ or\ s \xrightarrow{\alpha} s')\})$
by the second part of the induction hypothesis
$= in([\![(\widehat{P} \wedge (t : \langle\alpha\rangle B')) \vee \langle\alpha\rangle B']\!]_{\underline{t}}\ \rho)$
by definition of $[\![\_]\!]_{\underline{t}}\ \rho)$
$= in([\![\mathrm{red}(rec\ P.t : \langle\alpha\rangle B; \sigma)]\!]_{\underline{t}}\ \rho)$
by definition

Let $in(U)$ be the right-hand side of the third equality (marked $*$). It is easy to observe that $j^{-1}(in(U)) = U$ as the predicate determining whether $s \in U$ only depends on the value of $j(s)$. Moreover, notice that the last five equalities hold without in, hence $[\![\mathrm{red}(rec\ P.t : \langle\alpha\rangle B; \sigma)]\!]_{\underline{t}}\ \rho = U$ and is therefore balanced (this property actually dictated the construction of the reduction for $\langle\alpha\rangle$).

$A \equiv A_0 \vee A_1$ and $A \equiv \neg B$. Simple.

□

A.6 Proof of Reduction for Product

Reduction for product, theorem 3.7 *Assume given a pure and closed assertion A of type $\eta_1 \times \eta_2$, a change of variables σ, and a term p of type η_2 with no restrictions and relabellings. We then have for an arbitrary term q of type η_1:*

$$\models (q \times p : A) \leftrightarrow (q : \mathrm{red}_{\times p}(A; \sigma)).$$

□

Proof: We will prove that

$$[\![A[\sigma]]\!]_{q\times p}\ \rho \quad = \quad in([\![A/p_1]\!]_q\ \rho, \ldots, [\![A/p_n]\!]_q\ \rho) \tag{A.4}$$

for all environments ρ. Assuming without loss of generality that $p = p_1$ it follows that

$$\models q \times p : A \leftrightarrow q : (A/p_1).$$

Let $slice_i : \mathcal{P}(S_{q\times p}) \to \mathcal{P}(S_q)$ be the function that projects onto the i'th component, i.e.

$$slice_i(U) = \{s \in S_q \mid s \times p_i \in U\}$$

From the definition of in, it is easy to see that (A.4) is equivalent to the following:

$$\forall 1 \leq i \leq n.\ slice_i([\![A[\sigma]]\!]_{q\times p}\ \rho) \quad = \quad [\![A/p_1]\!]_q\ \rho, \tag{A.5}$$

which we will take as our induction hypothesis (but apply (A.4) when most appropriate).

$A \equiv X$. By definition we immediately have

$$slice_i([\![X[\sigma]]\!]_{q\times p}\ \rho) = \rho(X_{p_i}) = [\![X/p_i]\!]_q\ \rho$$

$A \equiv \mu X.B$. Let $\theta : \mathcal{P}(S_q)^n \to \mathcal{P}(S_q)^n$ be defined by

$$\theta(V_1, \ldots, V_n) = ([\![B/p_1]\!]_q\ \rho', \ldots, [\![B/p_n]\!]_q\ \rho'),$$

where

$$\rho' = \rho[V_1/X_{p_1}, \ldots, V_n/X_{p_n}].$$

Let $\psi : \mathcal{P}(S_{q\times p}) \to \mathcal{P}(S_{q\times p})$ be defined by

$$\psi(U) = [\![B[\sigma/X]]\!]_{q\times p}\ \rho[U/X].$$

We show that θ and ψ are as required by the reduction lemma:

$$\begin{array}{rcl} in \circ \theta(V_1, \ldots, V_n) & = & [\![B[\sigma]]\!]_{q\times p}\ \rho[V_1/X_{p_1}, \ldots, V_n/X_{p_n}] \\ & & \quad \text{by the induction hypothesis} \\ & = & [\![B[\sigma/X]]\!]_{q\times p}\ \rho[in(V_1, \ldots V_n)/X]) \\ & & \quad \text{as } \sigma(X) = IN(X_{p_1}, \ldots, X_{p_n}) \text{ and } \sigma \text{ is fresh} \\ & = & \psi \circ in(V_1, \ldots, V_n) \\ & & \quad \text{by def. of } \psi \end{array}$$

From the reduction lemma we now conclude:

$$in(\mu\theta) = \mu\psi$$

which, by writing out θ and ψ, yields

$$in(\mu\vec{V}.([\![B/p_1]\!]_q\ \rho[\vec{V}/\vec{X}], \ldots, [\![B/p_n]\!]_q\ \rho[\vec{V}/\vec{X}])) = [\![\mu X.B]\!]_{q\times p}\ \rho.$$

By repeated application of Bekič's theorem, the simultaneous fixed-point on the left-hand side can be converted into a unary fixed-point, yielding the claimed reduction.

$A \equiv \neg B$ and $A \equiv A_0 \wedge A_1$. Immediate by definition.

$p_i \equiv r \times s$. We rewrite from the left-hand side:

$$\begin{array}{rcl} slice_i([\![A[\sigma]]\!]_{q\times p}\ \rho) & = & \{u \in S_q \mid u \times (r \times s) \in [\![A(\sigma]]\!]_{q\times p}\ \rho\} \\ & & \quad \text{by definition of } slice_i \\ & = & \{u \in S_q \mid (u \times r) \times s \in [\![\tilde{A}[\sigma]]\!]_{q\times p}\ \rho\} \\ & & \text{by reassociating modalities in } A \\ & = & [\![(\tilde{A}/s)/r]\!]_{q\times p}\ \rho \\ & & \quad \text{by definition.} \end{array}$$

$A \equiv \langle \alpha \times \beta \rangle$ and $p_i \equiv nil, \alpha \times \beta \neq *$. We immediately get:

$$
\begin{array}{ll}
\multicolumn{2}{l}{slice_i([\![\langle\alpha \times \beta\rangle B]\!]_{q\times p}\rho)} \\
= & \left\{ \begin{array}{ll} \{u \in S_q \mid \exists u' \in S_q . u \xrightarrow{\alpha} u' \ \&\ u' \times p_i \in [\![B[\sigma]]\!]_{q\times p}\ \rho\} & \mathit{if}\ \beta = * \\ \emptyset & \mathit{if}\ \beta \neq * \end{array} \right. \\
& \text{by definition} \\
= & \left\{ \begin{array}{ll} [\![\langle\alpha\rangle(B/p_i)]\!]_q\ \rho\} & \mathit{if}\ \beta = * \\ \emptyset & \mathit{if}\ \beta \neq * \end{array} \right. \\
& \text{by the induction hypothesis} \\
= & [\![A/p_i]\!]_q\ \rho \\
& \text{by definition.}
\end{array}
$$

The missing cases are all similar to the last case considered.

□

Appendix B

Proofs of Theorems of Chapter 4

B.1 Adequacy for ω-Regular Expressions

In this appendix we prove an adequacy result for ω-regular expressions. Denote by Act^∞ the set of all finite and infinite sequences over Act. Given a transition system T, let $ActSeq_s \subseteq Act^\infty$ be the set of finite and infinite sequences of actions out of s generated by the transition relation, i.e.

$$\begin{aligned} ActSeq_s &= \{a_0 \dots a_n \mid n \in \omega, \exists s_0, \dots, s_{n+1}.\ s = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_{n+1}\} \\ &\quad \cup \{a_0 \dots a_n \dots \mid \exists \{s_i\}_{i\in\omega}. s = s_0, \forall i \in \omega.\ s_i \xrightarrow{a_i} s_{i+1}\} \end{aligned}$$

(Thus $ActSeq_s$ is the set of all prefixes of the action part of the sequences of R_s, where R_s is the set of maximal runs defined on page 97 in definition 4.2.) For any set $U \subseteq Act^\infty$ let $\mathit{fin}(U) \subseteq U$ be the finite sequences of U and $\mathit{inf}(U) \subseteq U$ the infinite sequences s.t $U = \mathit{fin}(U) \cup \mathit{inf}(U)$.

Theorem B.1 (Adequacy for ω-regular expressions) *Assume A is a closed assertion, T a transition system with states S and R an ω-regular* *expression. For all* $s \in S$,

$$s \models \langle R \rangle A$$

$$\textit{iff}$$

$$\exists \delta \in \mathit{fin}(\mathit{ActSeq}_s \cap [\![R]\!]), s' \in S.\ s \xrightarrow{\delta} s' \ \&\ s' \models A \tag{B.1}$$

$$\textit{or}$$

$$\mathit{inf}(\mathit{ActSeq}_s \cap [\![R]\!]) \neq \emptyset \tag{B.2}$$

Proof: The proof is by structural induction on R. The cases for $R \equiv a$ and $R \equiv *$ are immediate.

$R \equiv R_0 R_1$.

$$\begin{array}{lll} s \models \langle R_0 R_1 \rangle A & \text{iff} & s \models \langle R_0 \rangle \langle R_1 \rangle A \qquad\qquad\qquad \text{by def.} \\ & \text{iff} & \exists \delta \in \mathit{fin}(\mathit{ActSeq}_s \cap [\![R_0]\!]), \\ & & \quad s' \in S.\ s \xrightarrow{\delta} s' \ \&\ s' \models \langle R_1 \rangle A \quad (^*) \\ & & \text{or} \\ & & \mathit{inf}(\mathit{ActSeq}_s \cap [\![(R_0]\!]) \neq \emptyset \qquad \text{by i.h.}(^{**}) \end{array}$$

Now, observing that $\mathit{inf}([\![R_0]\!]) \subseteq \mathit{inf}([\![R_0 R_1]\!])$ we see that $(^{**})$ implies (47). Similarly, if (*) holds we can apply the induction hypothesis again to get

$$\begin{array}{ll} \exists \delta' \in \mathit{fin}(\mathit{ActSeq}_{s'} \cap [\![R_1]\!]), s'' \in S.\ s' \xrightarrow{\delta} s'' \ \&\ s'' \models A & (^{***}) \\ \text{or} & \\ \mathit{inf}(\mathit{ActSeq}_{s'} \cap [\![(R_1]\!]) \neq \emptyset & (^{****}) \end{array}$$

Hence $(^*)$ and $(^{****})$ implies (B.2), and $(^*)$ and $(^{***})$ implies (B.1) using the sequence $\delta\delta'$. This completes the only-if-direction. For the if-direction we notice that (B.1) implies $(^*)$ and $(^{***})$ by definition of concatenation, and that (B.2) implies $(^*)$ and $(^{****})$ or $(^{**})$.

$R \equiv R_0 \cup R_1$. Analogously.

$R \equiv R_0^*$. If-direction. First suppose (B.2) holds. Then

$$\mathit{inf}(\mathit{ActSeq}_s \cap [\![R_0^*]\!]) \neq \emptyset$$

$$\begin{array}{ll}
\Rightarrow \exists n \in \omega, r_1, \ldots, r_n \in \mathit{fin} & \\
\quad ([\![R_0]\!]), r \in \mathit{inf}([\![R_0]\!]).r_1 \ldots r_n \in \mathit{ActSeq}_s & \\
\Rightarrow s \models \langle R_0\rangle^{n+1}B & \text{for any } B \text{ by i.h.} \\
\Rightarrow s \models \langle R_0\rangle^{n+1}\mu X.\langle R_0\rangle X & \\
\Rightarrow s \models \mu X.\langle R_0\rangle X \wedge A & \text{by unfolding and weakening} \\
\Rightarrow s \models \langle R_0^*\rangle A & \text{by def.}
\end{array}$$

Now, suppose (B.1) holds. Then

$$\begin{array}{ll}
\exists \delta \in \mathit{fin}(\mathit{ActSeq}_s \cap [\![R_0^*]\!]).s' \in S.\ s \xrightarrow{\delta} s' \ \&\ s' \models A & \\
\quad \Rightarrow \exists n \in \omega,\ r_1, \ldots, r_n \in \mathit{fin}([\![R_0]\!]), s' \in S & \\
\qquad\qquad \delta = r_1 \ldots r_n \in \mathit{ActSeq}_s \ \&\ s \xrightarrow{\delta} s' \ \&\ s' \models A & \\
\quad \Rightarrow s \models \langle R_0\rangle^n A & \text{by i.h.} \\
\quad \Rightarrow s \models \mu X.\langle R_0\rangle X \vee A & \text{by unfolding} \\
\quad \Rightarrow s \models \langle R_0^*\rangle A & \text{by def.}
\end{array}$$

Only-if-direction. First, recall that by definition $[\![\langle R_0^*\rangle A]\!]\rho = \mu U.f(U)$ where

$$f(U) = [\![\langle R_0^*\rangle X \vee A]\!]\rho[U/X].$$

Define M to be the set of states s satisfying (B.1) or (B.2), i.e.

$$M = \{s \mid \begin{array}{l} \exists\delta \in \mathit{inf}(\mathit{ActSeq}_s \cap [\![R_0]\!]), s' \in S.\ s\ \ s' \ \&\ s' \models A \text{ or} \\ \mathit{inf}(\mathit{ActSeq}_s \cap [\![R_0]\!]) \neq \emptyset \end{array} \}.$$

It can be argued that M is a pre-fixed point of f, i.e. $f(M) \subseteq M$ and therefore $\mu U.f(U) \subseteq M$ from which it follows that if $s \in \mu U.f(U)$ then $s \in M$. This is done along the lines of the proof of lemma 4.1.

$R \equiv R_0^\omega$. First notice that by definition $\mathit{fin}([\![R_0^\omega]\!]) = \emptyset$.

For the if-direction we have for each infinite sequence $\delta \in \mathit{ActSeq}_s \cap [\![R_0^\omega]\!]$ that

$$\begin{array}{c}
\exists n \in \omega, r_1, \ldots, r_n \in \mathit{fin}([\![R_0]\!]), r \in \mathit{inf}([\![R_0]\!]).\delta = r_1 \ldots r_n \\
\text{or} \\
\exists r_1, \ldots, r_n, \ldots \in \mathit{fin}([\![R_0]\!]).\delta = r_1 \ldots r_n \ldots
\end{array}$$

Using this it is not hard to see that $ActSeq_s \cap [\![R_0^\omega]\!]$ is a post-fixed point for f, i.e. $f(U) \supseteq U$, where $f(U) = [\![\langle R_0 \rangle X \vee A]\!]\rho[U/X]$ and hence as $\nu U.f(U)$ is the maximum post-fixed point we have that $ActSeq_s \cap [\![R_0^\omega]\!] \subseteq \nu U.f(U) = [\![\langle R_0^\omega \rangle A]\!]$ by definition.

For the only-if-direction we assume that $s \models \langle R_0^\omega \rangle A$. We prove by mathematical induction that for all $n \in \omega$

$$\exists s' \in S, \delta_1, \dots, \delta_n \in \mathit{fin}([\![R_0]\!]).\ s \overset{\delta_1 \dots \delta_n}{\rightarrow} s' \ \&\ s' \models \nu X.\langle R_0 \rangle X \tag{B.3}$$

$$\text{or}$$

$$\begin{aligned} &\exists k \in \omega, s' \in S, \delta_1, \dots, \delta_k \in \mathit{fin}([\![R_0]\!]), \delta \in \mathit{inf}([\![R_0]\!]). \\ &s \overset{\delta_1 \dots \delta_k}{\rightarrow} s' \ \&\ \delta \in \mathit{inf}(ActSeq_{s'}) \end{aligned} \tag{B.4}$$

The base case $n = 0$ is trivial. For the inductive step we assume that (B.3) or (B.4) holds. If (B.4) holds the inductive step is trivially valid. Hence, assume that (B.3) holds. Now, as $s' \models \nu X.\langle R_0 \rangle X$ hence by unfolding $s' \models \langle R_0 \rangle \nu X.\langle R_0 \rangle X$, we have by the induction hypothesis of the structural induction that either $\exists \delta \in \mathit{fin}(ActSeq_{s'} \cap [\![R_0]\!]), s'' \in S.s'' \models A$ or $\mathit{inf}(ActSeq_{s'} \cap [\![R_0]\!]) \neq \emptyset$. In both cases the inductive step of the mathematical induction is easily completed. □

B.2 Correctness of Embedding of CTL$^\bullet$

We prove by structural induction on CTL$^\bullet$ formulae Ψ that

$$[\![\Psi]\!]_{T,V} = [\![I(\Psi)]\!]_{T,V}$$

for any transition system T with valuation V. In fact we only consider the case where $\Psi \equiv \exists \Box \Diamond \Psi'$. The case $\Psi \equiv \forall \Diamond \Box \Psi'$ is dual and the remaining cases are quite similar to the proof of adequacy for ω-regular expressions in appendix B.1.

Proof (Lemma 4.2): Hence, we assume that $\Psi \equiv \exists \Box \Diamond \Psi'$. Writing out from the definition of $[\![]\!]$ we get

$$\begin{array}{lll} s \in [\![\exists\Box\Diamond\Psi']\!] & \Leftrightarrow & \exists\delta \in R_s.\ \delta \in\parallel \Box\Diamond\Psi' \parallel \\ & & \quad \text{by definition} \\ & \Leftrightarrow & \exists\delta \in R_s.\forall k.|\delta| < k \text{ or } (\exists l.\ k \leq l \leq |\delta|\ \&\ \delta^l \in\parallel \Psi' \parallel) \\ & & \quad \text{by definition} \\ & \Leftrightarrow & \exists\delta \in R_s.\forall k.|\delta| < k \text{ or } \exists l.\ k \leq l \leq |\delta|\ \&\ \delta_l \in [\![\Psi']\!]) \\ & & \quad \text{as } \Psi' \text{ is a state formula} \\ & \Leftrightarrow & \exists\delta \in R_s.\forall k.|\delta| < k \text{ or } (\exists l.\ k \leq l \leq |\delta|\ \&\ \delta_l \in [\![I(\Psi')]\!]) \\ & & \quad \text{by the induction hypothesis} \end{array}$$

We first consider the case of an infinite sequence. Observe, that if we have an l with $k \leq l$ and $\delta_l \in [\![I(\Psi')]\!]$ then for all k' with $k \leq k' \leq l$ we have that there exists an l' such that $k' \leq l'$ and $\delta_{l'} \in [\![I(\Psi')]\!]$ namely l. Hence in the case of δ being infinite we have the equivalent formulation

$$\exists\delta \in \mathit{inf}(R_s).\ \exists\{l_i\}_{i\in\omega}.\ l_0 < l_1 < \ldots < l_i < \ldots\ \&\ \forall i.\delta_{l_i} \in [\![I(\Psi)]\!]$$

and in the finite case we have

$$\exists\delta \in \mathit{inf}(R_s).\ \exists k.\ \exists\{l_i\}_{i\leq k}.\ l_0 < l_1 < \ldots < l_k\ \&\ l_k = |\delta|\ \&\ \forall i \leq k.\delta_{l_i} \in [\![I(\Psi)]\!]$$

which is equivalent to[1]

$$\begin{array}{l} \exists s_0, s'_0, s_1, s'_1, \ldots, s_i, s'_i, \ldots \in \omega.\ s = s_0, \\ \quad \forall i.s_i \overset{\cdot *}{\longrightarrow} s'_i \overset{\cdot}{\longrightarrow} s_{i+1}\ \&\ s'_i \in [\![I(\Psi')]\!] \\ \text{or} \\ \exists k.\ \exists s_0, s'_0, s_1, s'_1, \ldots, s_k, s'_k.\ s = s_0, \\ \quad s_k \overset{\cdot *}{\longrightarrow} s'_k\ \&\ s'_k \not\rightarrow\ \&\ s'_k \in \delta \in [\![I(\Psi')]\!] \\ \quad \forall i < k.s_i \overset{\cdot *}{\longrightarrow} s'_i \overset{\cdot}{\longrightarrow} s_{i+1}\ \&\ s'_i \in [\![I(\Psi')]\!] \end{array} \tag{B.5}$$

We will argue that this is equivalent to

$$s \models \nu X.(\cdot^*)(\langle\cdot\rangle' X \wedge I(\Psi')). \tag{B.6}$$

[1]It is not hard to realize that for finite sequences this is equivalent to saying that the final state must be in $I(\Psi')$, but this apparently simpler formulation will not be very helpful in getting a compact μ-calculus formula.

Now, expanding the $\langle\cdot^*\rangle$ we get

$$s \models \nu X.\mu Y.\langle\cdot\rangle' Y \vee (\langle\cdot\rangle' X \wedge I(\Psi')).$$

which is what we wanted to prove.
(B.5) implies (B.6). Let M be the set of states s satisfying (B.5). Notice, that if $s \in M$ then there exists an $s' \in [\![I(\Psi')]\!]$ such that either $s' \not\to$ or there exists an s'' such that $s' \overset{\cdot}{\to} s''$ and $s'' \in M$. Using this observation we will prove that M is a post-fixed point of f, i.e. $f(M) \supseteq M$ where

$$f(U) = [\![\langle\cdot^*\rangle(\langle\cdot\rangle' X \wedge I(\Psi'))]\!] \rho[U/X]$$

and hence $M \subseteq \nu U.f(U) = [\![\nu X.\langle\cdot^*\rangle(\langle\cdot\rangle' X \wedge I(\Psi'))]\!] \rho$. This is straightforward: picking an $s \in M$ then by the previous discussion there exists an s' such that $s \overset{\cdot *}{\to} s'$ and $s' \in [\![\langle\cdot\rangle' X \wedge I(\Psi')]\!] \rho[M/X]$. Hence $s \in f(M)$.
(B.6) implies (B.5). Assume s satisfies (B.6). By mathematical induction on n we show that for all $n \in \omega$

$$\begin{array}{l}
\exists s_0, s'_0, s_1, s'_1, \ldots, s_n, s'_n. \\
\qquad s = s_0, s_n \overset{\cdot *}{\to} s'_n, s'_n \models \nu X.\langle\cdot^*\rangle(\langle\cdot\rangle' X \wedge I(\Psi')), \\
\qquad \forall i < n.\ s_i \overset{\cdot *}{\to} s'_i \overset{\cdot *}{\to} s_{i+1} \ \&\ s'_i \in [\![I(\Psi')]\!] \\
\text{or} \\
\exists k.\ \exists s_0, s'_0, s_1, s'_1, \ldots, s_k, s'_k. \\
\qquad s = s_0, s_k \overset{\cdot *}{\to} s'_k \ \&\ s'_k \in [\![I(\Psi')]\!] \ \&\ s'_k \not\to, \\
\qquad \forall i < k. s_i \overset{\cdot *}{\to} s'_i \overset{\cdot}{\to} s_{i+1} \ \&\ s'_i \in [\![I(\Psi')]\!]
\end{array}$$

The base case is trivial (taking $s'_0 = s_0$). The inductive step follows by unfolding the maximal fixed-point splitting into the two situations where there exists an s_{n+1} such that $s'_n \overset{\cdot}{\to} s_{n+1}$ and when $s'_n \not\to$.

Hence we have proven

$$s \in [\![\exists\Box\Diamond\Psi']\!] \Leftrightarrow s \models \nu X.\mu Y.\langle\cdot\rangle Y \vee (\langle\cdot\rangle' X \wedge I(\Psi'))$$

□

Index