Angelika Mader

Verification of Modal Properties Using Boolean Equation Systems

EDITION VERSAL 8

Angelika Mader

Verification of Modal Properties Using Boolean Equation Systems

Dieter Bertz Verlag

Die Deutsche Bibliothek – CIP Einheitsaufnahme

Mader, Angelika:
Verification of Modal Properties Using Boolean Equation Systems / Angelika Mader. – Berlin : Bertz, 1997
(Edition Versal ; Bd. 8)
Zugl.: München, Techn.-Univ., Diss., 1997
ISBN 3-929470-5n-x
NE: GT

Abstract

The thesis is concerned with verification of properties of concurrent systems expressed in the modal μ-calculus. This approach is called model-checking.

The modal μ-calculus contains fixpoint-operators which give great expressive power. In order to treat the model-checking problem algebraically we introduce fixpoint-equation systems as an extension of expressions containing least and greatest fixpoints. Fixpoint-equation systems interpreted over the Boolean lattice or an infinite product of Boolean lattices are called Boolean equation systems. Model checking for systems with finite state spaces is shown to be equivalent to solving finite Boolean equation systems. We discuss existing model-checking algorithms from the perspective of Boolean equation systems and

present a new algorithm, similar to Gauß elimination for linear equation systems.

As an application we investigate algorithms solving the problem of mutual exclusion, construct formulae for liveness properties and verify them with an implementation of the Gauß elimination algorithm.

Model-checking in the modal μ-calculus has already been treated in automata theory and game theory. We are able to show a new equivalence to an automata-theoretic problem by going via Boolean equation systems. There existed a reduction of model-checking to a game theoretic problem. Using Boolean equation systems we can prove the equivalence.

For the case of infinite state spaces we also show that model-checking is equivalent to solving infinite Boolean equation systems. Additionally, we present an algorithm, similar to the Gauß elimination algorithm for the finite case.

Acknowledgement

In the first place, I would like to thank my supervisor Wilfried Brauer for the constant support of my all activities here, the comfortable environment he provided for us, and his liberal attitudes, which made a carefree and concentrated way of working possible.

I am indebted to my proof readers Julian Bradfield, Ed Brinksma and Ekkart Kindler. Their comments and careful criticism were of great help for me in finding out what I was doing, in improving my work, and, what is perhaps even more valuable, they increased the fun I had when writing up. Thanks also to Christine Röckl who gave useful comments on parts of the thesis.

I am very grateful to people in Edinburgh: Colin Stirling for being referee and for mostly illuminating discussions, Julian Bradfield for his friendship and pleasant cooperation, Kyriakos Kalorkoti, who shared his insight in Boolean equation systems with me, Perdita Stevens for her theoretical and practical help concerning games, and Roope Kaivola for clarifying automata-theoretic concepts. The impressive scientific atmosphere and the great variety of single malts contributed enormously to my enjoyment of my visits to Edinburgh.

Ed Brinksma and Peter Rossmanith supported me in finding an exponential example for my version of the Gauß algorithm and delivered me from a long-term passion. In Frank Wallner I found a colleague who was not afraid of fixpoints and discussions with him were a great pleasure to me. His never ending engagement in implementing the Gauß algorithm was extremly helpful to me. Furthermore, I would like to acknowledge all people of the group here, and those who left to Berlin. I spent a good time with them. Particularly, I am missing the sharp tongue of Dominik Gomm.

Many thanks go to Barbara Roemer who gave valuable hints concerning layout.

I thank Fa. Siemens, ZFE, and the Deutsche Forschungsgemeinschaft (Sonderforschungsbereich 342) for funding my position at the TU.

Without Gerhard's support I could not have done this work and many other things at the same time while having a child. Many thanks also

to David for consistently relativizing all ups and downs concerning my work and for all the nights he slept through.
¿From Ed, my parents, family and friends I received valuable support of various kinds during all the time, for which I owe them great thanks.

Contents

Chapter 1

Introduction.

1.1 General introduction.

When it is necessary that a thing should be, it is possible that it should be. ... Yet, from the proposition 'it may be' it follows that it is not impossible, and from that it follows that it is not necessary; it comes about therefore that the thing which must necessarily be need not be; which is absurd....

Aristotle, Hermeneia[1]

The beginning of modal logic dates back to Aristotle who was already concerned with the logic of necessity and possibility. Later, the Megarian Stoics also dealt with modal logics, introducing a time based interpretation: possible is just what either is or will be; a thing is necessary only if it is now true and always will be true.

Leibniz gave a semantic model for logics including the modalities 'necessarily' and 'possibly': he assumed a set of worlds and defined a proposition being necessarily true if it is true in all worlds, and being possibly true if there exists some world where it is true. In addition,

[1]see [Boc70]

he proved that we live in the best of all possible worlds.

Formal mathematical treatment of modal logic started in this century. Nowadays philosophers, logicians, linguists and computer scientists share an interest in the subject, and various systems of modal logic have been developed.

In further development, more structure was given to the model of worlds. When deciding whether some proposition p is necessary in one world only a specified set of worlds may be relevant, which need not include every world in the model. This feature is represented by an accessibility relation between worlds, and p is necessarily true in one world means that p being true in all worlds accessible from the current one. Temporal logic is then defined as a modal logic, where accessibility between worlds represents time passing by, and the worlds are ordered linearly in time.

In computer science modal and temporal logic play a role in the verification of systems. Here, the task is to show that a system meets its specification which may consist of set of properties expressed as formulae of a logic.

Models for modal logic are Kripke structures, also called transition systems. They consist of a set of states (representing the worlds) and transitions between the states (the accessibility relation). A transition system models the different states an arbitrary system can enter, and actions leading from one state to another. A state can represent e.g. the content of a memory, the value of a program counter, a state of a pinball machine. Transitions may carry a label identifying an action (write 1 to a memory cell, shoot the pinball) or modelling just the on-going of a system as time passes. The latter case provides a model for temporal logic.

Propositions are about states or paths of a model, e.g. for the pinball machine initially the only possible action is to insert a coin; there exists a run of the pinball machine, where I always get a free game, or, if I hit the pinball machine infinitely often then the ball will eventually roll down.

In the first period, objects of verification were sequential and imperative programs. Proving correctness for a program was to show that given a specified input the program would terminate and produce a specified output. The works of Floyd [Flo67], Manna and Pnueli [MP69], Park [Par70] and Hoare [Hoa69] were important developments in this context.

The first modal logics for verification were dynamic logics introduced by Pratt [Pra76], and mostly used in the propositional version. Propositional Dynamic Logic (PDL) is built up from Propositional Logic extended by the modalities $\langle \alpha \rangle$, where a program α is a regular expression over a set of atomic programs. The formula $\langle \alpha \rangle p$ is true at a state, where it is possible for the program α to execute and result in a state satisfying p. Various restrictions and extensions of PDL have been investigated. The most famous ones are PDL with test progams, and PDL-Δ [Str81] where an infinite loop-operator is added to program expressions.

The introduction of concurrency caused change concerning the characteristics of programs: termination and results produced were not longer necessary features, but on-going and interaction with an environment became relevant. Pnueli called them "reactive systems". Proving correctness here required more expressive logics. Manna and Pnueli [MP83] found that temporal logic is suitable in this context. They applied a proof-theoretic style of verification: for a given program they derived a set of temporal properties and showed that the specifying property was a consequence of this set (or was not).

Clarke, Emerson and Sistla [CES86], and others started with a new approach, called model-checking. Here, verification for finite state systems is performed automatically and, in contrast to deriving a proof, an algorithm receiving a formula and a model as input gives the result **true** or **false**. The temporal logic they used is Computation Tree Logic (CTL). In this logic a number of useful properties is expressible (e.g. if the pinball is shot then it will eventually roll down again), but some relevant properties are not (e.g. if infinitely often a player hits the pinball machine then infinitely often it will be in the state "tilt").

In subsequent development, work was centered mainly on two issues: the tackling of the size of problems and the definition of more expressive logics. Of course, the problems are not mutually independent of each other; roughly, the more expressive a logic is, the more complex is verification and the smaller is the size of solvable problems.

An extension of CTL that can express the "tilt"-property cited above is called CTL*. For this temporal logic Emerson and Lei [EL86] presented a model-checking algorithm.

Meanwhile also various extensions of CTL and CTL* have been investigated which are more expressive, but still simple enough for model-checking.

Kozen [Koz83] introduced a very powerful logic, subsuming all other modal and temporal logics mentioned above: the modal μ-calculus. In addition to Propositional Logic it contains the modalities $[a]$ and $\langle a \rangle$ and the fixpoint operators μ and ν. The modalities allow one to express properties for one next-step, while by means of least (and dually greatest) fixpoint immediately properties over finite and infinite paths can be modelled. The beauty of this logic lies in its expressiveness in combination with its simplicity. The first model-checking algorithm for the modal μ-calculus was developed by Emerson and Lei [EL86]. However, the complexity of their algorithm is higher than that for less expressive logics such as CTL: it is of exponential complexity in the size of the formula in contrast to polynomial complexity of model-checking algorithms for CTL. Since then a number of algorithms for the modal μ-calculus have been suggested, yet there has not been any essential improvement concerning complexity so far, and the lower bounds for the complexity of this problem have not yet been detected.

Concerning the size of problems considerable progress has been achieved by so-called "symbolic model-checking". For earlier algorithms the model, a transition system, had to be represented explicitly. In a new approach for CTL model-checking Burch, Clarke and McMillan [BCM$^+$92] chose Binary Decision Diagrams (BDDs) as data-structure, which allowed a very compact encoding of transition systems, and the size of problems that could be treated grew enormously.

However, the size of the transition systems is still the most limiting problem in this area. Especially for concurrent systems the so-called "state space explosion" makes verification difficult or even impossible. Reduction techniques for transition systems have been investigated including e.g. abstractions and symmetries, which relativize the purely automatic approach and reintroduce elements of proof to model-checking.

The method of model-checking described above is "global" in the sense that the algorithms traverse the whole state space and determine the set of all states satisfying a property. Usually, we are interested in whether a property holds of paths starting from the initial state of a system. Showing its correctness may not require the whole state space, or not even the set of reachable states, but a (hopefully small) subset of it. Algorithms based on this idea are called "local". A local model-checking algorithm for the modal μ-calculus was first introduced by Stirling and Walker [SW89] in form of a tableau system.

In the case of general infinite state-spaces there is no hope for fully automatic methods. However, proving properties with computer assistance is a possibility. Bradfield and Stirling [BS90, Bra92] developed a tableau method allowing computer-aided verification for formulae of the modal μ-calculus. Other work has been done in this area for infinite models defined e.g. by some Petri-net classes [EN94], or context-free grammars.

Also in this work, we are concerned with model-checking for the modal μ-calculus. The approach is an algebraic one: model-checking is transformed to the problem of solving a class of equation systems, called Boolean equation systems. In fact, we can show that the two problems are equivalent, for the case of finite systems as well as for infinite ones. Based on this equivalence we discuss model-checking algorithms and show their relations to other techniques, in automata theory and game theory. The following section goes on to outline this in more detail.

1.2 Synopsis.

In the beginning we give a brief collection of relevant definitions and facts from lattice theory and the fixpoint theorems which are structures and facts basic for the whole work.

In computer science mainly least fixpoints have been considered. Propositions for expressions containing least and greatest fixpoint operators do not go beyond duality arguments so far. Chapter 3 contains the first contribution of this work: an introduction of fixpoint-equation systems as a generalization of nested and alternating fixpoint-expressions. It entails an extensive collection of properties of fixpoint-equation systems. The difference between more traditional equation systems and fixpoint-equation systems consists of the additional structure given to the latter: there is an order defined on the equations and each equation is equipped with a minimality or maximality condition. Because of this structure known results for solutions of equation systems over lattices do not apply for the fixpoint-equation systems. In this work fixpoint-equation systems will be interpreted over the Boolean lattice for finite state space model-checking as well as over an infinite product of Boolean lattices for model-checking of infinite state spaces. Section 3.2 contains definitions and properties for the finite case, extending properties for fixpoint-equation systems over arbitrary lattices. The infinite case will be treated in chapter 9. Fixpoint-equation systems interpreted in this way are called Boolean equation systems and infinite Boolean equation systems.

Chapter 4 contains an introduction to the modal μ-calculus, including syntax, semantics, basic notations and facts.

The main point of chapter 5 is the equivalence of the model-checking problem for finite state spaces and the problem of solving Boolean equation systems. Reductions to Boolean equation systems for the case of non-alternating μ-calculus expressions have already been treated by other people. The extension to the general case could be done by the well-known fixpoint theorems. Here, in section 5, we give a reduction applying directly to the general case. The size of a Boolean equation system derived is linear in the size of the model and linear in the size of the formula. In order to get a representation of a Boolean equation system linear in the size of the original model-checking problem a a simple form for equations has to be defined following known techniques. Section 5.2 shows the reduction in the other direction. Given a Boolean equation system, we construct a formula of the modal μ-calculus and a model, such that the Boolean equation system has the solution true iff the the model satisfies the formula. The size of the model is quadratic in the size of the Boolean equation system, the size of the formula is linear.

Chapter 6 deals with methods for solving Boolean equation systems, local as well as global ones. We start with a discussion of the problem, relating it to the "classical" version of Boolean equation systems without order on the equations and without side conditions for fixpoints. The known methods solving the model-checking problem are the approximation technique and a tableau method. We interprete them on Boolean equation systems. In addition we present a new solving technique for Boolean equation systems which is similar to Gauß elimination for linear equation systems. It leads to both, a local and a global algorithm. The last section contains a simple proof for solving Boolean equation systems being in NP $\cap$ co-NP, and according to the equivalence results also the model-checking problem is contained in this class, which is a known result.

Examples for application are presented in chapter 7. Here, we focus on composing and proving different liveness properties for Peterson's algorithm solving the problem of mutual exclusion. These properties provide non-trivial examples for μ-calculus formulae. They are verified with an implementation of Gauß elimination for Boolean equation systems.

The model-checking problem for the modal μ-calculus has been treated in other frameworks: there exist reductions to problems in automata- and game-theory. In the first case all automata derived are tree-automata. In section 8.1 we show the equivalence of model-checking and the non-emptiness-problem of alternating automata on infinite words over a single-letter alphabet with a parity acceptance condition. The model-checking problem has also been reduced to model-checking games. In section 8.2, we show the equivalence of deciding whether a player has a winning strategy for a game and solving a Boolean equation system. The reduction of Boolean equation systems to model-checking gives immediately a reduction from a model-checking game to a model-checking problem, which has been an open question.

So far we have only been considering finite state spaces. In chapter 9, the theory of Boolean equation systems is extended to the infinite case. Boolean equation systems as they are used here are derived from fixpoint-equation systems interpreted over a (possibly infinite) product of Boolean lattices. The equivalence of infinite Boolean equation systems and the model-checking problem for infinite state spaces is proved by reductions in both directions. These results are only useful when having a finite representation of the problem which is given by set based equation systems. We present an elimination method using ideas from Gauß elimination for the finite case and from the tableau method of Bradfield and Stirling. It solves set based equation systems and also the model-checking problem for the infinite case. Small examples demonstrate the technique.

The thesis ends with concluding remarks putting our results in a general framework.

Chapter 2

Basics.

2.1 Orders and lattices.

The basic structure in this work are lattices; formulae of modal logic with implication order form a lattice, the powerset of a state space is complete lattice. The semantic of a formula of modal logic can be interpreted as an order preserving function between two lattices. The fixpoint operators of modal logic have to be defined via continuous functions. Therefore, we collect here the relevant definitions and facts. A detailed introduction into lattices and orders can be found [DP90].

Definition 2.1 A binary relation $\leq$ on a set P is a **partial order** if for all $x, y, z \in P$:

(**reflexivity**)	$x \leq x$
(**antisymmetry**)	$x \leq y$ and $y \leq x$ imply $x = y$
(**transitivity**)	$x \leq y$ and $y \leq z$ imply $x \leq z$

A set equipped with a partial order is called an **ordered set**.

Definition 2.2 Given an ordered set P and a subset Q of P the **greatest element** of Q is $a \in Q$ if $a \geq x$ for all $x \in Q$. Dually, the **least element** of Q is $a \in Q$ if $a \leq x$ for all $x \in Q$.

Definition 2.3 Let P be an ordered set. The greatest element of P, if it exists, is called the **top element** of P and written $\top$. Dually, the least element of P, if it exists, is called the **bottom element** of P and written $\bot$.

Proposition 2.4 Given an ordered set P any subset $Q \subseteq P$ is an ordered set.

Proposition 2.5 Let $(P_1, \leq_1), \ldots, (P_n, \leq_n)$ be ordered sets. Their product $P_1 \times \ldots \times P_n$ can be equipped with a partial order by **point-wise** definition: $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n)$ iff $x_i \leq_i y_i$ for $1 \leq i \leq n$.

Definition 2.6 Let P and Q be ordered sets. The **set of functions** from P to Q is denoted by $(P \to Q)$. For each function $f \in (P \to Q)$ the **domain** is P and the **codomain** is Q.

A function $f \in (P \to Q)$ is **monotone**, if for all $p_1, p_2 \in P$ with $p_1 \leq p_2$ it is the case that $f(p_1) \leq f(p_2)$.

The set of all monotone functions is denoted by $\langle P \to Q \rangle$.

On the set of functions $(P \to Q)$ an order is inherited from the order on their codomain Q: Let $f, g \in (P \to Q)$. Then $f \leq g$ if $f(a) \leq g(a)$ for all $a \in A$.

Definition 2.7 Let P be an ordered set and S be a subset of P. Then $x \in P$ is an **upper bound** of S, if $s \leq x$ for all $s \in S$. Dually $x \in P$ is a **lower bound** of S, if $x \leq s$ for all $s \in S$.
All upper bounds of S are collected in a set $\uparrow S$, the lower bounds in a set $\downarrow S$. The least element of $\uparrow S$, if it exists, is called **least upper bound** of S, and denoted by $\bigvee S$. The greatest element of $\downarrow S$ if it exists, is called **greatest lower bound** of S, and denoted by $\bigwedge S$. They are also called the **supremum** and **infimum** of S.

Notation: For the supremum $\bigvee\{x, y\}$ we write $x \vee y$, and $x \wedge y$ for the infimum $\bigwedge\{x, y\}$. When speaking about powersets we will use $\bigcup$ and $\bigcap$ instead of $\bigvee$ and $\bigwedge$, and $\cup$ and $\cap$ instead of $\vee$ and $\wedge$.

Definition 2.8 Let P be a non-empty ordered set. P is a **lattice**, if $x \vee y$ and $x \wedge y$ exist for all $x, y \in P$. P is a **complete lattice**, if $\bigvee S$ and $\bigwedge S$ exist for all subsets $S \subseteq P$.

Proposition 2.9

(1) In a lattice $\bigvee S$ and $\bigwedge S$ exist for all finite subsets $S \subseteq P$.

(2) Every finite lattice is complete.

(3) In a complete lattice the bottom element $\perp$ and the top element $\top$ exist.

(4) For any set X its powerset $\mathfrak{P}(X)$ equipped with the set inclusion order $\subseteq$ is a complete lattice.

(5) If P and Q are (complete) lattices then also the sets of functions $(P \to Q)$ and $\langle P \to Q \rangle$ are (complete) lattices. Supremum and infimum are obtained pointwise.

In most cases we think of functions as represented by function expressions. These are built up by variables X from a set of variables $\mathcal{X}$, the operations supremum $\vee$ and infimum $\wedge$, and a set of operators $\{Op_1^{(k_1)}, \ldots, Op_n^{(k_n)}\}$ for some $n \in I\!N$, where k_i denotes the arity of the operator $Op_i^{(k_i)}$.

$$f ::= X \mid f \vee f \mid f \wedge f \mid Op_i^{(k_i)}(f, \ldots, f)$$

Proposition 2.10 Let P and Q be ordered sets, $f : P \to Q$ a monotone function, and $S \subseteq P$ such that $\bigvee S$ and $\bigwedge S$ exist in P, and $\bigvee f(S)$, $\bigwedge f(S)$ exist in Q. Then $f(\bigvee S) \geq \bigvee f(S)$ and $f(\bigwedge S) \leq \bigvee f(S)$.

Proposition 2.11 Products of complete lattices equipped with a partial order as in proposition 2.5 are complete lattices.

Definition 2.12 A non-empty subset S of an ordered set P is **directed**, if every finite subset F of S has an upper bound in S.

Definition 2.13 Let P and Q be complete lattices.
Then $f : P \to Q$ is **continuous** if for every directed set in P it is the case that $f(\bigvee D) = \bigvee f(D)$.

A function that preserves $\bot$, i.e. $f(\bot) = \bot$ is called **strict**.

Proposition 2.14 Let P and Q be complete lattices. Then every monotone function $f : P \to Q$ is also continuous.

Definition 2.15 Given a lattice P and a function $f : P \to P$. An element $x \in P$ is a **fixpoint** of f if $f(x) = x$.

2.2 Fixpoints and their properties.

This section is a collection of various properties of fixpoints which can be found in the literature. It starts with properties of simple fixpoints, both least and greatest. Then we look at the more general case where fixpoint operators of possibly different type are nested.

2.2.1 Simple fixpoints.

The very basic theorem comes from Tarski [Tar55] (see also [LNS82]). It guarantees the existence of a least and greatest fixpoint for a monotone function over a complete lattice.

Theorem 2.16 Let $(A, \leq)$ be a complete lattice, $f : A \to A$ a monotone function, and P the set of all fixpoints of f. Then P is not empty and the system $(P, \leq)$ is a complete lattice; in particular the least fixpoint is $\mu X.f(X) = \bigwedge\{a \in A \mid f(a) \leq a\}$ and the greatest fixpoint is $\nu X.f(X) = \bigvee\{a \in A \mid f(a) \geq a\}$.

We will use σ when referring to either μ or ν.
The next properties (for monotone f) can be found e.g. in [Koz83].

Proposition 2.17

(1) $f(\sigma X.f(X)) = \sigma X.f(X)$

(2) If $f(a) \leq a$ then $\mu X.f(X) \leq a$.

(3) If $f(a) \geq a$ then $\nu X.f(X) \geq a$.

(4) If $f(a) \leq g(a)$ for all $a \in A$ then $\sigma X.f(X) \leq \sigma X.g(X)$.

(5) If $f(a) = f(b)$ for all $a, b \in A$ then $\sigma X.f(X) = f(X)$.

(6) $\sigma X.f(X) = \sigma X.f(f(X))$

The following property is known as the reduction lemma, see for example [Koz83], [Win89].

Lemma 2.18 $a \leq \nu X.f(X)$ iff $a \leq f(\nu X.(f(X) \vee a))$

or, dually, $a \geq \mu X.f(X)$ iff $a \geq f(\mu X.(f(X) \wedge a))$.

Tarski's theorem shows the existence of a least and greatest fixpoint, but no constructive method to yield it. This is the subject of the next well-known theorem based on approximants. It is presented here in its general version, using transfinite iteration (see [LNS82]).

Definition 2.19 Let $(A, \leq)$ be a complete lattice and $f : A \rightarrow A$ a monotone function. Then $\sigma^\alpha X.f$ is an approximant term, where α is an ordinal. The approximant terms are defined by transfinite induction:

$$\begin{aligned}
\mu^0 X.f(X) &\overset{\text{def}}{=} \bot \\
\nu^0 X.f(X) &\overset{\text{def}}{=} \top \\
\sigma^{\alpha+1} X.f(X) &\overset{\text{def}}{=} f(\sigma^\alpha X.f(X)) \\
\mu^\lambda X.f(X) &\overset{\text{def}}{=} \bigvee_{\alpha<\lambda} \mu^\alpha X.f(X) \\
\nu^\lambda X.f(X) &\overset{\text{def}}{=} \bigwedge_{\alpha<\lambda} \mu^\alpha X.f(X)
\end{aligned}$$

where λ is a limit ordinal.

Proposition 2.20 For a complete lattice $(A, \leq)$ and a monotone function $f : A \to A$

$$\mu X.f(X) = \bigvee_{\alpha \in Ord} \mu^{\alpha} X.f(X)$$

$$\nu X.f(X) = \bigwedge_{\alpha \in Ord} \nu^{\alpha} X.f(X)$$

where Ord is the class of all ordinals.

Moreover there exists an ordinal α of cardinality less or equal to that of A such that for $\beta \geq \alpha$:

$$\mu X.f(X) = \mu^{\beta} X.f(X)$$

and, dually,

$$\nu X.f(X) = \nu^{\beta} X.f(X).$$

2.2.2 Nested fixpoints.

We now want to consider nested fixpoints, such as $\nu X.f(X, \mu Y.g(X, Y))$ where X and Y are variables over lattices $(A, \leq)$ and $(B, \leq)$, and f and g are monotone in both arguments. As a first step we will define the inner fixpoint $\mu Y.g(X, Y)$ as a function g' from A to B. We will abuse notation and do not introduce new names for f and g when their domains are interpreted in different ways. For technical reasons we assume from now on that there are not two different variables in a nested fixpoint expression having the same names.

Definition 2.21 Let $(A, \leq)$ and $(B, \leq)$ be complete lattices, g a monotone function on $A \times B$ to B. Then the least fixpoint with respect to B is a function from A to B

$$\mu Y.g(X, Y) \overset{\text{def}}{=} \bigwedge \{g' \in (A \to B) \mid g(X, g'(X)) \leq g'(X)\}$$

and the greatest fixpoint is

$$\nu Y.g(X, Y) \overset{\text{def}}{=} \bigvee \{g' \in (A \to B) \mid g(X, g'(X)) \geq g'(X)\}.$$

Proposition 2.22 The least (greatest) fixpoint of $g : A \times B \to B$ is a monotone function $g' : A \to B$ and it is the case that $g'(a) = \mu Y.g(a,Y)$ $(g'(a) = \nu Y.g(a,Y))$ for every $a \in A$, where $g(a,Y) : B \to A$ and $\sigma Y.g(b,Y)$ follows definition 2.16.

Proof: straightforward □

The monotonicity of g' implies that $f(X, g'(X))$ is a monotone function from A to A and its fixpoints are well defined according to definition 2.16. The application to arbitrary nesting of fixpoints works straightforwardly. In the remark below g' might be a vector of functions resulting from inner fixpoints and all domains could be interpreted as (possibly empty) products of complete lattices.

Remark 2.23 We want to point out, that there exist two basically different interpretations of the inner fixpoints which have consequences for algorithms calculating them. The first one is the more common one: g' as a function on A to B is defined pointwise, $g'(a) \overset{\text{def}}{=} \sigma Y.g(a,Y)$. For every argument $a \in A$ we get the simple function $g(a,Y)$ on B to B and the application of a fixpoint operator σY is well defined. This interpretation gives rise to the approximation based algorithms. Evaluation of g' at a is done by a simple approximation of $\sigma Y.g(a,Y)$ as in proposition 2.20.

The other interpretation focuses on the fact, that in some cases we can explicitly calculate the function g', not in a pointwise manner, but as a function expression with a free variable Y. Here the evaluation of $g'(a)$ consists of a simple function evaluation and not of an approximation.

Bekič's theorem [Bek84] for elimination of simultaneous fixpoints shows how a simultaneous fixpoint can be transformed to a nested fixpoint expression.

Theorem 2.24 Let $(A, \leq)$ and $(B, \leq)$ be complete lattices, $f : A \times B \to A$ and $g : A \times B \to B$ monotone functions.

Then $\mu(X,Y).(f(X,Y), g(X,Y)) = a, b$, where $a = \mu X.f(X, \mu Y.g(X,Y))$, and $b = \mu Y.g(a,Y)$.

Chapter 3

Fixpoint-equation systems.

We introduce fixpoint-equation systems extending the notion of nested fixpoint expressions. The intention of this chapter is to provide the technical basis for the rest of the work. Therefore, apart from definitions of syntax and semantics it contains an extensive collection of properties of fixpoint-equation systems. In the first section the general case of fixpoint-equation systems is investigated, where they are interpreted over arbitrary complete lattices. For the issue of this work the required domains are the Boolean lattice and a possibly infinite product of Boolean lattices. The second section focuses on the fixpoint-equation systems over the Boolean lattice, Boolean equation systems. For this case some definitions simplify and we get a number of further properties. Proofs of this chapter are shifted to the appendix.

3.1 Fixpoint - equation systems for complete lattices.

First syntax and semantics[1] are defined, then we give a translation from fixpoint expressions to fixpoint equation systems. The main part of this section contains an extensive collection of properties of fixpoint-equation systems.

In the following we consider sequences of functions $f_1, f_2, \ldots$ over a lattice $(A, \leq)$. Often, free variables will be substituted by the same values in each function. Instead of performing explicitly the substitution in each function we collect the values of the variables in a valuation, called **environment**. $\theta, \theta_1, \ldots$ will range over environments, where each θ is a function $\theta : \mathcal{X} \to A$.

A function f can be applied to an environment θ, and the result $f(\theta)$ is the value of the function f after substituting each free variable X of f by $\theta(X)$. By $\theta[X/a]$ we denote the environment that coincides with θ for all variables except X, i.e. $\theta(Y) = (\theta\,[X/a])(Y)$ for $Y \not\equiv X$, and $(\theta[X/a])(X) = a$. In the remainder $[X/a]$ has priority over all other operations, and $\theta[X/a]$ always stands for $(\theta[X/a])$.

The order on a lattice $(A, \leq)$ extends naturally to an order on environments over A (see Definiton 2.6). We have $\theta_1 \leq \theta_2$ iff for all variables $X \in \mathcal{X}$ it is the case that $\theta_1(X) \leq \theta_2(X)$. Thus the set of environments (for a fixed set of variables $\mathcal{X}$) forms a lattice. Obviously, the lattice operations $\vee$ and $\wedge$ can be applied also to environments when interpreting them pointwise.

Definition 3.1 Let $(A, \leq)$ be a complete lattice.

A **fixpoint-equation system** over A is a finite sequence of equations of the form $(\sigma X = f)$, where $f : A^n \to A$ for some $n \in I\!\!N$ is a monotone function.

The empty sequence is denoted by ϵ.

[1]The version of notation used here was inspired from Vergauwen [Ver95] who pointed me to it for the special case of fixpoint-equation systems over the Boolean lattice. It turned out to be more compact than earlier versions.

In the following $\mathcal{E}, \mathcal{E}', \mathcal{E}_1, \ldots$ will range over fixpoint-equation systems. For technical reasons we assume that no two equations of a fixpoint-equation system $\mathcal{E}$ have the same left hand side variable. Variables which appear on the left hand side of an equation of $\mathcal{E}$ are collected in the set $lhs(\mathcal{E})$, i.e. $lhs((\sigma X = f)\,\mathcal{E}) \overset{\text{def}}{=} \{X\} \cup lhs(\mathcal{E})$. Variables on the right side of an equation of $\mathcal{E}$ are collected in the set $rhs(\mathcal{E})$. Variables of $rhs(\mathcal{E})$ which are contained in $lhs(\mathcal{E})$ are called **bound**. Variables which are not bound are **free**, $free(\mathcal{E}) \overset{\text{def}}{=} rhs(\mathcal{E}) \setminus lhs(\mathcal{E})$. A **block** in a fixpoint-equation system $\mathcal{E}$ is a set of consecutive equations of $\mathcal{E}$ all having the same fixpoint operator in front.
The order defined below reflects the linear order of equations in a fixpoint-equation system. It will be applied to both equations and variables.

Definition 3.2 Let $(\sigma X = f)\ \mathcal{E}$ be a fixpoint-equation system and $\sigma' Y = g$ an equation of $\mathcal{E}$. Then $\sigma X = f \lhd \sigma' Y = g$ and also $X \lhd Y$. As usual $X \unlhd Y$ abbreviates ($X \lhd Y$ or $X = Y$).

A fixpoint-equation system $\mathcal{E}'$ is a **subsystem** of a fixpoint-equation system $\mathcal{E}$, if for each pair of equations with $(\sigma_X X = f_X) \lhd (\sigma_Y Y = f_Y)$ in $\mathcal{E}'$ both equations are contained in $\mathcal{E}$ and ordered in the same way. A subsystem $\mathcal{E}'$ of a fixpoint-equation system $\mathcal{E}$ is called **closed with respect to** $\mathcal{E}$, if $free(\mathcal{E}') \subseteq free(\mathcal{E})$.

Definition 3.3 Let $(A, \leq)$ be a complete lattice, $(\sigma X = f)\ \mathcal{E}$ a fixpoint-equation system over A, and $\theta : \mathcal{X} \to A$ an environment.

The **solution of a fixpoint-equation system relative to** θ is an environment defined by structural induction:

$$\begin{array}{rcl}
[\epsilon]\,\theta & \overset{\text{def}}{=} & \theta \\
[(\mu X = f)\,\mathcal{E}]\,\theta & \overset{\text{def}}{=} & [\mathcal{E}]\,\theta\,[X/\mu X.f([\mathcal{E}]\theta)] \\
[(\nu X = f)\,\mathcal{E}]\,\theta & \overset{\text{def}}{=} & [\mathcal{E}]\,\theta\,[X/\nu X.f([\mathcal{E}]\theta)] \\
\text{where} & & \\
\mu X.f([\mathcal{E}]\,\theta) & = & \bigwedge\{a \mid a \geq f([\mathcal{E}]\,\theta\,[X/a])\} \\
\nu X.f([\mathcal{E}]\,\theta) & = & \bigvee\{a \mid a \leq f([\mathcal{E}]\,\theta\,[X/a])\}
\end{array}$$

Note, that if all variables of $rhs(\mathcal{E})$ are bound, then $[\mathcal{E}]\,\theta_1 = [\mathcal{E}]\,\theta_2$ holds for all environments θ_1, θ_2.

Definition 3.4 Given a fixpoint-equation system $\mathcal{E}$ we define a **lexicographic order** $\leq_{\mathcal{E}}$ on environments.

$\theta_1 \leq_{\epsilon} \theta_2$ iff $\theta_1 = \theta_2$

Let $\mathcal{E} \equiv (\mu X = f)\,\mathcal{E}'$.

$\theta_1 \leq_{\mathcal{E}} \theta_2$ iff $\theta_1(X) < \theta_2(X)$ or $\theta_1(X) = \theta_2(X)$ and $\theta_1 \leq_{\mathcal{E}'} \theta_2$.

Dually, if $\mathcal{E} \equiv (\nu X = f)\,\mathcal{E}'$, then

$\theta_1 \leq_{\mathcal{E}} \theta_2$ iff $\theta_1(X) > \theta_2(X)$ or $\theta_1(X) = \theta_2(X)$ and $\theta_1 \leq_{\mathcal{E}'} \theta_2$.

There exists an alternative characterization of the solution of a fixpoint-equation system, which in some contexts will be more suitable.

Proposition 3.5 The solution of $[\epsilon]\,\theta$ is θ.

The solution of $[(\sigma X = f)\ \mathcal{E}]\,\theta$ is the lexicographically least (w.r.t $(\sigma X = f)\ \mathcal{E})$ environment θ_1 satisfying:

(1) $f(\theta_1) = \theta_1(X)$ and

(2) θ_1 is the solution of $[\mathcal{E}]\,\theta[\,X\,/\,\theta_1(X)\,]$.

Definition 3.6 For $\mathcal{E} = (\sigma_1 X_1 = f_1)(\sigma_2 X_2 = f_2)\ldots(\sigma_n X_n = f_n)$ let $\mathcal{E}^{(i)} \overset{\text{def}}{=} (\sigma_i X_i = f_i)\ldots(\sigma_n X_n = f_n)$ for $1 \leq i \leq n$.

Corollary 3.7 If $[\mathcal{E}]\,\theta = \theta'$ then $[\mathcal{E}^{(i)}]\,\theta' = \theta'$ for $1 \leq i \leq n$.

The characterization of the solution will be illustrated by an example over the Boolean lattice $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$, where $\mathsf{false} < \mathsf{true}$.

Example: Let $(\nu X_1 = X_2 \wedge X_4)$ $(\mu X_2 = X_3 \vee X_1)$ $(\nu X_3 = X_4 \wedge X_2)$ $(\mu X_4 = X_1 \vee X_3)$ be a fixpoint-equation system over $\mathbb{B}$.
Starting from the fixpoint-equation system consisting only of the last equation $\mu X_4 = X_1 \vee X_3$ we will select stepwise all environments fulfilling point (2) of proposition 3.5, then those fulfilling point (1), and in the next step the remaining environments are considered for the equation system with one equation more.

For readability we write an environment θ here as a vector (b_1, b_2, b_3, b_4), meaning an environment θ where $\theta(X_i) = b_i$.
For the equation system consisting of the last equation $\mathcal{E}^{(4)} \equiv (\mu X_4 = X_1 \vee X_3)$, it is the case that

$$\begin{array}{rcl}
[\mathcal{E}^{(4)}]\,(\mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true}) & = & (\mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{true}) & = & (\mathsf{true}, \mathsf{false}, \mathsf{true}, \mathsf{true}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{true}, \mathsf{true}, \mathsf{false}, \mathsf{true}) & = & (\mathsf{true}, \mathsf{true}, \mathsf{false}, \mathsf{true}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}) & = & (\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{false}, \mathsf{false}, \mathsf{true}, \mathsf{true}) & = & (\mathsf{false}, \mathsf{false}, \mathsf{true}, \mathsf{true}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{true}) & = & (\mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{true}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{false}, \mathsf{true}\,\mathsf{false}, \mathsf{false}) & = & (\mathsf{false}, \mathsf{true}\,\mathsf{false}, \mathsf{false}) \\
[\mathcal{E}^{(4)}]\,(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false}) & = & (\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false})
\end{array}$$

Now we go on with $\mathcal{E}^{(3)} \equiv (\nu X_3 = X_4 \wedge X_2)\ (\mu X_4 = X_1 \vee X_3)$
Each of the environments above fulfill point (2) of proposition 3.5, but not all of them fulfill point (1), i.e., the equation $X_3 = X_4 \wedge X_2$; the following do:

$(\mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true})$
$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})$
$(\mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{true})$
$(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false})$

Note that for all these four environments it is
$[\mathcal{E}^{(3)}]\,\theta = [(\nu X_3 = X_4 \wedge X_2)\ (\mu X_4 = X_1 \vee X_3)]\,\theta = \theta$
The next step is to select these environments θ which fulfill also the equation $X_2 = X_3 \vee X_1$. These are:

$(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})$
$(\mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{true})$
$(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false})$

But here it is the not the case that each of these satisfies
$[\mathcal{E}^{(2)}]\,\theta = [(\mu X_2 = X_3 \vee X_1)\ (\nu X_3 = X_4 \wedge X_2)\ (\mu X_4 = X_1 \vee X_3)]\,\theta = \theta.$

For $(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})$ we have

$$\begin{aligned}
& [\mathcal{E}^{(2)}]\,(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}) \\
= \; & [(\mu X_2 = X_3 \vee X_1)\ (\nu X_3 = X_4 \wedge X_2)\ (\mu X_4 = X_1 \vee X_3)] \\
& \qquad\qquad\qquad\qquad (\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}) \\
= \; & [(\mu X_2 = X_3 \vee \mathsf{true})\ (\nu X_3 = X_4 \wedge X_2)\ (\mu X_4 = \mathsf{true} \vee X_3)] \\
& \qquad\qquad\qquad\qquad (\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}) \\
= \; & [(\mu X_2 = \mathsf{true})\ (\nu X_3 = X_4 \wedge X_2)\ (\mu X_4 = \mathsf{true}]\,(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true}) \\
= \; & (\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})
\end{aligned}$$

On the other hand, $(\mathsf{false}, \mathsf{true}, \mathsf{true}, \mathsf{true})$ and $(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false})$ coincide in the free variable of $\mathcal{E}^{(2)}$, which is X_1 and equals to false. Both fulfill point (1) and (2) of proposition 3.5. Hence solution is only the lexicographiclly smaller one with respect to $\mathcal{E}^{(2)}$, which is $(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false})$ (because of the mu-fixpoint in the equation of X_2).

Both environments $(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})$ and $(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false})$ fulfill equation $X_1 = X_2 \wedge X_4$ and the lexicographically smaller one, here $(\mathsf{true}, \mathsf{true}, \mathsf{true}, \mathsf{true})$ (because of the ν-fixpoint of X_1) is the solution of the equation system. $\triangleleft$

Unfortunately, the definition of the solution of a Boolean equation system is not very intuitive, and an interesting question is, whether there exists a more illuminating characterization.

A natural idea is to determine the set of all environments that fulfill all equations $\theta(X_i) = f_i(\theta)$, and then, according to the fixpoint operators, select one environment as the solution.

Unfortunately this approach cannot work. Counterexamples can be found in section 6.1 and also some more discussion of this point.

The question for a clearer characterization of the solution is closely related to the methods which determine the solution. This will be treated in chapter 6.

Fixpoint equation systems are introduced as an extended notation for nested fixpoint expressions. We now define a transformation from fixpoint expressions to fixpoint-equation systems and show that the semantic is preserved.

The transformation is divided into two functions. One, $\mathbf{E}$, maps the tree-like structure of a fixpoint expression to a sequence of expressions. The other one, $\mathbf{E}'$ turns expressions into fixpoint equations. We start with an example and give the formal definition afterwards.

Example:

$$
\begin{aligned}
&\mathbf{E}(\ \mu X.((\nu Y.X \vee Y) \wedge (\nu Z.X \wedge Z))\ ) \\
&\quad = (\mu X = \mathbf{E}'(\ (\nu Y.X \vee Y) \wedge (\nu Z.X \wedge Z)\ )) \\
&\qquad\qquad\qquad\qquad\qquad \mathbf{E}(\ (\nu Y.X \vee Y) \wedge (\nu Z.X \wedge Z)\ ) \\
&\quad = (\mu X = \mathbf{E}'(\nu Y.X \vee Y) \wedge \mathbf{E}'(\nu Z.X \wedge Z))\ \mathbf{E}(\nu Y.X \vee Y)\ \mathbf{E}(\nu Z.X \wedge Z) \\
&\quad = (\mu X = Y \wedge Z)\ (\nu Y = \mathbf{E}'(X \vee Y))\ (\nu Z = \mathbf{E}'(X \wedge Z)) \\
&\quad = (\mu X = Y \wedge Z)\ (\nu Y = X \vee Y)\ (\nu Z = X \wedge Z)
\end{aligned}
$$

Definition 3.8 Let $\sigma X.f$ be a fixpoint expression over a lattice $(A, \leq)$, where f is a monotone function on A consisting of constants, variables, fixpoint expressions, the lattice operations $\vee$ and $\wedge$ and additionally a set of monotone k_i-ary operations on A, denoted by $Op_i^{(k_i)}$ for some $i \in I\!N$. Assume that in $\sigma X.f$ names are unique, i.e. each variable is bound only once by a fixpoint operator. $\mathbf{E}$ maps $\sigma X.f$ to a fixpoint-equation system and is defined as follows:

$$
\begin{aligned}
\mathbf{E}(a) &= \epsilon \\
\mathbf{E}(X) &= \epsilon \\
\mathbf{E}(f_1 \wedge f_2) &= \mathbf{E}(f_1)\,\mathbf{E}(f_2) \\
\mathbf{E}(f_1 \vee f_2) &= \mathbf{E}(f_1)\,\mathbf{E}(f_2) \\
\mathbf{E}(Op_i^{(k_i)}(f_1, \ldots, f_{k_i})) &= \mathbf{E}(f_1) \ldots \mathbf{E}(f_{k_i}) \\
\mathbf{E}(\sigma X.f) &= (\sigma X = \mathbf{E}'(f))\ (\mathbf{E}(f))
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{E}'(a) &= a \\
\mathbf{E}'(X) &= X \\
\mathbf{E}'(f_1 \wedge f_2) &= \mathbf{E}'(f_1) \wedge \mathbf{E}'(f_2) \\
\mathbf{E}'(f_1 \vee f_2) &= \mathbf{E}'(f_1) \vee \mathbf{E}'(f_2) \\
\mathbf{E}'(Op_i^{(k_i)}(f_1, \ldots, f_{k_i})) &= Op_i^{(k_i)}(\mathbf{E}'(f_1), \ldots, \mathbf{E}'(f_{k_i})) \\
\mathbf{E}'(\sigma X.f) &= X
\end{aligned}
$$

Proposition 3.9 Let $\sigma X.f$ be a fixpoint expression over a lattice $(A, \leq)$ and θ an arbitrary environment.

Then $(\sigma X.f)(\theta) = ([\mathbf{E}(\sigma X.f)]\,\theta)(X)$.

The proof of this proposition requires the following lemma:

Lemma 3.10 Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be fixpoint-equation systems, such that

- $lhs(\mathcal{E}_1) \cap lhs(\mathcal{E}_2) = \emptyset$,
- $lhs(\mathcal{E}_1) \cap rhs(\mathcal{E}_2) = \emptyset$,
- $lhs(\mathcal{E}_2) \cap rhs(\mathcal{E}_1) = \emptyset$.

Then $[\mathcal{E}_1][\mathcal{E}_2]\theta = [\mathcal{E}_1\mathcal{E}_2]\theta$

Note that a straightforward transformation back from a fixpoint-equation system to a (nested) fixpoint expression is not always possible. For Boolean equation systems in the context of μ-calculus model checking we will show a method in section 5.2. In general, a fixpoint-equation system can be transformed back to a set of (nested) fixpoint expressions. For example $(\nu X = Y)(\mu U = V)$ is a fixpoint-equation system, and $\nu X.Y$ and $\mu U.V$ are (the only sensible) fixpoint expressions corresponding to it.
Another example is $(\nu X = Z)(\mu Y = X)(\nu Z = Y)$. It might correspond to the expression $\nu X.\nu Z.\mu Y.X$, but the transformation of this expression to a fixpoint-equation system gives $(\nu X = Z)(\nu Z = Y)(\mu Y = X)$.
In the following we present a collection of properties of fixpoint equation systems which describe equivalence and order relations on their solutions. The first one states that a fixpoint-equation system is a monotone operator on environments.

Lemma 3.11 If $\theta_1 \leq \theta_2$ then $[\mathcal{E}]\,\theta_1 \leq [\mathcal{E}]\,\theta_2$.

It is easy to see that in the lemma above only for variables X that are free in $\mathcal{E}$ we need the condition $\theta_1(X) \leq \theta_2(X)$. Hence the order of the environments defined pointwise on all variables can be restricted to the variables which are free in $\mathcal{E}$.

Corollary 3.12 $[\mathcal{E}]\,\theta_1 \vee [\mathcal{E}]\,\theta_2 \leq [\mathcal{E}](\,\theta_1 \vee\, \theta_2)$, and

$$[\mathcal{E}]\,\theta_1 \wedge [\mathcal{E}]\,\theta_2 \geq [\mathcal{E}](\,\theta_1 \wedge\, \theta_2).$$

We define an equivalence relation $\sim$ and an order $\precsim$ on fixpoint-equation systems, relating those that have the same solution, or solutions ordered by $\leq$ respectively, for all environments. It extends the equivalence relation $\sim$ defined in [Ver95] for Boolean equation systems.

Definition 3.13

Define $\mathcal{E}_1 \sim \mathcal{E}_2$ iff $[\mathcal{E}_1]\,\theta\; = [\mathcal{E}_2]\,\theta$ for all environments θ.

Define $\mathcal{E}_1 \precsim \mathcal{E}_2$ iff $[\mathcal{E}_1]\,\theta\; \leq [\mathcal{E}_2]\,\theta$ for all environments θ.

Equivalence and ordering of fixpoint-equation systems is preserved for prefixing of equations. For equivalence on Boolean equation systems this result was stated in [Ver95].

Lemma 3.14 If $\mathcal{E}_1 \sim \mathcal{E}_2$ then $\mathcal{E}\mathcal{E}_1 \sim \mathcal{E}\mathcal{E}_2$.

If $\mathcal{E}_1 \precsim \mathcal{E}_2$ then $\mathcal{E}\mathcal{E}_1 \precsim \mathcal{E}\mathcal{E}_2$.

Definition 3.15 Let for some $n \in I\!N$

- $\mathcal{E}_1 \equiv (\sigma_1 X_1 = f_1)\ldots(\sigma_n X_n = f_n)$,
- $\mathcal{E}_2 \equiv (\sigma_1 X_1 = g_1)\ldots(\sigma_n X_n = g_n)$.

Then $\mathcal{E}_1 \leq \mathcal{E}_2$ iff $f_i \leq g_i$.

$\mathcal{E}_1 \vee \mathcal{E}_2 \overset{\text{def}}{=} (\sigma_1 X_1 = f_1 \vee g_1)\ldots(\sigma_n X_n = f_n \vee g_n)$,

$\mathcal{E}_1 \wedge \mathcal{E}_2 \overset{\text{def}}{=} (\sigma_1 X_1 = f_1 \wedge g_1)\ldots(\sigma_n X_n = f_n \wedge g_n)$,

Lemma 3.16 If $\mathcal{E}_1 \leq \mathcal{E}_2$ then also $\mathcal{E}_1 \precsim \mathcal{E}_2$

Corollary 3.17 $[\mathcal{E}_1]\,\theta \vee [\mathcal{E}_2]\,\theta\; \leq [\mathcal{E}_1 \vee \mathcal{E}_2]\,\theta$, and

$$[\mathcal{E}_1]\,\theta \wedge [\mathcal{E}_2]\,\theta\; \geq [\mathcal{E}_1 \wedge \mathcal{E}_2]\,\theta\,.$$

This corollary will be illustrated by an example, where even in the case that both systems have the same solution their disjunction has a greater one:

Example: Again the lattice is $(\mathbb{B}, \leq)$. Consider two fixpoint-equation systems $\mathcal{E}_1 \equiv (\nu X_1 = X_2)\ (\mu X_2 = X_2)\ (\nu X_3 = X_1)$ and $\mathcal{E}_2 \equiv (\nu X_1 = X_3)\ (\mu X_2 = X_3)\ (\nu X_3 = X_2)$

Both have the same solution $(\mathsf{false}, \mathsf{false}, \mathsf{false})$ for any θ. However, the solution of their disjunction $(\nu X_1 = X_2 \vee X_3)\ (\mu X_2 = X_2 \vee X_3)\ (\nu X_3 = X_1 \vee X_2)$ is $(\mathsf{true}, \mathsf{true}, \mathsf{true})$. $\lhd$

There are other simple, desirable properties which surprisingly do **not** hold. We demonstrate here one of them.

If $\mathcal{E}_1 \precsim \mathcal{E}_2$ then $\mathcal{E} \vee \mathcal{E}_1 \precsim \mathcal{E} \vee \mathcal{E}_2$ and $\mathcal{E} \wedge \mathcal{E}_1 \precsim \mathcal{E} \wedge \mathcal{E}_2$

Counterexample: Let $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ be fixpoint-equation systems over the Boolean lattice $(\mathbb{B}, \leq)$.

$$\begin{array}{lcl} \mathcal{E}_1 & \equiv & (\mu X_1 = X_1)\ (\nu X_2 = X_2)\ (\nu X_3 = X_3) \\ \mathcal{E}_2 & \equiv & (\mu X_1 = X_2)\ (\nu X_2 = X_3)\ (\nu X_3 = X_2) \\ \mathcal{E} & \equiv & (\mu X_1 = X_2)\ (\nu X_2 = X_1)\ (\nu X_3 = X_3) \end{array}$$

$\mathcal{E}_1$ has the solution $(\mathsf{false}, \mathsf{true}, \mathsf{true})$ and $\mathcal{E}_2$ has the solution $(\mathsf{true}, \mathsf{true}, \mathsf{true})$. Hence $\mathcal{E}_1 \precsim \mathcal{E}_2$.

The solution of $\mathcal{E}$ is $(\mathsf{false}, \mathsf{false}, \mathsf{true})$. $\mathcal{E}_1 \wedge \mathcal{E}$ has also the solution $(\mathsf{false}, \mathsf{false}, \mathsf{true})$, $\mathcal{E}_2 \wedge \mathcal{E}$ has $(\mathsf{false}, \mathsf{false}, \mathsf{false})$ as solution. Here $\mathcal{E} \wedge \mathcal{E}_2 \precsim \mathcal{E} \wedge \mathcal{E}_1$. $\lhd$

The following lemma extends a property for Boolean equation systems in [Ver95] to fixpoint-equation systems.

Lemma 3.18 If $([(\sigma X = f)\ \mathcal{E}]\,\theta)(X) = ([(\sigma X = g)\ \mathcal{E}]\,\theta)(X)$
then $[(\sigma X = f)\ \mathcal{E}]\,\theta = [(\sigma X = g)\ \mathcal{E}]\,\theta$.

The next both lemmata deal with a quite natural property: when knowing parts of the solution then these parts may be "removed" from the equation system preserving the solution. This allows stepwise solving and reduction methods for fixpoint-equation systems.

Lemma 3.19 Let

- $\mathcal{E} \equiv \mathcal{E}_1 \ (\sigma X = f) \ \mathcal{E}_2$,
- $([\mathcal{E}]\,\theta)(X) = a$, and
- $\mathcal{E}' \equiv \mathcal{E}_1 \ (\sigma X = a) \ \mathcal{E}_2$.

Then $[\mathcal{E}]\,\theta = [\mathcal{E}']\,\theta$.

Lemma 3.20 $[\mathcal{E}_1 \ (\sigma X = a) \ \mathcal{E}_2]\,\theta = [\mathcal{E}_1 \ \mathcal{E}_2]\,\theta\,[X/a]$.

The following lemmata describe properties of the solution of a fixpoint-equation system when interchanging equations or switching the fixpoint operator, from μ to ν, or the other way round. From Bekič's Theorem 2.24 it follows that interchanging subsequent equations with the same fixpoint operator does not influence the solution. The same holds for equations with different fixpoint operators if the variables of both equations are different. Otherwise interchanging equations implies different solutions which are ordered pointwise. This property is slightly surprising having in mind the characterization of a solution which refers to lexicographic ordering (Proposition 3.5).

Lemma 3.21 Let

- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1 \ (\sigma X_1 = f_1) \ (\sigma X_2 = f_2) \ \mathcal{E}_2]\,\theta$,
- $\theta_2 \overset{\text{def}}{=} [\mathcal{E}_1 \ (\sigma X_2 = f_2) \ (\sigma X_1 = f_1) \ \mathcal{E}_2]\,\theta$.

Then $\theta_1 = \theta_2$.

Lemma 3.22 If

- X_1 is not free in f_2,
- X_2 is not free in f_1,
- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1 \ (\sigma_1 X_1 = f_1) \ (\sigma_2 X_2 = f_2) \ \mathcal{E}_2]\,\theta$
- $\theta_2 \overset{\text{def}}{=} [\mathcal{E}_1 \ (\sigma_2 X_2 = f_2) \ (\sigma_1 X_1 = f_1) \ \mathcal{E}_2]\,\theta$

Then $\theta_1 = \theta_2$.

Lemma 3.23 Let

- $\theta_1 \stackrel{\text{def}}{=} [\mathcal{E}_1 \ (\mu X_1 = f_1) \ (\nu X_2 = f_2) \ \mathcal{E}_2]\,\theta\,,$
- $\theta_2 \stackrel{\text{def}}{=} [\mathcal{E}_1 \ (\nu X_2 = f_2) \ (\mu X_1 = f_1) \ \mathcal{E}_2]\,\theta\,.$

Then it is $\theta_1 \leq \theta_2$, and moreover, if the inequality is strict then $\theta_1(X_1) < \theta_2(X_1)$ and $\theta_1(X_2) < \theta_2(X_2)$.

Lemma 3.24 Let

- $\theta_1 \stackrel{\text{def}}{=} [\mathcal{E}_1 \ (\mu X = f) \ \mathcal{E}_2]\,\theta,$
- $\theta_2 \stackrel{\text{def}}{=} [\mathcal{E}_1 \ (\nu X = f) \ \mathcal{E}_2]\,\theta.$

Then it is $\theta_1 \leq \theta_2$, and moreover, if the inequality is strict then $\theta_1(X) < \theta_2(X)$.

Often we need some standard representation of fixpoint-equation systems, where each right hand side contains at most one of the operators $\vee$ or $\wedge$. Every fixpoint-equation system can be transformed into such a form by introduction of additional "fresh" variables.

Lemma 3.25

$$([(\sigma X = f_1 \wedge f_2) \ \mathcal{E}]\,\theta)(Y) = ([(\sigma X = f_1 \wedge X') \ (\sigma' X' = f_2) \ \mathcal{E}]\,\theta)(Y),$$

$$([(\sigma X = f_1 \vee f_2)\mathcal{E}]\,\theta)(Y) = ([(\sigma X = f_1 \vee X')(\sigma' X' = f_2)\mathcal{E}]\,\theta)(Y),$$

where X' is a new variable, i.e. (*) X' does not occur on the right hand side of $\mathcal{E}$ or in f_1 or f_2, and (**) $Y \neq X'$.

For reduction of fixpoint-equation systems the next property is useful: within a block duplicate equations may be removed.

Lemma 3.26 Let

- $\theta_1 \stackrel{\text{def}}{=} [\mathcal{E}_1 \ (\sigma X_1 = f) \ (\sigma X_2 = f) \ \mathcal{E}_2]\,\theta$
- $\theta_2' \stackrel{\text{def}}{=} [\mathcal{E}_1[X_1/X_2] \ (\sigma X_2 = f[X_1/X_2]) \ \mathcal{E}_2[X_1/X_2]]\,\theta$
- $\theta_2 \stackrel{\text{def}}{=} \theta_2'[X_1/\theta_2'(X_2)]$

Then $\theta_1 = \theta_2$.

3.2 Boolean equation systems.

We now introduce Boolean equation systems as a special case of fixpoint-equation systems, where the underlying lattice is the Boolean lattice $\{\mathsf{true}, \mathsf{false}\}$ with $\mathsf{false} < \mathsf{true}$. Of course, syntax and semantics as defined in section 3 apply also to Boolean equation systems. However, interpreted over the Boolean lattice dealing with fixpoints gets simpler, and we will reintroduce syntax and semantics for this special case. In order to distinguish the Boolean case also syntactically we choose $[\![\ ]\!]$ instead of $[\ ]$ as semantic brackets.
Let $\mathcal{X}$ be a set of Boolean variables, and $f, g, \ldots$ range over Boolean expressions. Analogously to definition 3.1 we define:

Definition 3.27 The set of negation free Boolean expressions over a countable set of variables $\mathcal{X}$ is denoted by $\mathbb{B}^+(\mathcal{X})$.

A Boolean equation is of the form $\sigma X{=}f$, where $\sigma \in \{\mu, \nu\}$, X is a Boolean variable $X \in \mathcal{X}$, and $f \in \mathbb{B}^+(\mathcal{X})$.

A Boolean equation system is a sequence of Boolean equations. The empty sequence ϵ is a Boolean equation system; if $\sigma X{=}f$ is a Boolean equation and $\mathcal{E}$ is a Boolean equation system, then $(\sigma X{=}f)\ \mathcal{E}$ is also a Boolean equation system.

Dealing with fixpoints gets much simpler over the Boolean lattice. The following two lemmata show that the least and greatest fixpoints of Boolean functions can be represented as functions themselves. In contrast to standard definitions it is not necessary to evaluate them pointwise. (See also definition 2.21 and remark 2.23.)

Lemma 3.28 Suppose $f(X)$ is a monotone Boolean function in the single variable X. Then $\mu X.f(X) = f(\mathsf{false})$ and $\nu X.f(X) = f(\mathsf{true})$.

Lemma 3.29 Suppose $f(X_1, \ldots, X_n)$ is a monotone Boolean function from $\mathbb{B}^n$ to $\mathbb{B}$. Then its least and greatest fixpoints with respect to X_1 are $\mu X_1.f(X_1, \ldots, X_n) = f(\mathsf{false}, X_2, \ldots, X_n)$ and $\nu X_1.f(X_1, \ldots, X_n) = f(\mathsf{true}, X_2, \ldots, X_n)$ and both are monotone function from $\mathbb{B}^{n-1}$ to $\mathbb{B}$.

Proposition 3.30 Let $\mathcal{E}$ be a Boolean equation system, $\sigma X{=}f$ a Boolean equation, θ an environment, $b_\mu = \mathsf{false}$ and $b_\nu = \mathsf{true}$. Then for the solution of a Boolean equation system it is the case that:

$$\begin{array}{rcl} [\![\epsilon]\!]\,\theta & = & \theta \\ [\![(\sigma X{=}f)\ \mathcal{E}]\!]\,\theta & = & [\![\mathcal{E}]\!]\,\theta\,[X\,/\,f(\,[\![\mathcal{E}]\!]\,\theta\,[X/b_\sigma]\,)\,]. \end{array}$$

Example: Consider the equation system $\nu X_1{=}X_1$ and arbitrary θ.

$$\begin{array}{rcl} [\![\nu X_1 = X_1]\!]\,\theta & = & [\![\epsilon]\!]\,\theta[X_1/(X_1)([\![\epsilon]\!]\,\theta[X_1/\mathsf{true}])] \\ & = & \theta[X_1/(X_1)(\theta[X_1/\mathsf{true}])] \\ & = & \theta[X_1/\mathsf{true}] \end{array}$$

Hence the solution of this Boolean equation system is $X_1 = \mathsf{true}$. $\lhd$

The **size** of a Boolean equation system $\mathcal{E}$ is defined as the number of its variables and the size of all its right-hand side expressions,

$$\begin{array}{rcl} |\epsilon| & \stackrel{\text{def}}{=} & 0 \\ |(\sigma X = f)\ \mathcal{E}| & \stackrel{\text{def}}{=} & 1 + |f| + |\mathcal{E}|. \end{array}$$

The size of a negation free Boolean expression $|f|$ is the number of variables and constants contained in f.

A Boolean equation system $\mathcal{E}$ is in **simple form**, if each right-hand side expression consists of conjunctions, or disjunctions, or a single variable or a constant.

In some contexts it is useful to restrict the size of the right-hand size expressions to 2. This gives rise to the following definition of a standard form for Boolean equation systems.

A Boolean equation system $\mathcal{E}$ is in **standard form**, if

- $lhs(\mathcal{E}) = \{X_1, \ldots, X_n\}$ for some $n \in I\!N$
- if $X_i \lhd X_j$ then $i < j$
- each right-hand side expression consists of a disjunction $X_i \vee X_j$, a conjunction $X_i \wedge X_j$, a single variable X_i or a constant true or false.

Proposition 3.31 For each Boolean equation system $\mathcal{E}$ there exists a Boolean equation system $\mathcal{E}'$ in standard form and a renaming function λ, such that $([\![\mathcal{E}]\!]\,\theta)(X) = ([\![\mathcal{E}']\!]\,\theta)(\lambda(X))$, and $\mathcal{E}'$ has size linear in the size of $\mathcal{E}$.

A Boolean equation system can be devided into **blocks**. A block is defined as a set of consecutive equations of $\mathcal{E}$ having the same fixpoint operator in front. Hence we can distinguish **μ-blocks** and **ν-blocks**. The linear orderings $\trianglelefteq$ and $\triangleleft$ on the equations of a Boolean equation system extends naturally to an order $\trianglelefteq$ and $\triangleleft$ on the blocks of a Boolean equation system. We now define **active variables**, **nesting depth** and **alternation depth** for Boolean equation systems..

Definition 3.32 Let

- $\mathcal{E}$ be a Boolean equation system,
- $\sigma_X X = f_X$, $\sigma_Y Y = f_Y$ be equations of $\mathcal{E}$,
- $\sigma_X X = f_X \triangleleft \sigma_Y Y = f_Y$

Then X is active in $\sigma_Y Y = f_Y$ iff

- there is a free occurrence of X in $\sigma_Y Y = f_Y$, or
- some variable Z is free in $\sigma_Y Y = f_Y$, $X \triangleleft Z \triangleleft Y$ and X is active in $\sigma_Z Z = f_Z$.

A variable X is active in a block $\mathcal{E}_j$, if it is active in any equation of $\mathcal{E}_j$.

A block $\mathcal{E}_i$ is active in a block $\mathcal{E}_j$, if some variable X in an equation $\sigma_X X = f_X$ of $\mathcal{E}_i$ is active in $\mathcal{E}_j$.

When defining the nesting depth and alternation depth of fixpoint operators for expressions we have to take into account that the subformula order is a partial order. In the case of Boolean equation systems we have just a linear order on the equations. However, the partial order is reflected by the possible applications of lemma 3.22 to interchange equations.

Definition 3.33 The nesting depth of a Boolean equation system $\mathcal{E}$ is the minimal number of blocks of all Boolean equation systems that can be derived from $\mathcal{E}$ by (repeated) interchanging of equations according to lemma 3.22.

Definition 3.34 Let

- $\mathcal{E}$ be a Boolean equation system,
- $\sigma_1 X_1 = f_1 \lhd \ldots \lhd \sigma_n X_n = f_n$ a chain of Boolean equations of maximal length, such that for every $1 \leq i < n$
  (1) X_i is active in $\sigma_{i+1} X_{i+1} = f_{i+1}$,
  (2) X_n is free in f_n, and
  (3) $\sigma_i \neq \sigma_{i+1}$,

Then $\mathcal{E}$ has alternation depth n, i.e. $ad(\mathcal{E}) = n$.

Section 3 contains a number of properties of fixpoint equation systems, which are, of course, also valid for the special case of Boolean equation systems. In addition to these there exist more properties for Boolean equation systems, which will be needed in later chapters.
The complementation $\overline{\mathcal{E}}$ of a Boolean equation system is defined inductively as follows:

$$\begin{array}{rrcl}
 & \overline{\epsilon} & = & \epsilon \\
 & \overline{(\sigma X = f)\ \mathcal{E}} & = & (\overline{\sigma} X = \overline{f})\ \overline{\mathcal{E}}, \\
\text{where} & \overline{\mu} & = & \nu \\
 & \overline{\nu} & = & \mu \\
 & \overline{\mathsf{true}} & = & \mathsf{false} \\
 & \overline{\mathsf{false}} & = & \mathsf{true} \\
 & \overline{X} & = & X \text{ for } X \in \mathcal{X} \\
 & \overline{f_1 \wedge f_2} & = & \overline{f_1} \vee \overline{f_2} \\
 & \overline{f_1 \vee f_2} & = & \overline{f_1} \wedge \overline{f_2}
\end{array}$$

The complement $\overline{\theta}$ of an environment θ is defined as $\overline{\theta}(X) = \overline{\theta(X)}$.
The complementation lemma for Boolean equation systems is:

Lemma 3.35 $([\![\mathcal{E}]\!]\,\theta)(X) = \mathsf{false}$ iff $([\![\overline{\mathcal{E}}]\!]\,\overline{\theta})(X) = \mathsf{true}$.

The next is a very strong property about a reduction of Boolean equation systems preserving their solution. Having a Boolean equation system $\mathcal{E}$ in standard form we can construct a new Boolean equation system $\mathcal{E}'$ in the following way: all equations having conjunctions or constants on their right-hand side are unchanged and become equations of $\mathcal{E}'$. In every equation of $\mathcal{E}$ with a disjunction on the right-hand side one disjunct is selected as the new right-hand side for the equation in $\mathcal{E}'$. Note that $\mathcal{E}'$ contains no proper disjunctions and we say, such a system is in **conjunctive form**. The order of variables in $\mathcal{E}$ and $\mathcal{E}'$ is the same. From definition 3.15 and lemma 3.16 we know already that for every environment θ the solution of $\mathcal{E}'$ is lower or equal to the solution of $\mathcal{E}$. The following proposition says even more: for every θ there exists a choice of disjuncts, such that $\mathcal{E}$ and $\mathcal{E}'$ have the same solution. The dual property holds when choosing conjuncts instead of disjuncts.

Proposition 3.36 Given a Boolean equation system $\mathcal{E}$ and an environment θ there exist Boolean equation systems $\mathcal{E}'$ and $\mathcal{E}''$ with the properties:

- $\mathcal{E}'$ is in conjunctive form,
- $\mathcal{E}' \leq \mathcal{E}$, and
- $[\![\mathcal{E}']\!]\,\theta = [\![\mathcal{E}]\!]\,\theta$.

For $\mathcal{E}''$ the dual properties hold:

- $\mathcal{E}''$ is in disjunctive form,
- $\mathcal{E}'' \geq \mathcal{E}$, and
- $[\![\mathcal{E}'']\!]\,\theta = [\![\mathcal{E}]\!]\,\theta$.

Corollary 3.37 For Boolean equation system $\mathcal{E}$ and an environment θ there exists a Boolean equation system $\mathcal{E}'$ with the following properties:

- $[\![\mathcal{E}]\!]\,\theta = [\![\mathcal{E}']\!]\,\theta$, and
- $\mathcal{E}'$ is derived from $\mathcal{E}$ by selecting in every equation one variable of the right hand expression.

Proof: Apply proposition 3.36 for the disjunctive and then for the conjunctive case. □

Chapter 4

The modal μ-calculus.

This chapter gives an introduction to the modal μ-calculus according to Kozen's propositional μ-calculus [Koz83]. The modal μ-calculus has been widely studied and detailed introductions can be found in several places, such as [Sti93] and [Eme91]. Here we will briefly review the logic and its properties and we give associated definitions relevant to our work.

4.1 Syntax and semantics.

The syntax of the modal μ-calculus is defined with respect to a set $\mathcal{Q}$ of atomic propositions including true and false, a finite set $\mathcal{L}$ of action labels and a denumerable set $\mathcal{Z}$ of propositional variables. A formula of the modal μ-calculus is an expression of the form:

$$\Phi ::= Z \mid Q \mid \neg\Phi \mid \Phi \wedge \Phi \mid [a]\Phi \mid \mu Z.\Phi,$$

where $Z \in \mathcal{Z}$, $Q \in \mathcal{Q}$ and $a \in \mathcal{L}$, and where in $\nu Z.\Phi$ every free occurrence of Z in Φ falls under an even number of negations. The standard conventions for the derived operators and abbreviations are:

$$\begin{aligned}
\Phi_1 \vee \Phi_2 &\stackrel{\text{def}}{=} \neg(\neg\Phi_1 \wedge \neg\Phi_2)\\
\langle a\rangle\Phi &\stackrel{\text{def}}{=} \neg[a]\neg\Phi\\
[K]\Phi &\stackrel{\text{def}}{=} \textstyle\bigwedge_{a\in K}[a]\Phi\\
\langle K\rangle\Phi &\stackrel{\text{def}}{=} \textstyle\bigvee_{a\in K}\langle a\rangle\Phi
\end{aligned}$$

$$\begin{aligned}[-K]\Phi &\overset{\text{def}}{=} [\mathcal{L}\setminus K]\Phi\\ [-]\Phi &\overset{\text{def}}{=} [\mathcal{L}]\Phi\\ \nu Z.\Phi &\overset{\text{def}}{=} \neg\mu Z.\neg\Phi[Z/\neg Z],\end{aligned}$$

where $K \subseteq \mathcal{L}$, and $\Phi[Z/\neg Z]$ means the syntactical substitution of every occurrence of Z in Φ by $\neg Z$.

We denote the set of all modal μ-calculus formulae by μL, and the subset of fixpoint free formulae by Π_0.

A formula is in **positive normal form**, if negations apply only to atomic propositions and no variable is quantified twice. Every formula can be transformed syntactically into positive normal form by using the derived operators, applying the De Morgan rules and renaming variables. Therefore, we can restrict the set of formulae to the positive fragment assuming that for every atomic proposition $Q \in \mathcal{Q}$ the negation of Q is also an atomic proposition, i.e. an element of $\mathcal{Q}$. In this sense, an equivalent definition of the syntax is:

$$\Phi ::= Z \mid Q \mid \Phi\wedge\Phi \mid \Phi\vee\Phi \mid [a]\Phi \mid \langle a\rangle\Phi \mid \mu Z.\Phi \mid \nu Z.\Phi$$

In the following we will refer only to formulae of the positive fragment and assume that they are in normal form.

Formulae of the modal μ-calculus with the set $\mathcal{L}$ of action labels are interpreted relative to a **labelled transition system** $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{L}\})$, where $\mathcal{S}$ is a possibly infinite set of states and $\xrightarrow{a}\subseteq \mathcal{S}\times\mathcal{S}$ for every $a \in \mathcal{L}$ a binary relation on states. The union of all relations is denoted by $\rightarrow \overset{\text{def}}{=} \bigcup_{a\in\mathcal{L}} \xrightarrow{a}$. A **valuation function** $\mathcal{V}$ assigns to every atomic proposition Q in $\mathcal{Q}$ and propositional variable Z in $\mathcal{Z}$ a set of states $\mathcal{V}(Q) \subseteq \mathcal{S}$ and $\mathcal{V}(Z) \subseteq \mathcal{S}$ meaning that proposition Q and variable Z hold for every state in $\mathcal{V}(Q)$ and $\mathcal{V}(Z)$. The pair $\mathcal{T}$ and $\mathcal{V}$ is called a **model** $\mathcal{M}$ of the modal μ-calculus. The semantics of each μ-calculus formula Φ is the set of states $\|\Phi\|^{\mathcal{T}}_{\mathcal{V}}$. A state s satisfies a formula Φ, written as $s \models_{\mathcal{M}} \Phi$, iff $s \in \|\Phi\|^{\mathcal{T}}_{\mathcal{V}}$, which is defined inductively as follows:

$$\begin{aligned}\|Q\|^{\mathcal{T}}_{\mathcal{V}} &= \mathcal{V}(Q)\\ \|Z\|^{\mathcal{T}}_{\mathcal{V}} &= \mathcal{V}(Z)\end{aligned}$$

$$\begin{array}{rcl}
\|\Phi_1 \vee \Phi_2\|^{\mathcal{T}}_{\mathcal{V}} & = & \|\Phi_1\|^{\mathcal{T}}_{\mathcal{V}} \cup \|\Phi_2\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\Phi_1 \wedge \Phi_2\|^{\mathcal{T}}_{\mathcal{V}} & = & \|\Phi_1\|^{\mathcal{T}}_{\mathcal{V}} \cap \|\Phi_2\|^{\mathcal{T}}_{\mathcal{V}} \\
\|[a]\Phi\|^{\mathcal{T}}_{\mathcal{V}} & = & [\![a]\!]^{\mathcal{T}} \; \|\Phi\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\langle a\rangle\Phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \langle\!\langle a\rangle\!\rangle^{\mathcal{T}} \; \|\Phi\|^{\mathcal{T}}_{\mathcal{V}} \\
\|\mu Z.\Phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \bigcap\{S' \subseteq \mathcal{S} \mid \|\Phi\|^{\mathcal{T}}_{\mathcal{V}[Z/S']} \subseteq S'\} \\
\|\nu Z.\Phi\|^{\mathcal{T}}_{\mathcal{V}} & = & \bigcup\{S' \subseteq \mathcal{S} \mid S' \subseteq \|\Phi\|^{\mathcal{T}}_{\mathcal{V}[Z/S']}\}
\end{array}$$

$$\begin{array}{lrcl}
\text{where} & [\![a]\!]^{\mathcal{T}} S' & \overset{\text{def}}{=} & \{s \mid \forall s' \in S.\ \text{if } s \xrightarrow{a} s' \text{ then } s' \in S'\} \\
 & \langle\!\langle a\rangle\!\rangle^{\mathcal{T}} S' & \overset{\text{def}}{=} & \{s \mid \exists s' \in S'. s \xrightarrow{a} s'\}
\end{array}$$

Examples for μ-calculus formulae will be given below. First we want to introduce some technical terms.

The **size** of a transition system includes the number of states and the number of transitions, $|\mathcal{T}| \overset{\text{def}}{=} |\mathcal{S}| + |{\rightarrow}|$.

The **branching degree** $|\overset{\swarrow}{\underset{\nwarrow}{\rightarrow}}|$ is the maximal number of successors that any state of the transition system has, $|\overset{\swarrow}{\underset{\nwarrow}{\rightarrow}}| \overset{\text{def}}{=} max_{s\in\mathcal{S}}|\{s' \mid s \rightarrow s'\}|$. An upper bound for the branching degree is the number of states $|\mathcal{S}|$.

The **size of a formula** $|\Phi|$ is defined as follows:

$$\begin{array}{rcl}
|Z| & = & |Q| = 1 \\
|\Phi_1 \vee \Phi_2| & = & |\Phi_1| + |\Phi_2| \\
|\Phi_1 \wedge \Phi_2| & = & |\Phi_1| + |\Phi_2| \\
|[a]\Phi| & = & 1 + |\Phi| \\
|\langle a\rangle\Phi| & = & 1 + |\Phi| \\
|\mu Z.\Phi| & = & 1 + |\Phi| \\
|\nu Z.\Phi| & = & 1 + |\Phi|
\end{array}$$

Definition 4.1 As usual, **subformulae** of a formula Φ are defined inductively on the structure of Φ. If Ψ is a subformula of Φ we will write $\Phi \trianglelefteq \Psi$, and $\Phi \triangleleft \Psi$ if it is a proper subformula.

Definition 4.2 In a formula $\sigma Z.\Psi$ each occurrence of the variable Z is **bound**. An occurrence of X in Φ is bound, if it is bound in any subformula of Φ. An occurrence of a variable which is not bound is called **free**.

We now want to introduce the notions of **nesting depth** and **alternation depth** of fixpoint operators for formulae of μL. The latter will be defined via **active variables** as introduced by Kozen [Koz83]. Alternation depth was defined by Emerson and Lei [EL86] and is a relevant size for many model checking algorithms. Niwiński [Niw86] gave a more sensible definition for alternation depth which we will use with a minor extension. Its definition based on active variables follows Kaivola [Kai96]. There a more detailed discussion of these concepts can be found. A small example for demonstrating the differences is: the Emerson-Lei alternation depth of $\nu X.\mu Y.\nu Z.X$ is 3, its Niwiński alternation depth is 2, whereas we want it to be 1.

Definition 4.3 Let

- Φ be a formula,
- $\Phi \trianglelefteq \sigma_1 X_1.\Psi_1 \triangleleft \ldots \triangleleft \sigma_n X_n.\Psi_n$ a chain of subformulae of maximal length.

Then Φ has nesting depth n, i.e. $nd(\Phi) = n$.

Definition 4.4 Let

- Φ be a modal μ-calculus formula,
- Ψ be a subformula of Φ, i.e. $\Phi \trianglelefteq \Psi$, and
- Z a variable.

Then Z **is active in** Ψ iff

- there is a free occurrence of Z in Ψ, or
- some variable Z' is free in Ψ, $\Phi \trianglelefteq \sigma Z'.\Psi' \trianglelefteq \Psi$, and Z is active in $\sigma Z'.\Psi'$.

Definition 4.5 Let

- Φ be a formula,
- $\Phi \trianglelefteq \sigma_1 X_1.\Psi_1 \lhd \ldots \lhd \sigma_n X_n.\Psi_n$ a chain of subformulae of maximal length, such that for every $1 \leq i < n$
  (1) X_i is active in $\sigma_{i+1}X_{i+1}.\Psi_{i+1}$,
  (2) X_n is free in Ψ_n, and
  (3) $\sigma_i \neq \sigma_{i+1}$.

Then Φ has alternation depth n, i.e. $ad(\Phi) = n$.

Note that our extension consists of point (2) in the previous definition. Leaving it out would give Niwiński alternation depth.

4.2 Basic formulae.

It needs some practice to read and create μ-calculus formulae. However, there are only a few basic structures, from which formulae are built up, and we want to explain them here.
The first aspect to make clear is the difference between the modalities $[a]$ and $\langle a\rangle$. The formula $[a]\Phi$ is **true** at a state for which **necessarily** the a-successors fulfill Φ, the formula $\langle a\rangle\Phi$ is **true** at a state for which **possibly** the a-successors fulfill Φ.

$$\begin{tikzcd}
& s_1 \models Q & & t_1 \models Q & & \\
s_0 \arrow[ur, "a"] \arrow[r, "a"] \arrow[dr, "b"'] & s_2 \models Q & t_0 \arrow[ur, "a"] \arrow[r, "a"] \arrow[dr, "b"'] & t_2 \models \neg Q & u_0 \arrow[r, "b"] & u_3 \\
& s_3 & & t_3 & &
\end{tikzcd}$$

For the first transition system, we have $s_0 \models \langle a\rangle Q$ and $s_0 \models [a]Q$. For the second one it is $t_0 \models \langle a\rangle Q$, because t_0 has an a-successor fulfilling Q, but $t_0 \not\models [a]Q$, as t_0 also has an a-successor fulfilling $\neg Q$. In the third transition system we get $u_0 \not\models \langle a\rangle Q$, due to the absence of an a-successor, but $u_0 \models [a]Q$, because there is no a-successor **not** fulfilling Q.

Crucial is the difference between least and greatest fixpoints. In general, least fixpoints describe finite behaviour, greatest fixpoints additionaly also infinite behaviour. In the transition systems below we assume $\neg Q$ at each state, where the opposite is not stated explicitly.

$$\begin{tikzcd}
v_1 \arrow[r, bend left, "a"] & v_2 \models Q \arrow[l, bend left, "a"]
\end{tikzcd}
\qquad
\begin{tikzcd}
w_1 \arrow[r, bend left, "a"] & w_2 \arrow[l, bend left, "\mathrm{a}"] \arrow[r, "a"] & w_3 \models Q \arrow[loop right, "a"]
\end{tikzcd}$$

The formula $\mu Z.\langle a\rangle Z \vee Q$ stands for the property "there exists an a-path on which eventually Q will hold". It holds both states v_1 and w_1. The universal counterpart "on all paths eventually Q will hold", expressed by $\mu Z.[a]Z \vee Q$, holds for v_1, but not for w_1, because on the infinite path $w_1 w_2 w_1 w_2 \ldots$ the atomic proposition Q never holds. Considering the semantic definition of a least fixpoint, we have that the subset $\{w_3\}$ satisfies the inequality:

$\|[a]X \vee Q\|^{\mathcal{T}}_{\mathcal{V}[Z/\{w_3\}]} \;=\; [\![a]\!]^{\mathcal{T}}\{w_3\} \cup \{w_3\} \;=\; \emptyset \cup \{w_3\} \;\subseteq\; \{w_3\}$

It follows that in the intersection of all sets satisfying the inequality w_1 is not contained, and hence is not an element of the least fixpoint, which coincides with the informal argumentation above.

With a greatest fixpoint, the proposition "on all a-paths always Q holds" can be formulated as $\nu Z.[a]Z \wedge Q$. In the transition systems above it only holds at w_3.

Combining least and greatest fixpoints allows us to express more complicated properties. The formula $\nu Z.\mu X.(([a]Z \wedge Q) \vee [a]X)$ corresponds to the proposition "on all infinite a-paths infinitely often Q holds". To make the structure plausible, consider the two fragments of the formula $\mu X.(([a]Z \wedge Q) \vee [a]X)$ saying "eventually $([a]Z \wedge Q)$ will hold" and $\nu Z.([a]Z \wedge Q)$ saying "always Q will hold". Combining them in one formula gives "always, either Q holds, or, if it does not, then eventually Q will hold" which is equivalent to the first explanation above. In the transition systems above, this formula satisfies v_1 and w_3. The existential version $\nu Z.\mu X.((\langle a\rangle Z \wedge Q) \vee \langle a\rangle X)$ holds also for w_1.

As last example consider the property "eventually Q will always hold", expressed by the formula $\mu X.\nu Z.(([a]Z \land Q) \lor [a]X)$. Here, the least fixpoint is the outermost and the difference to the previous formula lies in the order of "always eventually" and "eventually always".

In chapter 7 we use some more sophisticated formulae, but they are explainable with the basic examples discussed in this section.

4.3 Properties of the modal μ-calculus.

The modal μ-calculus is a **branching time** logic, in that at each state all successors or some successor may be considered. In fact, this is the only way of looking at the next state within a path. It stands in contrast to **linear time** (temporal) logic. There, the models are sets of runs and in each run there is one unique successor for each state. A branching time property which cannot be expressed in linear time is: it is always possible to continue in such a way that eventually Q holds.

Expressiveness

The modal μ-calculus subsumes many other temporal logics, such as Propositional Dynamic Logic (PDL) [FR79], PDL-Δ [Str82], Computation Tree Logic (CTL) [CE81], its extended versions CTL* [EH86], and ECTL* [VW83], Hennessy-Milner logic [HM85], and linear time μ-calculus.

However, translations from these logics (apart from Hennessy-Milner logic) into modal μ-calculus are non-trivial, e.g. for CTL* is it double-exponential, for ECTL* the translation is single-exponential [Dam92], as it is also for linear time μ-calculus.

The fragment L_1 of the modal μ-calculus consists of formulae which contain only disjunctions, *diaa* next step operators, and, as in the general case, constants, variables and the fixpoint operators. The fragment L_2 includes L_1 and allows conjunctions and $[a]$-operators in a restricted form: they may be applied only to closed subformulae. In [EJS93] L_2 was shown to be exactly as expressive as ECTL*. In the definition below Φ_2^c denotes a formula that is closed and generated by the grammar.

The set of formulae of L_2 is defined as:

$$\Phi_2 ::= Q \mid Z \mid \Phi_2 \vee \Phi_2 \mid \Phi_2 \wedge \Phi_2^c \mid \langle a\rangle\Phi_2 \mid [a]\Phi_2^c \mid \mu Z.\Phi_2 \mid \nu Z.\Phi_2$$

For a long time it was not known, whether alternation depth of more than 3 increases the expressiveness of the modal μ-calculus. Bradfield [Bra96] showed the strictness of the alternation hierarchy by transforming it to the mu-arithmetic hierarchy. Independently, Lenzi [Len96] proved the same result.

Comparing modal logics with propositional logic gives the following relations: in modal logic without fixpoints first order properties can be expressed, but modal logic lies strictly between propositional logic and first order logic. Adding fixpoint operators to modal logic shifts the expressive power beyond first order: the modal μ-calculus lies between first and second order logic.

Kozen and Parikh [KP83] reduced the modal μ-calculus to SnS, the monadic second order theory of n successors. In [Hüt90] Hüttel showed it to be equi-expressive to SnS.

Axiomatization

Kozen [Koz83] gave an axiomatization for a fragment of the modal μ-calculus (the aconjunctive fragment). For a long time the axiomatization for the full modal μ-calculus was an open question.

Walukiewicz [Wal95b] showed the completeness of Kozen's axiomatization for the full modal μ-calculus. Ambler, Kwiatkowska, Measor and Bonsangue ([AKM95], [BK95]) proved the same result, and independently also Hartonas [Har95], by means of modal duality theory.

Decidability

The question of decidability is: given a formula of the modal μ-calculus, does there exist a model for it?

From the reduction of the modal μ-calculus to SnS [KP83] the decidability follows giving a non-elementary decision procedure.

In [Koz88] Kozen proved a finite model theorem for the modal μ-calculus, saying that every formula having a model has also a finite model. He also gives a nonelementary decision procedure.

In [EJ88] Emerson and Jutla showed that decidability of the modal μ-calculus has deterministic exponential time complexity. By a reduction from alternating polynomial space Turing automata it follows that the problem is EXPTIME complete [Eme96].

Model Checking

The model checking problem is: Given a model and a formula of the modal μ-calculus does initial state of the transition system satisfy the formula?

Many authors restrict the models to finite ones. We want to consider models with both finite and infinite state spaces. Chapters 5 to 8 will deal with finite state space model checking, chapter 9 with the infinite case.

The **size** of the model checking problem is defined as $|\Phi| \times |\mathcal{T}|$, where Φ is a formula and $\mathcal{T}$ a transition system.

Zhang, Sokolsky and Smolka [ZSS94] showed that finite state space model checking is P-hard, even for the alternation free fragment. It follows from Emerson and Lei's [EL86] polynomial algorithm for fragments with restricted alternation depth that for these fragments model checking is P-complete.

Kalorkoti [Kal96] pointed out that a monotone Boolean circuit can trivially be expressed as a set of Boolean equations (with any fixpoint operators), and P-hardness follows from the equivalence of the model checking problem and solving Boolean equation systems, which will be proved in chapter 5.

The best known upper bound for model checking in the unrestricted μ-calculus is NP $\cap$ co-NP, proved by Emerson, Jutla and Sistla [EJS93]. Section 6.5 contains a proof of this result in our framework.

In figure 4.1 below, we illustrate the model checking problem.

The set of μ-calculus formulae L_μ factorized by the equivalence relation $\Leftrightarrow$ forms a lattice, where formulae are ordered by implication, known as the Lindenbaum algebra of L_μ. The semantic function $[\![\ ]\!] : L_\mu \rightarrow \mathfrak{P}(\mathcal{S})$
monotone and maps each formula to the set of states that satisfy the formula. The powerset $\mathfrak{P}(\mathcal{S})$ is a complete lattice.

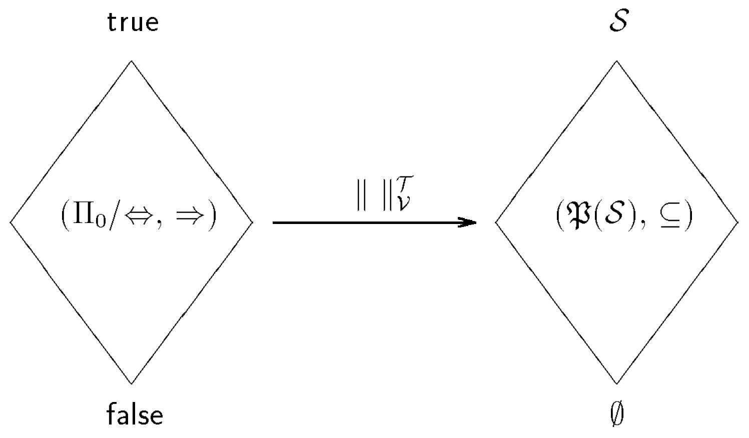

true
$(\Pi_0/\Leftrightarrow, \Rightarrow)$
false
$\| \ \|_{\mathcal{V}}^{\mathcal{T}}$
$\mathcal{S}$
$(\mathfrak{P}(\mathcal{S}), \subseteq)$
$\emptyset$

Figure 4.1. Lattices of the modal μ-calculus and its semantics

If $\| \ \|_{\mathcal{V}}^{\mathcal{T}}$ were continuous, we could immediately derive that $(L_\mu/ \Leftrightarrow)$ would also be a complete lattice. However, this is not the case, and one example for non-continuity is $[a]\Phi$, where $\|[a]\Phi\|_{\mathcal{V}}^{\mathcal{T}} = [\![a]\!]^{\mathcal{T}} \ \|\Phi\|_{\mathcal{V}}^{\mathcal{T}}$ and $[\![a]\!]^{\mathcal{T}}$ is only continuous for transition systems with finite branching degree (see [Sti93] p.499).

One strategy to solve the model checking problem is to determine the set of states for which the formula given holds, and then to check whether the initial state is an element of this set. This approach is called **global model checking**. The strategy of **local model checking** tries to answer the question directly for the initial state.

Chapter 5

Boolean equation systems for model checking.

The main interest of this chapter is to show the equivalence of the model checking problem for the modal μ-calculus and the problem of solving Boolean equation systems. Several authors have reduced the model checking problem into Boolean equation systems: Arnold and Crubille [AC88], Andersen [And92], Larsen [Lar92], Vergauwen and Levi [VL94] and others. However, they mainly derive Boolean equation systems for the case of simple fixpoints. One reason is, that the approximation scheme using backtracking, the most well known algorithm giving a solution to (nested) fixpoint expressions, requires subsequently solving simple fixpoint expressions. Therefore, there is no need for defining fixpoint-equation systems with nested and alternating fixpoint operators. In contrast to this we want to investigate the general case of Boolean equation systems independently from any algorithm. Existing model checking algorithms can now be interpreted as algorithms for solving Boolean equation systems and vice versa. Furthermore, we have a number of useful properties of fixpoint-equation

systems collected in chapter 3. They allow us to derive new algorithms and help to give a clearer understanding of the basic problem. Moreover, the equivalence to other frameworks solving the model checking problem will be shown, as they can be found in automata theory and game theory (chapter 8).
Section 5 contains the reduction of the model checking problem. It consists of a syntactical mapping from a μ-calculus formula and a model to a Boolean equation system, and the proof that the formula holds at the initial state of the transition system, iff the Boolean equation system derived has the solution true for a corresponding variable. In section 5.2 it will be shown that there exists a reduction which is linear in the size of the formula as well as in the size of the transition system. A polynomial reduction from Boolean equation systems to model checking problems is presented in section 5.2.

5.1 Reduction of the model checking problem.

The transformation function $\mathbf{E}$ maps a pair $(\Phi, \mathcal{M})$ consisting of a modal μ-calculus formula Φ and a model $\mathcal{M}$ to a Boolean equation system.
$\mathbf{E}$ refers to a set of functions $\{\mathbf{E_1}, \ldots \mathbf{E_n}\}$, where each $\mathbf{E_i}$, for $1 \leq i \leq n$, is related to state s_i of the transition system.
Roughly the function $\mathbf{E}$ is responsible for the linearization of a nested fixpoint formula, whereas the function $\mathbf{E_i}$ maps a modal μ-calculus formula to a Boolean expression at state s_i. We will omit the second argument $\mathcal{M}$ of $\mathbf{E}$ when it is clear from the context. Note, that the transformation is defined for formulae having a fixpoint as outermost operator. A formula Φ not in this form can easily transformed to an equivalent formula $\sigma X.\Phi$ by addition of an effectless fixpoint operator, where X is not free in Φ (see proposition 2.17(5)).

$$\begin{array}{rcl}
\mathbf{E}(Q) & = & \epsilon \\
\mathbf{E}(X) & = & \epsilon \\
\mathbf{E}(\Phi_1 \wedge \Phi_2) & = & \mathbf{E}(\Phi_1)\ \mathbf{E}(\Phi_2) \\
\mathbf{E}(\Phi_1 \vee \Phi_2) & = & \mathbf{E}(\Phi_1)\ \mathbf{E}(\Phi_2) \\
\mathbf{E}([a]\Phi) & = & \mathbf{E}(\Phi) \\
\mathbf{E}(\langle a\rangle\Phi) & = & \mathbf{E}(\Phi) \\
\mathbf{E}(\sigma X.\Phi) & = & (\sigma X_1 = \mathbf{E}_1(\Phi)) \ldots (\sigma X_n = \mathbf{E}_n(\Phi))\ \mathbf{E}(\Phi)
\end{array}$$

and for $1 \leq i \leq n$

$$\begin{array}{rcl}
\mathbf{E_i}(Q) & = & \begin{cases} \mathsf{true} & \text{if } s_i \in \mathcal{V}(Q) \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\mathbf{E_i}(X) & = & X_i \\
\mathbf{E_i}(\Phi_1 \wedge \Phi_2) & = & \mathbf{E_i}(\Phi_1) \wedge \mathbf{E_i}(\Phi_2) \\
\mathbf{E_i}(\Phi_1 \vee \Phi_2) & = & \mathbf{E_i}(\Phi_1) \vee \mathbf{E_i}(\Phi_2) \\
\mathbf{E_i}([a]\Phi) & = & \bigwedge_{s_i \xrightarrow{a} s_j} \mathbf{E_j}(\Phi) \\
\mathbf{E_i}(\langle a\rangle\Phi) & = & \bigvee_{s_i \xrightarrow{a} s_j} \mathbf{E_j}(\Phi) \\
\mathbf{E_i}(\sigma X.\Phi) & = & X_i
\end{array}$$

For a valuation $\mathcal{V}$ the environment $\theta_{\mathcal{V}}$ is defined as: $\theta_{\mathcal{V}}(X_i) = \mathsf{true}$ iff $s_i \in \mathcal{V}(X)$.

The following reduction theorem shows that the transformation preserves the semantics, i.e. a property satisfies a state in a model iff the corresponding variable in the Boolean equation system derived has the solution true.

Theorem 5.1 Let $\sigma X.\Phi$ be a formula of the modal μ-calculus, $\mathcal{M} = (\mathcal{T}, \mathcal{V})$ a model and s_i a state of $\mathcal{T}$.

Then for all environments $\theta_{\mathcal{V}}$ it is the case that

$s_i \models_{\mathcal{M}} \sigma X.\Phi$ iff $([\![\ \mathbf{E}(\,(\sigma X = \Phi), \mathcal{M}\,)\ ]\!]\ \theta_{\mathcal{V}})\,(X_i) = \mathsf{true}$.

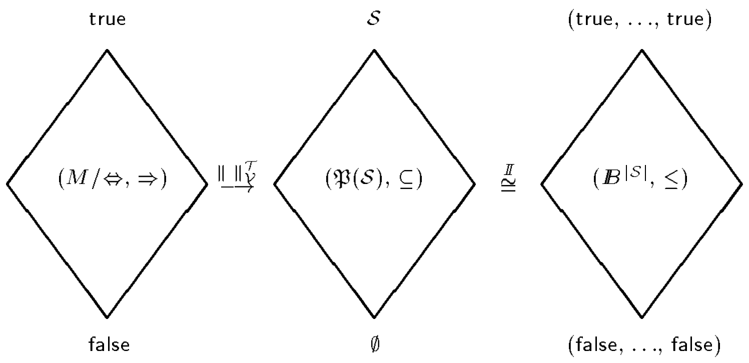
A proof can be found in the appendix. A motivation for it will be after the following example.

Example: Consider the transition system depicted, and let Q hold for s_2, but not for s_1, i.e. $\mathcal{V}(Q) = \{s_2\}$. The proposition we want to prove is "on some infinite a-path Q holds infinitely often", $\nu X.\mu Y.\langle a\rangle((Q \wedge X) \vee Y)$. The reduction to a Boolean equation system is:

$$\begin{tikzcd} s_1 \arrow[rr, bend left, "\mathbf{a}"] & & s_2 \arrow[ll, bend left, "\mathbf{a}"] \end{tikzcd}$$

$$\begin{aligned}
&\mathbf{E}(\nu X.\mu Y.\langle a\rangle((Q \wedge X) \vee Y)) \\
&= (\nu X_1 = \mathbf{E}_1(Y))\ \ (\nu X_2 = \mathbf{E}_2(Y))\ \ \mathbf{E}(\mu Y.\langle a\rangle((Q \wedge X) \vee Y) \\
&= (\nu X_1 = Y_1)\ \ (\nu X_2 = Y_2)\ \ (\mu Y_1 = \mathbf{E}_1(\langle a\rangle((Q \wedge X) \vee Y))) \\
&\quad\ (\mu Y_2 = \mathbf{E}_2(\langle a\rangle((Q \wedge X) \vee Y)))\ \ \mathbf{E}(\langle a\rangle((Q \wedge X) \vee Y)) \\
&= (\nu X_1 = Y_1)\ \ (\nu X_2 = Y_2) \\
&\quad\ (\mu Y_1 = \mathbf{E}_2((Q \wedge X) \vee Y)))\ \ (\mu Y_2 = \mathbf{E}_1((Q \wedge X) \vee Y))) \\
&\quad\ \mathbf{E}((Q \wedge X) \vee Y) \\
&= \ \ldots \\
&= (\nu X_1 = Y_1)\ (\nu X_2 = Y_2) \\
&\quad\ (\mu Y_1 = (\mathsf{true} \wedge X_2) \vee Y_2)))\ (\mu Y_2 = (\mathsf{false} \wedge X_1) \vee Y_1))) \\
&= (\nu X_1 = Y_1)\ (\nu X_2 = Y_2)\ (\mu Y_1 = X_2 \vee Y_2)\ \ (\mu Y_2 = Y_1) \qquad \lhd
\end{aligned}$$

The proof of theorem 5.1 will take several intermediate steps. Roughly a μ-calculus formula has to be mapped to a μ-calculus equation system. Then the latter is mapped to a equation system on the power set of the state space, where modal operators are mapped to set operators etc. The last step reflects the isomorphism between sets and Boolean vectors. For the base case of expressions the situation can be illustrated as follows: Recall that Π_0 is the set of fixpoint-free expressions of the modal μ-calculus, i.e., the expressions of the propositional modal logic. The equivalence classes of Π_0 together with the implication ordering form a lattice $(\Pi_0/\Leftrightarrow, \Rightarrow)$, the Lindenbaum algebra of Π_0.



true
$\mathcal{S}$
(true, ..., true)
$(M/\Leftrightarrow, \Rightarrow)$
$\xrightarrow{\|\ \|_{\mathcal{V}}^{\mathcal{T}}}$
$(\mathfrak{P}(\mathcal{S}), \subseteq)$
$\overset{\mathcal{I}}{\cong}$
$(\mathbb{B}^{|\mathcal{S}|}, \leq)$
false
$\emptyset$
(false, ..., false)

Figure 5.1. Lattices for modal μ-calculus, state space and Boolean vector space

The powerset of the state space $\mathcal{S} = \{s_1, \ldots, s_n\}$ with the inclusion order forms a complete lattice $(\mathfrak{P}(S), \subseteq)$. The evaluation function $\|\ \|_{\mathcal{V}}^{\mathcal{T}} : \Pi_0 \to \mathfrak{P}(S)$ is monotone (and continuous). The extension of the evaluation function from Π_0 to expressions over Π_0 maps modal operators $[a], \langle a \rangle$ to set operators $[\![a]\!]^{\mathcal{T}}, \langle\!\langle a \rangle\!\rangle^{\mathcal{T}}$, modal variables to set variables and the logical operators $\wedge, \vee$ to the set operators $\cap, \cup$. Thus we get an expression over the powerset of the state space. Defining false $\leq$ true the Boolean lattice $(\mathbb{B}^{|\mathcal{S}|}, \leq^{|\mathcal{S}|})$ with pointwise ordering is isomorphic to $(\mathfrak{P}(S), \subseteq)$. The last step leads from a vector expression in $\mathbb{B}^n$ to a Boolean equation system; a vector expression is split into n expressions and the operators $[\![a]\!]^{\mathcal{T}}, \langle\!\langle a \rangle\!\rangle^{\mathcal{T}}$ are evaluated.

5.2 Representation and complexity.

A straightforward application of the transformation $\mathbf{E}$ may lead to a Boolean equation system of exponential size in the nesting depth of modal operators. The problem is discussed in e.g. [And92]: an equation in μL_1 of the form $\sigma_i X_i = \langle a \rangle [a] \ldots \langle a \rangle [a] X_j$ with l modal operators is transformed to $|\mathcal{S}|$ equations of the form $\bigvee_{I_1} \bigwedge_{I_2} \cdots \bigvee_{I_{l-1}} \bigwedge_{I_l} X_k$ for some index sets $I_1, \ldots, I_l$. Obviously the size of this expression is bounded by $|\longrightarrow|^l$, where $|\longrightarrow|$ is the branching degree of the underlying transition system. The upper bound for the branching degree is the size of the state space $|\mathcal{S}|$.

In order to avoid such blow up Arnold and Crubille [AC88] suggested to transform μ-calculus equations into **simple form**, i.e. each right hand side consists of a disjunct $X_i \vee X_j$ or a conjunct $X_i \wedge X_j$ or one modal operator in front of a variable $[a]X$ or $\langle a \rangle X$. The transformation is done by introduction of additional variables. For the general case of nested fixpoint operators in proposition 3.25 the correctness of introducing new variables and equations is proved for disjunctions and conjunctions. The correctness of introducing new variables and equations for modal operators can be shown similarly to the proof of lemma 3.25.

Using this technique the size of a Boolean equation system resulting from the transformation **E** from a modal μ-calculus formula Φ and a model $(\mathcal{T}, \mathcal{V})$ is bound by $O(|\Phi||\mathcal{T}|)$. A discussion of the representation assumptions for this result can be found in Andersen [And92]. The Boolean equation system derived from a model checking problem is then also in simple form as defined in section 3.2.

Example: Consider a transition system with k states and from each state exists an a-transition to each *other* state (not to itself). The μ-calculus formula is $\nu X.[a]\langle a \rangle X$.

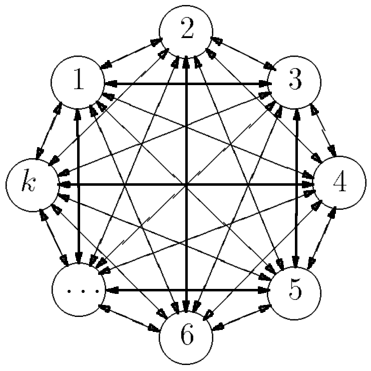


Figure 5.2. Transition system

The Boolean equation system derived from

an untransformed equation
$\nu X = [a]\langle a\rangle X$
has size of $O(k^2)$:

$$
\begin{aligned}
\nu X_1 &= \bigwedge_{j=2}^{k} \bigvee_{i=1, i\neq j}^{k} X_i \\
&\dots \\
\nu X_k &= \bigwedge_{j=1}^{k-1} \bigvee_{i=1, i\neq j}^{k} X_i
\end{aligned}
$$

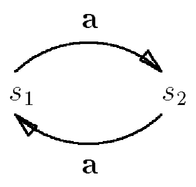
simple form equations
$(\nu X = [a]X')$ $(\nu X' = \langle a\rangle X)$
has size of $O(k)$:

$$
\begin{aligned}
\nu X_1 &= \bigwedge_{j=2}^{k} X'_j \\
\nu X'_1 &= \bigvee_{j=2}^{k} X_j \\
&\dots \\
\nu X_k &= \bigwedge_{j=1}^{k-1} X'_j \\
\nu X'_k &= \bigvee_{j=1}^{k-1} X'_j
\end{aligned}
$$

$\lhd$

It is obvious that nesting depth and alternation depth of a Boolean equation system are not greater than nesting depth and alternation depth of the underlying μ-calculus formula. In dependency of the model these numbers can decrease as the following example will show:

Example: Consider the μ-calculus formula
$\Phi \equiv \nu X.\langle a\rangle \mu Y.[b]X \wedge [a]Y$
and the transition system depicted.
Transformation to a Boolean equation system $\mathcal{E}$ gives:
$(\nu X_1 = Y_2)$ $(\nu X_2 = Y_1)$ $(\mu Y_1 = Y_2)$ $(\mu Y_2 = Y_1)$
Φ has nesting depth and alternation depth 2.
$\mathcal{E}$ has nesting depth 2 and alternation depth 1.



5.3 Reduction of Boolean equation systems.

In order to show that the model checking problem and the problem of solving a Boolean equation system are equivalent we also have to give a transformation in the other direction. For any closed Boolean equation system $\mathcal{E}$ we will construct a formula of the modal μ-calculus Φ and a model $\mathcal{M}$, such that $\mathcal{E}$ and $\mathbf{E}(\Phi, \mathcal{M})$ are equivalent, i.e. for all variables of $\mathcal{E}$ they have the same solution. Roughly, after some reordering of equations and introduction of new equations an equation system is divided into blocks. We define a transition system that consists of as many states as the largest block contains equations. Transitions are defined straightforwardly in such a way, that the transformation $\mathbf{E}$ produces the required expressions.

Theorem 5.2 For a closed Boolean equation system $\mathcal{E}$ there exists a proposition of the modal μ-calculus Φ and a model $\mathcal{M} = (\mathcal{T}, \mathcal{V})$, such that for a variable renaming function ρ on the variables of $\mathcal{E}$, all $X \in lhs(\mathcal{E})$ and environments θ

$$([\![\mathcal{E}]\!]\,\theta)(X) = ([\![\mathbf{E}(\Phi, \mathcal{M})]\!]\,\theta)(\rho(X)).$$

It is $ad(\mathcal{E}) \leq ad(\Phi)$, $\mathcal{T}$ is of size $O(|\mathcal{E}|^2)$ and Φ is of size $O(|\mathcal{E}|^2)$.

Proof: The construction of Φ and a transition system is performed in seven steps. We assume that the Boolean equation system $\mathcal{E}$ is in standard form.

(1) Divide $\mathcal{E}$ into blocks, such that consecutive blocks have different fixpoint operators and within one block there is a unique fixpoint operator.

(2) Within each block move all disjunctions to the top and the conjunctions to the bottom according to theorem 3.21. Now divide each block into two new blocks, such that one contains no disjunctions (called conjunctive block) and the other one contains no conjunctions (called disjunctive block).

(3) Introduce a new variable for each block and if X is the variable of a block rename all the left-hand side variables of this block to $X_1, \ldots, X_j$ for some $j \in \mathbb{N}$. Let ρ be the injective renaming function which maps an "old" variable to a "new" one.

(4) Transform the Boolean equation system into an equivalent one, $\mathcal{E}'$ in the following way. Assume there is occurrence of a variable X on the right hand side of an equation $\sigma Y = f$, where $Y \trianglelefteq X$ and X is not a variable of the same block as Y and not of the directly subsequent block. Then introduce a new variable X', transform the equation above to $\sigma Y = f[X/X']$ and add the equation $\sigma' X' = X$ to the directly subsequent block, where σ' is the fixpoint operator of this block. Continue with introduction of new variables until there is no occurrence of a variable which belongs to a subsequent, but not directly subsequent block. Choose names of new variables, such that within one block there is still a unique variable name and consequent variables are numbered by consequent indices. The transformation is correct according to lemmata 3.25 and 3.22. The additional blow-up of the Boolean equation system is not more than $O(n^2)$ for $n \overset{\text{def}}{=} \mid \mathcal{E} \mid$, because in each block of $\mathcal{E}'$ cannot be added more equations than the number of right-hand side variables in the preceding blocks of $\mathcal{E}$.

(5) If n is the highest index appearing in one of the blocks then create a transition system $\mathcal{T}$ consisting of n states numbered 1 to n. Define a set of action labels, such that for each ordered pair of (block) variables (X, Y) there exists a unique label x_y. Transform the equations and add labelled transitions to the transition system as follows. Let $1 \leq i, j, k \leq n$.

for equations of a disjunctive block,			add to $\mathcal{T}$ transition(s)
$\sigma_i X_i = Y_j \vee Z_k$	$\leadsto$	$\sigma_i X_i = \langle x_y \rangle Y \vee \langle x_z \rangle Z$	$i \xrightarrow{x_y} j,\ i \xrightarrow{x_z} k$
$\sigma_i X_i = Y_j$	$\leadsto$	$\sigma_i X_i = \langle x_y \rangle Y$	$i \xrightarrow{x_y} j$

for equations of a conjunctive block,			add to $\mathcal{T}$ transition(s)
$\sigma_i X_i = Y_j \wedge Z_k$	$\leadsto$	$\sigma_i X_i = [x_y] Y \wedge [x_z] Z$	$i \xrightarrow{x_y} j,\ i \xrightarrow{x_z} k$
$\sigma_i X_i = Y_j$	$\leadsto$	$\sigma_i X_i = [x_y] Y$	$i \xrightarrow{x_y} j$

The transformation does not increase the size of the equation system (apart from addition of modalities).

(6) Create a sequence of expressions, one for each block. For each disjunctive block with variables $X_1, \ldots, X_k$ define

$$\Phi_X \overset{\text{def}}{=} \bigvee_{i=1}^{k} \{\Phi_i | \sigma X_i = \Phi_i \text{ is an equation}\}.$$

Note that according to the choice of action labels each variable appears at most once in Φ_X (assuming that $\langle x_y \rangle Y \vee \langle x_y \rangle Y$ is reduced to $\langle x_y \rangle Y$). Create the expression $\sigma X.\Phi_X$.

Dually, for each conjunctive block with variables $X_1, \ldots, X_k$ define

$$\Phi_X \overset{\text{def}}{=} \bigwedge_{i=1}^{k} \{\Phi_i | \sigma X_i = \Phi_i \text{ is an equation}\}.$$

Again according to the choice of action labels each variable appears at most once in Φ_X. Create the expression $\sigma X.\Phi_X$.

The size of $\sigma X.\Phi_X$ is linear in the number of blocks of $\mathcal{E}'$.

(7) By construction the sequence of expressions has the property: in $\sigma X.\Phi_X$ occur only left-hand side variables from preceding expressions or from the directly subsequent one. Generate an expression Φ starting with the first expression of the sequence and the iteratively substituting the variable which is left-hand side variable of the next expression by the next expression. Show that $\mathcal{E}'$ is a subsystem of $\mathbf{E}(\Phi, \mathcal{M})$, where $\mathcal{M}$ consists of the transition system $\mathcal{T}$ and an arbitrary valuation (this is, because the formulae constructed do not contain atomic propositions).

The size of Φ is linear in the size of all Φ_X, and hence quadratic in the number of blocks of $\mathcal{E}'$. Then, altogether, the size of Φ is $O(b^2)$, where b is the number of blocks in $\mathcal{E}$, and the transition system $\mathcal{T}$ has at most n states and $2n$ transitions, where n is the number of equations in $\mathcal{E}$. □

It is easy to show that $\mathcal{E}'$ is a subsystem of $\mathbf{E}(\Phi, \mathcal{M}_\mathcal{T})$, where $\mathcal{M}_\mathcal{T} = (\mathcal{T}, \mathcal{V})$. The valuation $\mathcal{V}$ can be chosen arbitrarily, because the formulae constructed do not contain constants.

Remark 5.3 The number of action labels is quadratic in the number of blocks of $\mathcal{E}$, but does not depend on the size of blocks. When considering infinite Boolean equation systems we assume that the number of blocks is finite, but within each block there may be an infinite number of equations. The transformation for the infinite case then works as the one for the finite case, only the transition system will have an infinite number of states. The formula Φ has size quadratic in the number of blocks.

Example: step 1: Boolean equation system

$$\begin{array}{|lcccc|}
\hline
\mu Z_1 & = & Z_3 & \vee & Z_5 \\
\mu Z_2 & = & Z_4 & \wedge & Z_6 \\
\hline
\nu Z_3 & = & Z_1 & \vee & Z_6 \\
\hline
\mu Z_4 & = & Z_2 & \vee & Z_5 \\
\mu Z_5 & = & Z_3 & \wedge & Z_2 \\
\mu Z_6 & = & Z_4 & \vee & Z_3 \\
\hline
\end{array}$$

step 2: block structure

$$\begin{array}{|lcccc|}
\hline
\mu Z_1 & = & Z_3 & \vee & Z_5 \\
\hline
\mu Z_2 & = & Z_4 & \wedge & Z_6 \\
\hline
\nu Z_3 & = & Z_1 & \vee & Z_6 \\
\hline
\mu Z_4 & = & Z_2 & \vee & Z_5 \\
\mu Z_6 & = & Z_4 & \vee & Z_3 \\
\hline
\mu Z_5 & = & Z_3 & \wedge & Z_2 \\
\hline
\end{array}$$

step 3: renaming

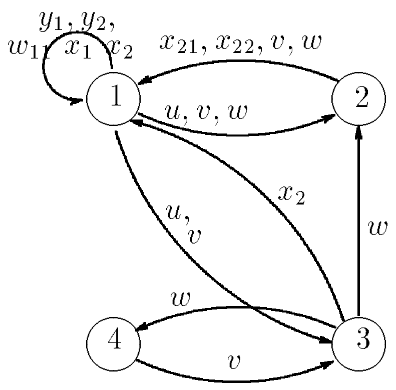
$$\begin{array}{|lcccc|}
\hline
\mu U_1 & = & W_1 & \vee & Y_1 \\
\hline
\mu V_1 & = & X_1 & \wedge & X_2 \\
\hline
\nu W_1 & = & U_1 & \vee & X_2 \\
\hline
\mu X_1 & = & V_1 & \vee & Y_1 \\
\mu X_2 & = & X_1 & \vee & W_1 \\
\hline
\mu Y_1 & = & W_1 & \wedge & V_1 \\
\hline
\end{array}$$

step 4: introduction of additional variables

$$\begin{array}{|lcccc|}
\hline
\mu U_1 & = & V_2 & \vee & V_3 \\
\hline
\mu V_1 & = & W_2 & \wedge & W_3 \\
\mu V_2 & = & W_1 & & \\
\mu V_3 & = & W_4 & & \\
\hline
\nu W_1 & = & U_1 & \vee & X_2 \\
\nu W_2 & = & X_1 & & \\
\nu W_3 & = & X_2 & & \\
\nu W_4 & = & X_3 & & \\
\hline
\mu X_1 & = & V_1 & \vee & Y_1 \\
\mu X_2 & = & X_1 & \vee & W_1 \\
\mu X_3 & = & Y_1 & & \\
\hline
\mu Y_1 & = & W_1 & \wedge & V_1 \\
\hline
\end{array}$$

step 5: creating equations and a transition system

μU_1	$=$	$\langle u_v\rangle V$	$\vee$	$\langle u_v\rangle V$
μV_1	$=$	$[v_w]W$	$\wedge$	$[v_w]W$
μV_2	$=$	$[v_w]W$		
μV_3	$=$	$[v_w]W$		
νW_1	$=$	$\langle w_u\rangle U$	$\vee$	$\langle w_x\rangle X$
νW_2	$=$	$\langle w_x\rangle X$		
νW_3	$=$	$\langle w_x\rangle X$		
νW_4	$=$	$\langle w_x\rangle X$		
μX_1	$=$	$\langle x_v\rangle V$	$\vee$	$\langle x_y\rangle Y$
μX_2	$=$	$\langle x_x\rangle X$	$\vee$	$\langle x_w\rangle W$
μX_3	$=$	$\langle x_y\rangle Y$		
μY_1	$=$	$[y_w]W$	$\wedge$	$[y_v]V$



step 6: create one expression for each block

$\mu U.\langle u_v\rangle V$
$\mu V.[v_w]W$
$\nu W.\langle w_u\rangle U \vee \langle w_x\rangle X$
$\mu X.\langle x_v\rangle V \vee \langle x_x\rangle X \vee \langle x_w\rangle W \vee \langle x_y\rangle Y$
$\mu Y.[y_w]W \wedge [y_v]V$

step 7: generate one expression

$$\begin{array}{l}
\mu U.\langle u_v\rangle(\\
\qquad \mu V.[v_w](\\
\qquad\qquad \nu W.\langle w_u\rangle U \vee \langle w_x\rangle(\\
\qquad\qquad\qquad \mu X.\langle x_v\rangle V \vee \langle x_x\rangle X \vee \langle x_w\rangle W \vee \langle x_y\rangle(\\
\qquad\qquad\qquad\qquad \mu Y.[y_w]W \wedge [y_v]V \qquad\qquad\qquad\qquad))))
\end{array}$$

translate formula and transition system created back to a Boolean equation system

μU_1	$=$	V_2	$\vee$	V_3
μU_2	$=$	false		
μU_3	$=$	false		
μU_4	$=$	false		
μV_1	$=$	W_2	$\wedge$	W_3
μV_2	$=$	W_1		
μV_3	$=$	W_4		
μV_4	$=$	true		
νW_1	$=$	U_1	$\vee$	X_2
νW_2	$=$	X_1		
νW_3	$=$	X_2		
νW_4	$=$	X_3		
μX_1	$=$	V_1	$\vee$	Y_1
μX_2	$=$	X_1	$\vee$	W_1
μX_3	$=$	Y_1		
μX_4	$=$	false		
μY_1	$=$	W_1	$\wedge$	V_1
μY_2	$=$	false		
μY_3	$=$	false		
μY_4	$=$	false		

$\triangleleft$

Chapter 6

Solving Boolean equation systems.

In this chapter we will illuminate various methods for solving Boolean equation systems. All of them are in fact model checking algorithms. Usually they are presented within different settings. Here they are all discussed within one framework. This allows a clearer understanding of concepts.

We distinguish two basic classes of methods, the global ones and the local ones. The global ones require the full Boolean equation system and their result is a complete solution for all variables. The local ones try to determine a subset of equations which gives sufficient information to calculate the solution for the single variable which is of interest. (Usually it is the variable which corresponds to the initial state of the transition system and the property to prove.) The worst case complexity of local algorithms can never be better than the one of global algorithms: in the worst case the whole equation system is involved in the solution for the first variable. However, in some average case it is likely that just a subset of the equations contains sufficient information and therefore local methods might have better average case complexity. Traditionally, approximation techniques (see section

6.2) belong to the global algorithms, tableau methods (see section 6.3) to the local ones. However, borders between the approaches are not strict. There exists an approximation based algorithm which works locally; the Gauß elimination algorithm (see section 6.4) exists in both versions.

In chapter 8 we will consider other frameworks, in which there exist problems equivalent to solving Boolean equation systems. Of course, algorithms solving an equivalent problem also solve Boolean equation systems.

6.1 Plain Boolean equation systems.

For the moment we consider closed Boolean equation systems in simple form without any minimality and maximality conditions, i.e. we just forget about the σs. The remaining system $\mathcal{E}_p$ is not an ordered set of Boolean equations of the form $X_i = f_i$ for some $1 \leq i \leq n$. A solution (or fixpoint) of $\mathcal{E}_p$ is an environment $\theta : lhs(\mathcal{E}_p) \rightarrow \mathbb{B}$, such that for each equation $X_i = f_i$ it is $f_i(\theta) = \theta(X_i)$. An equation system $\mathcal{E}_p$ can easily be transformed into a Boolean function of the form $\bigvee_{1 \leq i \leq n}(f_i \wedge X_i') \vee (f_i' \wedge X_i) = 0$. It is a well studied area what the solutions of such a function are (see for example [Rud74]).

The condition that all f_is are monotone ensures that the set of all solutions of the Boolean function form a complete lattice. The number of solutions is in general exponential in the number of equations.

From a plain Boolean equation system $\mathcal{E}_p$ we can derive two sorts of graphs: the **order graph** telling order conditions for the variables in every solution and the **dependency graph** showing the interdependency of the variables in the system.

The order graph is a representation of order conditions derived from the equations. It consists of a set of vertices $\{1, \ldots, n, \mathsf{true}, \mathsf{false}\}$, one vertex for each equation of the system $\mathcal{E}_p$ and two for the Boolean constants. If there is an equation $X_i = X_j \wedge X_k$ in $\mathcal{E}_p$ then for every solution of $\mathcal{E}_p$ it is the case that $X_i \leq X_j$ and $X_i \leq X_k$. Hence there will be the edges $j \rightarrow i$ and $k \rightarrow i$ in the order graph. Dually, if

$X_i = X_j \vee X_k$ then $X_i \geq X_j$ and $X_i \geq X_k$ and the order graph contains the edges $i \to j$ and $i \to k$. Cycles in the order graph indicate that all variables in the cycle have to be equal in every solution. However, there exist environments fulfilling all conditions of the order graph, but not being solutions of the system. For example $\theta(X_i) = \mathsf{false}$, $\theta(X_j) = \mathsf{true}$ and $\theta(X_k) = \mathsf{true}$ fulfills the order conditions derived from $X_i = X_j \wedge X_k$, but is not a solution of the equation.

The dependency graph of a plain Boolean equation system $\mathcal{E}_p$ also has the vertices $\{1, \ldots, n, \mathsf{true}, \mathsf{false}\}$. It is a representation of the dependency relations derived form the equations. For an equation $X_i = X_j \wedge X_k$ or $X_i = X_j \vee X_k$ the dependency graph contains the edges $j \to i$ and $k \to i$. The information we can get from the dependency graph is for example about the nesting structure of the equations. Parts of the graph which are not strongly connected indicate that the underlying system can be decomposed in parts which can be solved one after the other.

Our question now is what is the solution we are interested in, when we add to our equation system minimality and maximality conditions and order.

For Boolean equation systems with only maximal fixpoints or only minimal fixpoints Andersen [And94a] investigated dependency graphs (boolean graphs in his terminology) and derived efficient algorithms for determining the maximal, or minimal resp. fixpoint.

For the case of nested and alternating maximal and minimal fixpoints things get more complicated. Clearly, the solution of the system with fixpoint operators is one of the solutions of the related plain Boolean equation system. Now an interesting question is, which one of the solutions of the plain Boolean equation system is the one we want? A first idea is that it is the lexicographically least solution of the plain system. The lexicographic order is derived from the fixpoint operators as in definition 3.4 and the characterization of the solution from proposition 3.5 suggests such an idea. The first example below will show, that this is not the case. The second example will show that it is even worse. There we present two Boolean equation systems, both having the same fixpoint operators in the same order, and both having the set of solutions for their plain version. However, their solutions differ. This indicates, that the set of fixpoints of the plain system and the fixpoint operators do not provide enough information to select the solution. All algorithms we will discuss in this chapter have to determine the solutions of the subsystems first (in some abstract view). This is an argument for that the traditional methods for solving plain Boolean equation systems do not help in the case here.

Example: Let $(\mu X_1 = X_2)(\nu X_2 = X_2)$ be a Boolean equation system. There exist two environments fulfilling the both conditions above:
$\theta_1 = \theta[X_1/\mathsf{true}][X_2/\mathsf{true}]$ and $\theta_2 = \theta[X_1/\mathsf{false}][X_2/\mathsf{false}]$. For both, i=1,2, it is $(X_1)(\theta_i) = \theta_i(X_1)$ and $(X_2)(\theta_i) = \theta_i(X_2)$.
However, the solution of $[\![\nu X_2.X_2]\!]\,\theta[X_1/\theta_i(X_1)]$ is $X_2 = \mathsf{true}$ for both environments, i=1,2. Hence the solution of the whole system is θ_1, i.e. $X_1 = \mathsf{true}, X_2 = \mathsf{true}$, whereas the lexicographic least fixpoint is $X_1 = \mathsf{false}$, $X_2 = \mathsf{false}$. $\lhd$

Example: The plain equation system $X_1 = X_2$, $X_2 = X_2$, has the solutions $(\mathsf{true}, \mathsf{true})$ and $(\mathsf{false}, \mathsf{false})$. The solution for the Boolean equation system is $([\![(\nu X_1 = X_2)(\mu X_2 = X_2)]\!]\,\theta)(X_i) = \mathsf{false}$ for $i = 1, 2$. The plain equation system $X_1 = X_2$, $X_2 = X_1$ also has the solutions $(\mathsf{true}, \mathsf{true})$ and $(\mathsf{false}, \mathsf{false})$. However, here we have another solution for the Boolean equation system, $([\![(\nu X_1 = X_2)(\mu X_2 = X_1)]\!]\,\theta)(X_i) = \mathsf{true}$ for $i = 1, 2$. $\lhd$

6.2 Approximation.

The most well known method for solving fixpoint equations over lattices is based on the approximation technique from proposition 2.20. Calculating the least fixpoint $\mu X.f(X)$ of a monotone (and continuous) function $f(X)$ works in the well known manner: the function f is applied first to the bottom element of the lattice, then to the result of the previous application etc., and the increasing chain of these applications of f will reach the fixpoint after a finite number of steps, if the lattice is finite.

$\perp \subseteq f(\perp) \subseteq f(f(\perp)) \subseteq \ldots \subseteq f^i(\perp) = \mu X.f(X)$ for some $i \in I\!N$

Dually, when starting from the top element $\top$, the greatest fixpoint can be determined.

The method easily extends to nested fixpoints. For nested fixpoints of the same kind such as $\mu X_1.f_1(X_1, \mu X_2.f_2(X_1, X_2))$ both functions can be approximated simultaneously in order to reach the least fixpoint. For $f_1^{i+1}(\perp) \overset{\text{def}}{=} f_1(f_1^i(\perp), f_2^{i+1}(\perp))$ and $f_2^{i+1}(\perp) \overset{\text{def}}{=} f_2(f_1^i(\perp), f_2^i(\perp))$ we get by monotonicity arguments the increasing chain

$\perp \subseteq f_1(\perp) \subseteq f_1^2(\perp) \subseteq \ldots \subseteq f_1^i(\perp) = \mu X_1.f_1(X_1, \mu X_2.f_2(X_1, X_2))$ for some $i \in I\!N$.

For alternating fixpoints such as $\nu X_1.f_1(X_1, \mu X_2.f_2(X_1, X_2))$ a simultaneous calculation is not possible. When approximating $\nu X_1.f_1$ each evaluation of f_1 requires a full approximation of $\mu X_2.f_2$:

$f_1^{i+1}(\top) = f_1(f_1^i(\top), \mu X_2.f_2(f_1^i(\top), X_2))$.

Hence the algorithms based on this technique are exponential in the alternation depth.

The application of the approximation technique to Boolean equation systems is straightforward. From the explanations above follows that all variables of one block can be approximated simultaneously. Therefore the algorithm is most efficient for a Boolean equation system when it is transformed to an equivalent one with a minimal number of blocks. We assume that Boolean equation systems considered here are in such a form where the number of blocks is minimal. (See also definitions 3.33 and 3.34 for notions of nesting depth and alternation depth.) Before discussing the various approximation based algorithms we try to illustrate the approximation scheme for an alternating depth 3 equation system

In picture 6.1 we consider an alternation depth 3 fixpoint equation system $(\mu X_1 = f_1)$ $(\nu X_2 = f_2)$ $(\mu X_3 = f_3)$. The picture simplifies the actual situation in the way, that we draw lattices as lines.

Each fixpoint equation determines one of the planes:

$E_1 \;:\; (X_2, X_3) \rightarrow \mu X_1.f_1(X_1, X_2, X_3)$

$E_2 \;:\; (X_1, X_3) \rightarrow \nu X_2.f_2(X_1, X_2, X_3)$

$E_3 \;:\; (X_1, X_2) \rightarrow \mu X_3.f_3(X_1, X_2, X_3)$

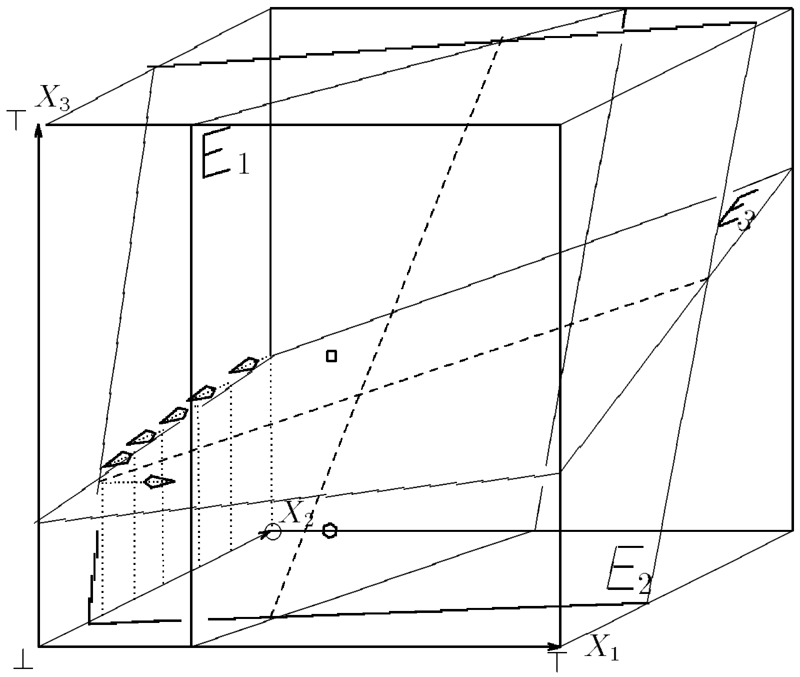


Figure 6.1. Visualizing an alternation depth 3 approximation

The planes E_1, E_2 and E_3 intersect in some of the fixpoints of the equation system. One of them is the solution we are interested in. It will be characterized by the order of equations. In the picture there is just one intersection point, for simplicity.

The approximation algorithm works as follows: it starts at point $X_1 = \perp$, $X_2 = \top$ and $X_3 = \perp$ represented by a dot in the picture. From this point it approximates in the direction of X_3 the E_3-plane. After that, one step is performed in the direction of X_2 corresponding to one evaluation of f_2. The result is a lower value for X_2, closer to the intersection point of planes E_3 and E_2. The next starting point is the lower value of X_2, $X_1 = \perp$ and $X_3 = \perp$. Again the E_3-plane is approximated in direction of X_3, followed by a step in direction of f_2, etc.. These iterative approximations are depicted each by a dotted line with an arrow showing the direction of the approximation. When the intersection line of E_3 and E_2 is reached, one step in direction of X_1 is performed, corresponding to an evaluation of f_3. The result is a new value for X_1 which gives a new starting point for the approximation, illustrated by a hexagon dot in the picture.

Altogether the algorithm moves along the intersection line of E_3 and E_2 until it reaches the first intersection with E_1, the first fixpoint, which is the solution of the system.

In Emerson and Lei's algorithm [EL86] the approximation for unnested fixpoints is performed by the straightforward application of proposition 2.20, the explicit calculation of an increasing chain. The time complexity of the algorithm for Boolean equation systems with one fixpoint operator is then $O(|\mathcal{E}|^2)$. By extension of the approximation technique to Boolean equation systems with arbitrary alternation depth the algorithm has time complexity $O(|\mathcal{E}|^{ad(\mathcal{E})+1)})$. Other authors developed faster algorithms for the approximation of unnested fixpoints, e.g. Arnold and Crubille [AC88], Cleaveland and Steffen [CS91], Andersen [And92, And93] and Vergauwen and Lewi [VL92]. Arnold and Crubille's and Vergauwen and Lewi's algorithms are based on Boolean equation systems, Andersen argues on dependency graphs, Cleaveland and Steffen on μ-calculus equation systems in simple form. However, the basic idea of all these algorithm is the same: in a Boolean equation system with only μ-operators every variable has the solution false unless it is "forced" to have the solution true. It must be true if the right hand side of its equation is the constant true or a disjunction where one variable has the solution true or a conjunction where both variables must be true. The extension of these algorithms to the general case according to the approximation schema then provides algorithms which are exponential in the alternation depth of the system [And92, And93], [CKS92].

A great acceleration was gained by Long & al [LBC$^+$94] for systems with at least alternation depth 3. Their crucial idea is visualized in picture 6.1: the standard approximation technique would continue the approximation of plane E_3 from the new start point, which is marked by a hexagon in the picture. Actually, from the previous approximation in the lower X_1-level and monotonicity of the functions we know that the E_3 plane must lie above the square point, which may be used as the new starting point then. Their algorithm is exponential in half of the alternation depth of the system.

All algorithms mentioned above are global ones. Andersen [And92] derived a local algorithm for alternation free fixpoint expressions based on approximation techniques, but having a slightly higher worst case complexity than the global ones. In [VL94] Vergauwen and Lewi presented a local algorithm for Boolean equation systems of alternation depth 2 which is also approximation based. Their algorithm has the same complexity as comparable global algorithms, but the advantage of local methods that it possibly needs just a small subset of equations to determine the variable of interest. This subset of equations has the property that the solutions of variables of the subset do not depend upon solutions of variables outside. It seems to be the case that the other local methods as e.g. tableaux make use of the same subsets of equations (up to nondeterministic choice).

In the table below complexity results of the algorithms mentioned are collected. Many of them were not intended for Boolean equation system and the complexity measures are for an adapted version. When applied directly to the model checking problem in some cases there are slightly better bounds. The alternation depth $ad(E)$ is abbreviated by ad. For the local model checking algorithm in [VL94] it is $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$.

6.3 Tableaux.

In this section we define a tableau method for solving Boolean equation systems. In contrast to global methods, which solve a Boolean equation system completely. a tableau gives a solution just for one variable. For this purpose not all equations are required. It is therefore called a local method. The tableau method presented here is the one of Stirling and Walker [SW89] applied to Boolean equation systems.

Consider a Boolean equation system $\mathcal{E}$ being in standard form and an environment θ. Assume the solution is $\theta' \overset{\text{def}}{=} [\![\mathcal{E}]\!]\,\theta$. The goal is to

Time Complexity of Approximation Based Algorithms			
algorithm from	fragment	complexity	
[EL86]	full	$O(\|\mathcal{E}\|^{ad+1})$	global
[AC88]	ad 1	$O(\|\mathcal{E}\|\|lhs(\mathcal{E})\|)$	global
[And92]	ad 1	$O(\|\mathcal{E}\|)$	global
[CS91]	ad 1	$O(\|\mathcal{E}\|)$	global
[VL92]	ad 1	$O(\|\mathcal{E}\|)$	global
[And92]	full	$O(\|\mathcal{E}\|^{ad})$	global
[CKS92]	full	$O(\|\mathcal{E}\|^{ad})$	global
[VL92]	full	$O(\|\mathcal{E}\|^{ad})$	global
[LBC+94]	full	$O(ad^2\|\mathcal{E}\|^{\lfloor ad/2\rfloor+1})$	global
[And92]	ad 1	$O(\|\mathcal{E}\|log(\|\mathcal{E}\|))$	local
[VL94]	ad 2	$O(\|\mathcal{E}_1\|+\|lhs(\mathcal{E}_1)\|\|\mathcal{E}_2\|)$	local
[VLAP94]	full	$O(\|\mathcal{E}\|^{c*ad})$	local

show that $([\![\mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$. The solution for X_i can only be true, if for equation $\sigma_i X_i = f_i$ the right-hand side f_i is true at the solution, i.e. $f_i(\theta') = \mathsf{true}$. A subgoal is then trying to show that f_i gets true for the solution. A tableau for variable X_i is a proof tree with root X_i. The sucessors of X_i are variables representing the subgoals. The rules for constructing a tableau are collected below. Rules are applied until a termination condition holds for a node. In the case that there is no rule applicable to a node we have reached a leaf and can decide whether it is successful or not. A tableau is successful if all its leaves are successful.

Termination condition 1: The node n containing X_j is a leaf of the tableau if on the path from n to the root there is another node n' containing X_j, and between n and n' there is no node containing a variable X_i such that X_i is a variable of a lower block than X_j in $\mathcal{E}$. The node n' is called the companion of n.

Termination condition 2: The node n containing X_j is a leaf of the tableau, if on the path from n to the root there is another node n' containing X_j. The node n' is called the companion of n.

Tableau rules:

$$[\wedge_1] \quad \frac{X_i}{X_j \qquad X_k} \qquad \sigma_i X_i = X_j \wedge X_k \text{ is an equation of } \mathcal{E}$$

$$[\wedge_2] \quad \frac{X_i}{X_j} \qquad \sigma_i X_i = X_j \qquad \text{is an equation of } \mathcal{E}$$

$$[\vee_1] \quad \frac{X_i}{X_j} \qquad \sigma_i X_i = X_j \vee X_k \text{ is an equation of } \mathcal{E}$$

$$[\vee_2] \quad \frac{X_i}{X_k} \qquad \sigma_i X_i = X_j \vee X_k \text{ is an equation of } \mathcal{E}$$

A leaf containing the constant true is successful, a leaf containing the constant false is unsuccessful. For leaves containing a variable the success criterion differs for the termination conditions:

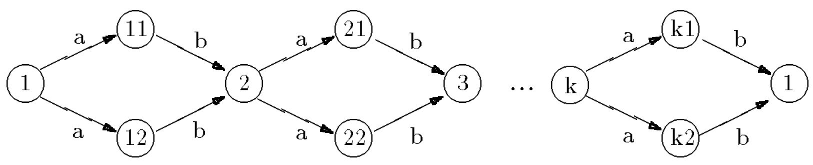
Success criterion 1: A leaf containing a ν-variable is successful, a leaf containing a μ-variable is unsuccessful.

Success criterion 2: A leaf n is successful, if the least (w.r.t. $\trianglelefteq$) variable at a node between n and its companion is a ν-variable. A leaf n is unsuccessful, if the least (w.r.t. $\trianglelefteq$) variable at a node between n and its companion is a μ-variable.

Proposition 6.1 $([\![\mathcal{E}]\!]\,\theta)(X_1) =$ true iff there exists a successful tableau with root X_1.

Example This is a demonstration of the exponential growth of a tableau (for both termination conditions) when the underlying transition system just grows linearly.

Consider the μ-calculus formula $\nu X.[a]\mu Y.\langle b\rangle(Y \vee X)$ and the following transition system:



The Boolean equation system derived is

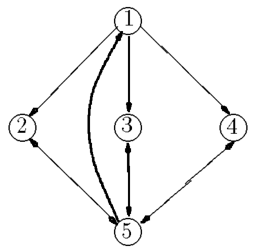
$$\begin{array}{rcl}
\nu X_1 & = & Y_{11} \wedge Y_{12} \\
\nu X_{11} & = & \mathsf{true} \\
 & \dots & \\
\nu X_k & = & Y_{k1} \wedge Y_{k2} \\
\nu X_{k1} & = & \mathsf{true} \\
\mu Y_1 & = & \mathsf{false} \\
\mu Y_{11} & = & Y_2 \vee X_2 \\
 & \dots & \\
\mu Y_k & = & \mathsf{false} \\
\mu Y_{k2} & = & Y_1 \vee X_1
\end{array}$$

The tableau for the case $k = 3$ is:

$$\dfrac{X_1}{\dfrac{Y_{11}}{\dfrac{X_2}{\dfrac{Y_{21}}{\dfrac{X_3}{\dfrac{Y_{31}}{X_1}\quad\dfrac{Y_{32}}{X_1}}}\quad\dfrac{Y_{22}}{\dfrac{X_3}{\dfrac{Y_{31}}{X_1}\quad\dfrac{Y_{32}}{X_1}}}}}\qquad\dfrac{Y_{12}}{\dfrac{X_2}{\dfrac{Y_{21}}{\dfrac{X_3}{\dfrac{Y_{31}}{X_1}\quad\dfrac{Y_{32}}{X_1}}}\quad\dfrac{Y_{22}}{\dfrac{X_3}{\dfrac{Y_{31}}{X_1}\quad\dfrac{Y_{32}}{X_1}}}}}}$$

It is obvious that the exponential size of the tableau is due to the fact that it contains the same subtrees several times. Another example where the subtrees are not exactly the same, but similar is the following:

Example:
Given the μ-calculus formula
$\nu X.\langle - \rangle \mu Y.\langle - \rangle\langle - \rangle X \wedge \langle - \rangle\langle - \rangle Y$
and the transition system



The Boolean equation system derived is

$$\begin{array}{lcl}
\nu X_1 & = & Y_2 \vee Y_3 \vee Y_4 \vee Y_5 \\
\nu X_2 & = & Y_5 \\
\nu X_3 & = & Y_5 \\
\nu X_4 & = & Y_5 \\
\nu X_5 & = & Y_1 \vee Y_2 \vee Y_3 \vee Y_4 \\
\mu Y_1 & = & X_5 \wedge Y_5 \\
\mu Y_2 & = & \bigvee_{i=1}^{5} X_i \wedge \bigvee_{i=1}^{5} Y_i \\
\mu Y_3 & = & \bigvee_{i=1}^{5} X_i \wedge \bigvee_{i=1}^{5} Y_i \\
\mu Y_4 & = & \bigvee_{i=1}^{5} X_i \wedge \bigvee_{i=1}^{5} Y_i \\
\mu Y_5 & = & \bigvee_{i=1}^{5} X_i \wedge \bigvee_{i=1}^{5} Y_i
\end{array}$$

The tableau for this Boolean equation system has an enormous size. An implementation of the original tableau method of Stirling & Walker was stopped after having created 22 million nodes. The sceptical reader may try it by hand.

The version of the tableau method of Cleaveland as implemented in the concurrency workbench can deal with redundancy of this kind. The examples presented here can also be solved without producing redundant information by the technique of [Mad92].

A tableau based model checking algorithm was introduced by Larsen [Lar95] for unnested fixpoint expressions. Stirling and Walker [SW89] and Cleaveland [Cle90] developed tableau methods for the full modal μ-calculus. Winskel [Win89] extracted the principles of these tableau methods and presented them as a rewrite system. Unfortunately these methods suffer from a high worst case complexity, which was demonstrated by examples in this section. One reason for that is that in different subtrees of the tableau the same (or very similar) subgoals may be computed repeatedly. For unnested fixpoint expressions Larsen [Lar92] presented a tableau method with polynomial worst case. There previously discovered (failed) results are remembered. In [Mad92] the tableau methods of [SW89] and [Cle90] are extended by additional structure which allows to make maximal use of results gained in one subtableau for later subtableaux during construction. However, some amount of redundancy is inherent to top-down constructions, and it can only be avoided by a bottom-up evaluation. This approach leads to the Gauß elimination method in section 6.4.

6.4 Gauß elimination.

The method for solving Boolean equation systems presented in this section is similar to the Gauß elimination algorithm for linear equation systems. It is the only method known so far which does not require backtracking techniques: an equation system is stepwise reduced by one variable and equation after the other until the solution is determined. The reduction consists of two steps which are applied iteratively. First comes an **elimination step**, where for a variable X an expression is constructed containing no occurrence of X. In a subsequent **substitution step** each occurrence of X in the rest of the equation system is substituted by the X-free expression. The remaining system contains no occurrence of X on the right-hand sides of its equations. Thus the problem of solving a Boolean equation system is reduced to the problem of solving a smaller Boolean equation system.
The Gauß elimination algorithm is also related to the tableau methods. The main idea here is that the construction of a tableau in a top-down manner leads to trees possibly containing many copies of identical (or similar) subtrees. A very natural way to overcome such an unnecessary blow-up is to construct a directed acyclic graph instead of a tree (i.e. a tableau). This can be done in a bottom-up manner.
A pure bottom-up method would again lead to a global algorithm involving all equations of the Boolean equation system. The combination of a tableau-like top-down selection of equations and bottom-up

evaluation gives an algorithm which makes use of the same information as a tableau, but avoids redundancy. In many examples, where the approximation method or tableau method have an exponential behavior, Gauß elimination solves the problem in linear time. However, for the naive algorithm derived from Gauß elimination there exists an example where the expressions created have exponential size.
The algorithm was introduced in [BM93, Mad95] and in a slightly different version by Kalorkoti [Kal96].

6.4.1 Global and local algorithm.

In the case of Gauß elimination for Boolean equation systems an elimination step infered in lemma 6.2 is a consequence of lemma 3.29. In an equation $\mu X = f$ each occurrence of X in f may be substituted by false, or dually for ν by true.
The substitution step derived from lemma 6.3 preserves the solution just in the case when we follow the order: an occurrence of a variable may be substituted by a right-hand side expression only in all lower (w.r.t. $\trianglelefteq$) equations. (See also proposition 2.21.)
The proofs presented here were partly suggested by Vergauwen [Ver95]. Different proofs can be found in [BM93, Mad95].
The elimination step is based on the following lemma.

Lemma 6.2 Let

- $\mathcal{E}_1, \mathcal{E}_2$ be Boolean equation systems,
- $\sigma X{=}f,\ \sigma X{=}f'$ Boolean equations,
  where $f' = f[X/b_\sigma]$.

Then $[\![\mathcal{E}_1\ (\sigma X{=}f)\ \mathcal{E}_2]\!]\,\theta = [\![\mathcal{E}_1\ (\sigma X{=}f')\ \mathcal{E}_2]\!]\,\theta$.

Proof: According to proposition 3.14 it is sufficient to show that
$[\![(\sigma X{=}f)\ \mathcal{E}_2]\!]\,\theta \quad = \quad [\![(\sigma X{=}f')\ \mathcal{E}_2]\!]\,\theta.$

$$\begin{aligned}
[\![(\sigma X{=}f)\ \mathcal{E}_2]\!]\,\theta &= [\![\mathcal{E}_2]\!]\,\theta[X/f(\ [\![\mathcal{E}_2]\!]\,\theta[X/b_\sigma]\ )] \\
&= [\![\mathcal{E}_2]\!]\,\theta[X/f'(\ [\![\mathcal{E}_2]\!]\,\theta[X/b_\sigma]\ )] \\
&= [\![(\sigma X{=}f')\ \mathcal{E}_2]\!]\,\theta
\end{aligned}$$
□

The following lemma is the basis for the substitution step:

Lemma 6.3 Let

- $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$ be Boolean equation systems,
- $\sigma_1 X_1{=}f, \sigma_1 X_1{=}f', \sigma_2 X_2{=}g$ Boolean equations, where $f' = f[X_2/g]$.
- $[\![\mathcal{E}_1\ (\sigma_1 X_1{=}f)\ \mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3]\!]\,\theta \stackrel{\text{def}}{=} \theta_1$
- $[\![\mathcal{E}_1\ (\sigma_1 X_1{=}f')\ \mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3]\!]\,\theta \stackrel{\text{def}}{=} \theta_2$.

Then $\theta_1 = \theta_2$

Proof: Again following proposition 3.14 we just need to show that for
$\theta'_1 \stackrel{\text{def}}{=} [\![(\sigma_1 X_1{=}f)\ \mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3]\!]\,\theta$ and
$\theta'_2 \stackrel{\text{def}}{=} [\![\ (\sigma_1 X_1{=}f')\ \mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3]\!]\,\theta$
it is the case that $\theta'_1 = \theta'_2$.
We will show that θ'_1 fulfills both conditions of proposition 3.5 for the solution of $(\sigma_1 X_1{=}f')\ \mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3$. Hence θ'_2 is lexicographically smaller than θ'_1, because θ'_2 is the solution.
Show $f'(\theta'_1) = \theta'_1(X_1)$ (condition (1) of proposition 3.5)

$$\begin{aligned}
g(\theta'_1) &= \theta'_1(X_2) \\
\theta'_1(X_1) &= f(\theta'_1) \\
&= f(\theta'_1[X_2/\theta'_1(X_2)]) \\
&= f(\theta'_1[X_2/g(\theta'_1)]) \\
&= f'(\theta'_1)
\end{aligned}$$

Show $[\![\mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3]\!]\,\theta'_1 = \theta'_1$ (condition (2) of proposition 3.5):
follows from proposition 3.7
Analogously, the dual holds: θ'_2 fulfills both conditions of proposition 3.5 for the solution of $(\sigma_1 X_1{=}f)\ \mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3$, and hence θ'_1 is lexicographically smaller than θ'_2.
$f(\theta'_2) = \theta'_2(X_1)$ (condition (1) of proposition 3.5):
analogously
$[\![\mathcal{E}_2\ (\sigma_2 X_2{=}g)\ \mathcal{E}_3]\!]\,\theta'_2 = \theta'_2$ (condition (2) of proposition 3.5)
Altogether we can conclude that $\theta'_1 = \theta'_2$. □

Based on these both lemmata is the following algorithm in pseudo code.

```
Input are $(\sigma_1 X_1 = f_1) \dots (\sigma_n X_n = f_n)$ and $\theta$

i := n;
while not $(f_1 \equiv \mathsf{true}$ or $f_1 \equiv \mathsf{false})$
    do
        Instantiate $X_i$ in $f_i$ to $b_{\sigma_i}$;                    (elimination step)
        Substitute $f_i$ for $X_i$ in $f_1, \dots, f_{i-1}$;        (substitution step)
        $f_1$:=Eval($f_1$); ... ; $f_{i-1}$:=Eval($f_{i-1}$);       (evaluation step)
        i := i - 1;
    od
```

Figure 6.2. Global Version of the Gauß Elimination Algorithm

A crucial point in the algorithm are the evaluation rules for Boolean expressions applied in the function Eval of the algorithm in figure 6.4.1. In an implementation binary decision diagrams were chosen as data structure for Boolean expressions. There the evaluation rules are performed implicitly with every substitution and elimination step. In the examples done by hand the following set of Boolean laws was used for evaluation.

$$\begin{aligned}
X \wedge \mathsf{true} &= X \\
X \vee \mathsf{true} &= \mathsf{true} \\
X \wedge \mathsf{false} &= \mathsf{false} \\
X \vee \mathsf{false} &= X \\
X \vee (X \wedge Y) &= X \\
X \wedge (X \vee Y) &= X \\
(X \wedge Y) \vee (X \wedge Z) &= X \wedge (Y \vee Z) \\
(X \vee Y) \wedge (X \vee Z) &= X \vee (Y \wedge Z)
\end{aligned}$$

In most contexts we are only interested in the first component of the solution, i.e. whether X_1 is true or false. Therefore the algorithm in figure 6.4.1 stops, if the solution of X_1 (f_1) is determined. If we are interested in the whole solution the Gauß elimination step and substitution step have to be applied n times giving an expression for every X_i where the variables $X_i, \ldots, X_n$ do not occur. A straight backward substitution leads to the whole solution.

Example: Starting with the Boolean equation system:

$$\begin{aligned} \mu X_1 &= X_2 \vee X_3 \\ \nu X_2 &= X_3 \wedge X_4 \\ \mu X_3 &= X_4 \vee X_1 \\ \nu X_4 &= X_1 \wedge X_2 \end{aligned}$$

Substitution of $X_1 \wedge X_2$ for X_4 and evaluation (The substituted expressions are underlined):

$$\begin{aligned} \mu X_1 &= X_2 \vee X_3 \\ \nu X_2 &= X_3 \wedge \underline{(X_1 \wedge X_2)} \\ \mu X_3 &= \underline{(X_1 \wedge X_2)} \vee X_1 = X_1 \end{aligned}$$

Substitution of X_1 for X_3 and evaluation:

$$\begin{aligned} \mu X_1 &= X_2 \vee \underline{X_1} \\ \nu X_2 &= \underline{X_1} \wedge (X_1 \wedge X_2) = X_1 \wedge X_2 \end{aligned}$$

Elimination of X_2 in $\nu X_2 = X_1 \wedge X_2$ gives $\nu X_2 = X_1 \wedge \mathsf{true} = X_1$. Substitution of X_1 for X_2:

$$\mu X_1 = \underline{X_1} \vee X_1 = X_1 = \mathsf{false} \text{ (by an elimination step)}$$

The complete system constructed by the algorithm is:

$$\begin{aligned} \mu X_1 &= \mathsf{false} && \text{(from 4)} \\ \nu X_2 &= X_1 && \text{(from 3)} \\ \mu X_3 &= X_1 && \text{(from 2)} \\ \nu X_4 &= X_1 \wedge X_2 && \text{(from 1)} \end{aligned}$$

Backward substitution gives $X_1 = X_2 = X_3 = X_4 = \mathsf{false}$. $\lhd$

If only the first variable is of interest, it suffices to consider only the subset of equations which is necessary to determine the solution for

X_1. The relevant subset of equations is selected in a top-down manner. This observation leads to the local version of Gauß elimination given in figure 6.3. The idea is as follows. We start with the equation system $\mathcal{E}'$ consisting only of the equation $(\sigma_1 X_1 = f_1)$. As long as X_1 is not evaluated to true or false we select a free variable from f_1, insert its equation in $\mathcal{E}'$, apply the global version of Gauß elimination, and continue in the same way with the modified equation system $\mathcal{E}'$.

```
$\mathcal{E}' := (\sigma_1 X_1 = f_1)$;
Instantiate $X_1$ in $f_1$;                                  (elimination step)
$f_1 := \mathrm{Eval}(f_1)$;                                 (evaluation step)
**while not** $(f_1 = \mathsf{true}$ **or** $f_1 = \mathsf{false})$
  **do**
    Select $X_j$ from $f_1$, where $X_j$ is not in $lhs(\mathcal{E}')$;
    Create $f_j$, insert $\sigma_j X_j = f_j$ in $\mathcal{E}'$
      according to the order by the transformation rules;
    Apply the global version of Gauß elimination to $\mathcal{E}'$
  **od**
```

Figure 6.3. Local Version of the Gauß Elimination Algorithm

There exists an acceleration of the algorithm which works as follows: an occurrence of a variable X_j may be substituted by true or false at an earlier stage than when occurring on the right hand side of its defining equation $\sigma_j X_j = f_j$. This is the case, when it does **not** happen that (a copy of) this occurrence of X_j is substituted into an equation $\sigma_i X_i = f_i$ where $X_i \trianglelefteq X_j$ during the algorithm. This property is a static one in the sense that it can be determined in advance, whether such a substitution into a prior equation will happen. A special case of this possibility appears in the definition of the semantics for Boolean equation systems (proposition 3.30): any occurrence of the first variable, X_1, will never be substituted into a prior equation, simply because there does not exist a prior one. Hence, every occurrence of X_1 may be substituted by true or false right in the beginning. However, for this acceleration it is the case that backward substitution does not work: it is only guaranteed that the algorithm produces the correct solution for the first variable.

6.4.2 Complexity for the general case.

In this section we argue that the naive algorithm derived from Gauß elimination is of complexity exponential in the number of equations, and give an example for it. The source of complexity here is the size of right-hand side expressions, which in an example growths exponentially. However, it is not known, whether there exists a version of the algorithm, where this exponential blow up is avoided by more intelligent storage of expressions.

In comparison to the approximation algorithm the behaviour cwof Gauß elimination algorithms is very different. We show that the complexity of Gauß elimination is independent from the alternation depth of the Boolean equation system, i.e. given an arbitrary Boolean equation system there exists a Boolean equation system with the same number of equations, but alternation depth 1, and for both systems the algorithm needs same time and space. An example demonstrates a case, where the approximation based algorithms needs exponentially many steps, but Gauß elimination only polynomial time and space. For some fragments we show that Gauß elimination has complexity polynomial in the number of equations. Especially for the fragment corresponding to L_2 Gauß elimination provides an $O(n^2)$ algorithm.

The number of substitution steps during the Gauß elimination in the global algorithm is less than $(n-1)+(n-2)+\ldots+1 \leq n^2$. The local version includes at most n applications of the global algorithm giving alltogether less than n^3 substitution steps.

The crucial point concerning complexity is the size of the Boolean expressions arising from iterative substitutions. In general substitution of Boolean expressions into Boolean expressions leads to size exponential in the number of variables involved. Assuming that a Boolean equation system in normal form consists of n equations (and different

variables), then the size of the Boolean expressions created during the algorithm is bound by 2^n. Hence the worst case complexity of the global and local algorithm is $O(2^n)$.
Trying a big number of examples showed that the application of evaluation rules as discussed above and the elimination rule keep the expressions created relatively small. Finding an example where the right-hand side expressions are of exponential size turned out to be a difficult task. The example below was constructed with help of Brinksma [Bri96] and Rossmanith [Ros96]. The basic idea is to find an expression where one variable appears twice and the laws for evaluation of Boolean expressions as fixed for the algorithm are not applicable in order to reduce it. Such an expression gives a scheme for iterative substitution with no possibility of reduction. The fixpoint operators in this example are irrelevant, because when building up expressions for X_n up to $X_{n/2}$ there is no application of the elimination rule possible. Therefore fixpoint operators are left away. Assume $n \in 10\mathbb{N}$. The size of expressions is then bound by $O(2^{n/5})$.

$$\begin{array}{lcl}
X_1 & = & X_2 \\
X_2 & = & X_3 \\
X_3 & = & X_4 \\
X_4 & = & X_5 \\
 & \ldots & \\
X_{n/2} & = & X_{n/2+1} \\
X_{n/2+1} & = & X_{n/2+2} \vee X_{n/2+3} \\
X_{n/2+2} & = & X_{n/2+4} \wedge X_{n/2-1} \\
X_{n/2+3} & = & X_{n/2+5} \wedge X_{n/2-2} \\
X_{n/2+4} & = & X_{n/2+6} \vee X_{n/2-3} \\
X_{n/2+5} & = & X_{n/2+6} \vee X_{n/2-4} \\
 & \ldots & \\
X_{n-14} & = & X_{n-13} \vee X_{n-12} \\
X_{n-13} & = & X_{n-11} \wedge X_{13} \\
X_{n-12} & = & X_{n-10} \wedge X_{12} \\
X_{n-11} & = & X_{n-9} \vee X_{11} \\
X_{n-10} & = & X_{n-9} \vee X_{10}
\end{array}$$

$$\begin{array}{lcll}
X_{n-9} & = & X_{n-8} & \vee \; X_{n-7} \\
X_{n-8} & = & X_{n-6} & \wedge \; X_9 \\
X_{n-7} & = & X_{n-5} & \wedge \; X_8 \\
X_{n-6} & = & X_{n-4} & \vee \; X_7 \\
X_{n-5} & = & X_{n-4} & \vee \; X_6 \\
X_{n-4} & = & X_{n-3} & \vee \; X_{n-2} \\
X_{n-3} & = & X_{n-1} & \wedge \; X_5 \\
X_{n-2} & = & X_n & \wedge \; X_4 \\
X_{n-1} & = & X_1 & \vee \; X_3 \\
X_n & = & X_1 & \vee \; X_2
\end{array}$$

In order to make the conceptual difference to the approximation method clear we show that Gauß elimination is independent of the alternation depth of a Boolean equation system. Least and greatest fixpoints are treated in a similar way: the corresponding variables are substituted by a constant, true for a variable with greatest fixpoint, false for a variable with least fixpoint.

Proposition 6.4 The complexity of the naive algorithm based on Gauß elimination is independent of the alternation depth of the Boolean equation system and hence also of the underlying μ-calculus formula.

Proof: The idea is that for a given Boolean equation system $\mathcal{E}$ of arbitrary alternation depth we construct a Boolean equation system $\mathcal{E}'$ with only μ-fixpoints, and $\mathcal{E}'$ has the property that the size of expressions created during Gauß elimination is at least the size of expressions created for $\mathcal{E}$. (Their solutions may differ.)
For this purpose we have to restrict the class of Boolean equation systems we consider to those which do not contain constants and all right hand side variables are bound. In fact this is not a real restriction, because constants and fixed right hand side variables can be eliminated from a Boolean equation system in linear time (in the size of the system) such that the solution of the system is preserved. Hence applying this elimination before starting any algorithm will not increase

its complexity. Furthermore we consider Boolean equation systems in standard form. This representation can be achieved by a linear blow-up of the original system (in the size of the underlying μ-calculus formula).

The transformation from $\mathcal{E}$ to $\mathcal{E}'$ works as follows:

- every conjunction containing a ν-variable is transformed to a disjunction (of the same variables), and
- every ν is substituted by a μ.

Note that the solution of $\mathcal{E}'$ will be θ', where $\theta'(X) = \mathsf{false}$ for all $X \in lhs(\mathcal{E}')$.

We have to show that the size of expressions when applying Gauß elimination to $\mathcal{E}'$ is greater or equal to those for $\mathcal{E}$. Both systems have the same dependency graph, and therefore also the same structure of substitutions. We just have to make sure that in $\mathcal{E}'$ "no variables get lost" in comparison to $\mathcal{E}$.

The property to show holds for the initial systems $\mathcal{E}$ and $\mathcal{E}'$. Furthermore corresponding equations of both systems contain the same variables. Let $\sigma_i X_i{=}f, \sigma_j X_j{=}g$ be equations of $\mathcal{E}$ and $\mu X_i{=}f'$, $\mu X_j{=}g'$ the corresponding equations of $\mathcal{E}'$, where $i < j$.

Applying a substitution step leads to an expression $\sigma_i X_i{=}f[X_j/g]$ and $\mu X_i{=}f'[X_j/g']$ respectively. If f and g contained at least the same (number of) variables as f' and g' then this will also hold for $f[X_j/g]$ and $f'[X_j/g']$ and the size of $f'[X_j/g']$ is greater or equal to the size of $f[X_j/g]$.

For an elimination step consider as part of an expression of $\mathcal{E}$ a conjunction $X_i \wedge X_j$, where X_i is a ν-variable and X_j is a μ-variable, and X_i is substituted by true. Then the conjunction evaluates to X_j. In the transformed system $\mathcal{E}'$ the conjunction was transformed to a disjunction $X_i \vee X_j$ and X_i will be substituted by false. The disjunction evaluates to X_j as in the other case. When X_j is substituted by false then the conjunction of $\mathcal{E}$ will evaluate to false, whereas the disjunction of $\mathcal{E}'$ will evaluate to X_i, leading to a greater expression (with at least one more variable) than in $\mathcal{E}$. Note that the case of substituting true for a variable in a disjunction introduced in $\mathcal{E}'$ does not happen, because the solution of $\mathcal{E}'$ gives false for every variable of $\mathcal{E}'$.

Any statement about size of Boolean expressions makes only sense if we choose a sensible representation of Boolean expressions. In the case here we evaluate expressions just with the rules for constants. □

We now want to demonstrate by some examples "good" behaviour of Gauß elimination, where tableau method and approximation method need exponential space and/or time. Two examples have already been treated in section 6.3, illustrating the exponential blow-up of the plain tableau method. These examples can easily be solved with the techniques from this section without any blow-up. This might not be too surprising as already extensions of the tableau method in [Cle90] and [Mad92] can deal with these examples.

Here we present another example. Its features are the following:

- It is scalable, i.e. it is a set of examples, which can have arbitrary size n and alternation depth n.
- The Gauß elimination method produces only expressions of a fixed constant length for any of the examples, and the complexity is $O(n^2)$.
- Known algorithms based on the approximation technique are exponentially in n.

The last aspect is due to the fact that the example is constructed in a way that a maximal number of backtracking steps is required.

Let $n \in 2\mathbb{N}$

$$\begin{array}{rcl}
\nu X_1 & = & X_2 \wedge X_n \\
\mu X_2 & = & X_1 \vee X_n \\
\nu X_3 & = & X_2 \wedge X_n \\
\mu X_4 & = & X_3 \vee X_n \\
 & \ldots & \\
\nu X_{n-3} & = & X_{n-4} \wedge X_n \\
\mu X_{n-2} & = & X_{n-3} \vee X_n \\
\nu X_{n-1} & = & X_{n-2} \wedge X_n \\
\mu X_n & = & X_{n-1} \vee X_{n/2}
\end{array}$$

6.4.3 Complexity for subclasses.

In this section we consider classes of Boolean equation systems for which Gauß elimination has complexity $O(n^2)$. These are the disjunctive and the conjunctive class and a combination of them. The fragments of the modal μ-calculus that gives rise to these classes are L_1 and L_2. In [EJS93] the fragment L_2 was shown being equi-expressive to $ECTL*$ [VW83], an extension of $CTL*$.
A Boolean equation is called **disjunctive**, if its right-hand side is a disjunction or it is a 2-ary conjunction where at least one conjunct is a constant. A Boolean equation system in standard form is disjunctive, if all its equations are. Expressions created during an application of Gauß elimination to disjunctive systems are always disjunctions. The size of an disjunction is bound by the number of different variables that are involved, which is at most the number of equations in the system.

Proposition 6.5 A disjunctive Boolean equation system of size n can be solved in time and space $O(n^2)$ with the global version of the Gauß elimination algorithm. Applying the local version of the Gauß elimination algorithm needs time $O(n^3)$ and space $O(n^2)$.

Proof: The global version of the Gauß elimination algorithm takes at most n^2 elimination and substitution steps. Each right-hand side of an equation can be represented as a set. Substitution corresponds then to a removing one element of a set and union of two sets. These operations can be performed in constant time. There exist always not more than n different expressions, or sets resp., each of size less than n. The local algorithm needs less than n^3 elimination and substitution steps. □
The conjunctive class is defined analogously: a Boolean equation system in standard form is **conjunctive**, if it contains only equations with conjunctions on their right hand sides, or disjunctions, where one of the disjuncts is a constant. The dual argument holds here.

Proposition 6.6 A conjunctive Boolean equation system of size n can be solved in time and space $O(n^2)$ with the the local version of the Gauß elimination algorithm. Applying the local version of the Gauß elimination algorithm needs time $O(n^3)$ and space $O(n^2)$.

Proof: Analogously to the previous proof of proposition 6.5 $\Box$.
Disjunctive and conjunctive classes may be combined in a restricted way. Intuitively, the requirement is, that when applying the Gauß elimination algorithm never a disjunction (containing more than a constant or a single variable) is substituted into a conjunction or vice versa. The formal definition of the **combined class** is given below. Recall that a subsystem $\mathcal{E}'$ of $\mathcal{E}$ is closed with respect to $\mathcal{E}$, iff $free(\mathcal{E}') \subseteq free(\mathcal{E})$.

- each disjunctive system is contained in the combined class;
- each conjunctive system is contained in the combined class;
- if a Boolean equation system $\mathcal{E}$ of the combined class containes a disjunctive equation $\sigma_d X_d = f_d$ and a conjunctive equation $\sigma_c X_c = f_c$, then there is a variable X in either f_d or f_c, such that
  - $\sigma X = f_X$ is the least (w.r.t. $\trianglelefteq$) equation of a subsystem $\mathcal{E}'$ closed with respect to $\mathcal{E}$,
  - $\mathcal{E}'$ is contained in the combined class,
  - $(\sigma_d X_d = f_d) \lhd (\sigma X = f_X)$,
  - $(\sigma_c X_c = f_c) \lhd (\sigma X = f_X)$.

Proposition 6.7 For a Boolean equation system in the combined class the global version of the Gauß elimination algorithm solves the system in space and time $O(n^2)$.

Proof: The observation here is that the Gauß elimination algorithm evaluates the least variable of a closed subsystem to a constant. The rest is analogous to the disjunctive and conjunctice case. $\Box$
Note, that $\sigma X = f_X$ has not to be necessarily the least equation of the subsystem; it may be one equation of a closed subsystem. In this case the Gauß elimination algorithm has to be modified in the way, that after each evaluation step equations with a constant right-hand side are eliminated from the equation system according to lemma 3.20, followed by a further evaluation step, and so on. In this case each variable of a closed subsystem is evaluated to a constant.

Also note, that the local version of the Gauß elimination algorithm applied to Boolean equation systems of the combined class possibly substitutes disjunctions into conjunctions and vice versa. The reason is that a subsystem of a Boolean equation system in the combined class is not necessarily contained in the combined class. In order to get a local algorithm for the combined class there is a modification necessary: before application of the global algorithm equations have to be created until the actual subsystem is in the combined class.
It is easy to see that Boolean equation systems derived from μ-calculus formulae of the fragment L_1 are disjunctive, and Boolean equation systems derived from μ-calculus formulae of fragment L_2 are contained in the combined class. (See definitions for L_1 and L_2 in chapter 4). Emerson, Jutla and Sistla [EJS93] presented a model checking algorithm for L_1 and L_2 which is of complexity $O(|\Phi|^2|T|)$. Transformation to Boolean equation systems gives also an $O(|\mathcal{E}|^2)$ algorithm.
Bhat and Cleaveland [BC96] developed a model checking algorithm for the fragment L_1. It operates on the dependency graph where nodes are additionally labelled by $\vee$ or $\wedge$. A formula of linear time temporal logic expresses that there exists a ν-cycle (or constant `true`) reachable from the node representing the μ-calculus formula and initial state, which implies that the formula satifies the transition system. The linear time formula is proved by a tableau system. The time complexity of their algorithm is $O(ad(\Phi)*|\Phi|*|T|)$, giving an $O(ad(\mathcal{E})|\mathcal{E}|)$ algorithm for the case of Boolean equation systems. For the extension of the algorithm to the fragment L_2 they claim, that the resulting algorithm may be shown also to have time complexity $O(ad(\Phi) * |\Phi| * |T|)$.

6.5 Complexity.

We give a proof that the problem of solving Boolean equation system is contained in NP $\cap$ Co-NP.
For the model checking problem this is a known result. Most of the proofs (e.g. [EJS93, BVW94]) reduce the model checking problem to non-emptiness problems of tree automata, which are in NP. Then the

model checking problem is also contained in in Co-NP, just because a property holds for some model if its negation does not and vice versa. We claim that the proof in the framework of Boolean equation systems is quite simple. Roughly the argumentation works as follows.

An arbitrary Boolean equation system in standard form can be reduced to a disjunctive system by choosing one variable out of every conjunction and throwing the other one away. In general the reduced Boolean equation system has a solution pointwise greater than the solution of the original one. However, in proposition 3.36 it was shown that there must be one reduction to a disjunctive system having the same solution. According to proposition 6.5 we can solve the reduced system in time $O(|\mathcal{E}|^2)$.

Dually a Boolean equation system in standard form can be reduced to a conjunctive system. In general it will have a pointwise lower solution than the original one, but there must exist one reduction giving the same solution. Again a disjunctive Boolean equation system can be solved in quadratic time according to proposition 6.6.

Given a Boolean equation system in standard form we can guess two reductions (out of exponentially many), one to a disjunctive system, the other one to a conjunctive one. Both can be solved in quadratic time. We know that the solution of the original system lies between the solutions of the conjunctive and the disjunctive one. Hence, if we guessed "correctly" and both systems have the same solution, this must also be the solution of the original system.

Theorem 6.8 Solving a Boolean equation system $\mathcal{E}$ is contained in NP $\cap$ Co-NP.

Proof: We guess a conjunctive system $\mathcal{E}'$: in each equation of $\mathcal{E}$ with a disjunction on the right hand side we reduce the right hand side to one of the disjuncts. The equations with a conjunction on the right hand side remain unchanged. By construction and definition 3.15 follows that $\mathcal{E}' \leq \mathcal{E}$. There are exponentially many possibilities to choose such a conjunctive system. Analogously we guess a disjunctive system $\mathcal{E}'' \geq \mathcal{E}$. Again there are exponentially many possibilities to

guess. In general the solution of $\mathcal{E}'$ is pointwise lower or equal to the solution of $\mathcal{E}$ (proposition 3.16). Proposition 3.36 says that there exists a conjunctive system $\mathcal{E}'$ having the same solution as $\mathcal{E}$. The solution of $\mathcal{E}''$ is pointwise greater or equal to the solution of $\mathcal{E}$ (proposition 3.16). And again there exists a disjunctive system $\mathcal{E}''$ having the same solution as $\mathcal{E}$.
$\mathcal{E}'$ and $\mathcal{E}''$ can be solved in quadratic time (propositions 6.5, 6.6). If they have the same solution then it must be the solution of $\mathcal{E}$.
In lemma 3.35 it was proved that the solving $\overline{\mathcal{E}}$ is the complementary problem to solving $\mathcal{E}$, i.e. $([\![\mathcal{E}]\!]\,\theta)(X) = \mathsf{false}$ iff $([\![\overline{\mathcal{E}}]\!]\,\overline{\theta})(X) = \mathsf{true}$. From the argumentation above follows that solving $\overline{\mathcal{E}}$ is also contained in NP and hence solving $\mathcal{E}$ is also in Co-NP. $\square$

Chapter 7

Peterson's mutex algorithm.

In this section we demonstrate two things: A non-trivial application of the modal μ-calculus and results from verification with a prototype implementation of the local Gauß elimination algorithm. For this purpose the algorithms for mutual exclusion (mutex) seem to be appropriate: on one hand they are more interesting than the coffee machine, but they are small enough to capture concepts easily, on the other hand the properties to be proved result in rather sophisticated μ-calculus formulae.

Roughly the mutex problem is the following: two (or more) processes share a common source which may be used by one process only at one time. When a process has access to the common source then we say it is in the critical section. The task of mutex algorithms is now to organize the availability of the common source in such a way that it never happens that both processes have access at the same time (safety property) and that a requesting process cannot be denied access forever (liveness property).

The basis for the examples presented here is the work of Walker [Wal91], who encoded the best known mutex algorithms as CCS processes and tried to prove safety and liveness properties for them. Whereas for safety properties he was successful, there remained open questions concerning liveness. One reason is that he did not treat fairness in his properties. As Kindler and Walter [Wal95a, KW97] and Vogler [Vog96] pointed out, liveness for mutex algorithms cannot be guaranteed without fairness assumptions. A common possibility is to require fairness for everything. In general, this is not necessary for most cases, and our interest here is to find out what are the precise fairness assumptions for mutex algorithms to fulfill the liveness property. The examples presented here are contained in [KM].

7.1 Modelling the algorithm.

We investigate Peterson's mutex algorithm. Other mutex algorithms can be treated analogously.
Peterson's algorithm works for two processes P_1 and P_2, each one having a Boolean variable, b_1 or b_2 resp., which is set to **true** if a process wishes to enter the critical section. There is a turn variable k taking values from $\{1, 2\}$ and in case of a conflict it gives a priority to the process with the corresponding index. Process P_1 writes to b_1 and reads b_2. Dually process P_2 writes to b_2 and reads b_1. Both processes read and write to variable k. Let $i, j \in \{1, 2\}$ and $j \neq i$.

```
while true do
begin
        (noncritical section);
        $b_i$ := true;
        $k := j$;
        wait until not $b_j$ or $k = i$;
        (critical section);
        $b_i$ := false
end;
```

The processes are modelled following Walker's [Wal91] approach. He formulated the two processes as CCS agents [Mil89]. Each variable is represented by its own agent and writing to a variable or reading it are actions where a process agent and a variable agent synchronize.

Modelling Process P_1:

$$\begin{array}{lcl}
P_1 & = & req_1 \,.\, b_1wt \,.\, kw2 \,.\, P_{11} + \tau \,.\, P_1 \\
P_{11} & = & b_2rf \,.\, P_{12} \;+\; b_2rt \,.\, (kr2 \,.P_{11} \;+\; kr1 \,.\, P_{12}) \\
P_{12} & = & enter_1 \,.\, exit_1 \,.\, b_1wf \,.\, P_1
\end{array}$$

Modelling Process P_2:

$$\begin{array}{lcl}
P_2 & = & req_2 \,.\, b_2wt \,.\, kw1 \,.\, P_{21} \;+\; \tau \,.\, P_2 \\
P_{21} & = & b_1rf \,.\, P_{22} \;+\; b_1rt \,.\, (kr1 \,.\, P_{21} \;+\; kr2 \,.\, P_{22}) \\
P_{22} & = & enter_2 \,.\, exit_2 \,.\, b_2wf \,.\, P_2
\end{array}$$

Modelling the whole process:

$$\begin{array}{lcl}
L & = & \{b_1rf, b_1rt, b_1wf, b_1wt, b_2rf, b_2rt, b_2wf, b_2wt, \\
 & & \;kr1, kr2, kw1, kw2\} \\
Peterson & = & (\, P_1 \mid P_2 \mid K_1 \mid B_1f \mid B_2f \,) \setminus L
\end{array}$$

Modelling the variables b_1, b_2 and k by process agents:

$$\begin{array}{lclclcl}
B_1f & = & \overline{b_1rf}.B_1f & + & \overline{b_1wf}.B_1f & + & \overline{b_1wt}.B_1t \\
B_1t & = & \overline{b_1rt}.B_1t & + & \overline{b_1wt}.B_1t & + & \overline{b_1wf}.B_1f \\
B_2f & = & \overline{b_1rf}.B_2f & + & \overline{b_1wf}.B_2f & + & \overline{b_1wt}.B_2t \\
B_2t & = & \overline{b_1rt}.B_2t & + & \overline{b_1wt}.B_2t & + & \overline{b_1wf}.B_2f \\
K_1 & = & \overline{kr1}K_1 & + & \overline{kw1}K_1 & + & \overline{kw2}K_2 \\
K_2 & = & \overline{kr2}K_2 & + & \overline{kw2}K_2 & + & \overline{kw1}K_1
\end{array}$$

However, there are small differences: in addition to Walker's version we also take into account that a process may never wish to enter the critical section and model this behavior by additional τ-loops for process P_1 and process P_2. Another point concerns the semantics of the wait-statement in the algorithm. In the process above the busy-waiting semantics is modelled. Alternatively we also want to look at the algorithm with a (non-busy) wait-statement giving different process agents for P_{11} and P_{21}:

$$\begin{aligned}
P_1 &= req_1 . b_1wt . kw2 . P_{11} \;+\; \tau . P_1 \\
P_{11} &= b_2rf . P_{12} \;+\; kr1 . P_{12} \\
P_{12} &= enter_1 . exit_1 . b_1wf . P_1 \\
P_2 &= req_2 . b_2wt . kw1 . P_{21} \;+\; \tau . P_2 \\
P_{21} &= b_1rf . P_{22} \;+\; kr2 . P_{22} \\
P_{22} &= enter_2 . exit_2 . b_2wf . P_2
\end{aligned}$$

7.2 Fairness and Liveness.

We distinguish three concepts: progress, weak fairness and strong fairness. They describe conditions for access to common sources, which are variables in the case here, whenever more than one process is involved. Getting access to a variable is either reading the variable or writing to it.

Progress: Whenever a process continuously wants to have access to a variable then either it eventually can access or infinitely often some other processes access.

Weak fairness: Whenever a process continuously wants to have access to a variable then it eventually gets it.

Strong fairness: Whenever a process infinitely often wants to have access to a variable then it eventually gets it.

The liveness property to prove is, that if a process wishes to enter the critical section then it eventually may do so. We want to show the property for process P_1 and by symmetry arguments it follows also for process P_2. A μ-calculus formula expressing this property is:

$\Phi_1 \equiv \nu Z.[-]Z \wedge [req_1](\mu X.[-]X \vee \langle enter_1 \rangle tt)$

Verifying it for process *Peterson* gives **false** for both interpretations of the wait statement as expected. The property does not hold if we do not include some additional assumptions. For example it is easy to see that in an interleaving based model we also have to make progress explicit. After requesting the critical section one process could stop doing anything, whereas the other one is reading variables continuously.

The whole system is doing something all the time, but, of course, we cannot prove that the one process eventually enters the critical section. What further fairness properties are required is the point which we want to make precise.

From the technical point of view we cannot formulate any fairness condition with μ-calculus expressions for process *Peterson* as encoded above. Every variable access results in a τ-action and it is not visible which process got access to which variable, or which process would like to do so. Walker used additional actions, called probes, in order to make request, entering and exiting of the critical section visible. We will use the same technique and add various probes for variable accesses to the processes.

A property we want to prove is the following:

Requiring progress for all variables, after requesting the critical section a process may eventually enter.

According to the definition of progress we have to add an individual probe to each variable access indicating which variable and which process are involved. The additional probes are b_11, b_12, b_21, b_22, $k1$ and $k2$. The new agents for processes P_1 and P_2 are for interpretation with busy waiting are below.

$$\begin{array}{lcl}
P_1 & = & req1 \, . \, b_1wt.b_11 \, . \, kw2.k1 \, . \, P_{11} \; + \; \tau.P_1 \\
P_{11} & = & b_2rf.b_21 \, . \, P_{12} \; + \; b_2rt.b_21 \, . \, (kr2.k1 \, . \, P_{11} \; + \; kr1.k1 \, . \, P_{12}) \\
P_{12} & = & enter1.exit1.b_1wf.b_11 \, . \, P_1 \\
P_2 & = & req2 \, . \, b_2wt.b_22 \, . \, kw1.k2 \, . \, P_{21} \; + \; \tau.P_2 \\
P_{21} & = & b_1rf.b_12 \, . \, P_{22} \; + \; b_1rt.b_12 \, . \, (kr1.k2 \, . \, P_{21} \; + \; kr2.k2 \, . \, P_{22}) \\
P_{22} & = & enter_2.exit_2 \, . \, b_2wf.b_22 \, . \, P_2 \\
Peterson_2 & = & (\, P_1 \mid P_2 \mid K_1 \mid B_1f \mid B_2f \,) \setminus L
\end{array}$$

The formula expressing liveness under progress conditions is quite large, but the construction is rather uniform, and I try to give a motivation. What is actually expressed is the property: always, after a request, each path has to fulfill the following: either it eventually leads to the possibility of entering the critical section or it fails (one of) the progress conditions. The possibility of failing progress conditions consists then in further disjunctions in the "pure" liveness formula Φ_1.

It is supposed that a process "wishes" to read or write a variable, if it could do it. In a CCS process the states, where a process could have access to a variable are those where it could do a τ-action and afterwards the indicating probe. For example at a state where process P_2 wants to have access to variable k the μ-calculus formula $\langle\tau\rangle\langle k2\rangle tt$ holds. According to this addition of probes we also have to model that a variable access and its probe have to performed as an atomic action. Paths where these actions are not directly subsequent should not be considered and they also fail the assumptions. In the formula this condition is expressed as "whenever a probe can be performed and it is not performed immediately, then this path will not be considered" (e.g. $\ldots \vee (\langle b_1 1\rangle tt \wedge [b_1 1]X)\ldots$). Additionally we assume that if process P_2 may enter the critical section or exit then it will eventually do it.

We will have a closer look to one of the subformulae expressing the possibility to fail a progress condition, e.g.

$\mu X.\ldots\nu Y.\langle\tau\rangle\langle b_1 1\rangle tt \wedge [b_1 1, b_1 2]X \wedge [-b_1 1, b_1 2](X \vee Y)\ldots$

According to the discussions in section 4.2 this combination of fixpoint operators expresses an "eventually always" property. It is fulfilled on all paths, where always access to variable $b_1 1$ is possible (by $\langle\tau\rangle\langle b_1 1\rangle tt$), but only finitely often one of the processes performs an access (by $[b_1 1, b_1 2]X$) and eventually there will be always no access (by $[-b_1 1, b_1 2]Y$). The disjunction $[-b_1 1, b_1 2](X \vee Y)$ is necessary because of the branching structure: imagine a path failing the progress-condition, but on paths branching off there is eventually an $enter_1$ action.

$$\begin{array}{lllllll}
\Phi_2 & \equiv & \nu Z. & [-]Z & \wedge & [req_1]\Phi_2' & \\
\Phi_2' & \equiv & \mu X. & [-]X & \vee & \langle enter_1\rangle tt & \\
& \vee & \nu Y. & \langle\tau\rangle\langle b_1 1\rangle tt & \wedge & [b_1 1, b_1 2]X & \wedge\ [-b_1 1, b_1 2]\ (X \vee Y) \\
& \vee & \nu Y. & \langle\tau\rangle\langle b_1 2\rangle tt & \wedge & [b_1 1, b_1 2]X & \wedge\ [-b_1 1, b_1 2]\ (X \vee Y) \\
& \vee & \nu Y. & \langle\tau\rangle\langle b_2 1\rangle tt & \wedge & [b_2 1, b_2 2]X & \wedge\ [-b_2 1, b_2 2]\ (X \vee Y) \\
& \vee & \nu Y. & \langle\tau\rangle\langle b_2 2\rangle tt & \wedge & [b_2 1, b_2 2]X & \wedge\ [-b_2 1, b_2 2]\ (X \vee Y)
\end{array}$$

$$
\begin{array}{llllll}
\vee & \nu Y. & \langle\tau\rangle\langle k1\rangle tt & \wedge & [k1, k2]X & \wedge\ [-k1, k2]\,(X \vee Y) \\
\vee & \nu Y. & \langle\tau\rangle\langle k2\rangle tt & \wedge & [k1, k2]X & \wedge\ [-k1, k2]\,(X \vee Y) \\
\vee & \nu Y. & \langle enter_2\rangle tt & \wedge & [enter_2]X & \wedge\ [-enter_2]\,(X \vee Y) \\
\vee & \nu Y. & \langle exit_2\rangle tt & \wedge & [exit_2]X & \wedge\ [-exit_2]\,(X \vee Y)
\end{array}
$$

$$
\begin{array}{llll}
\vee & (\langle b_1 1\rangle tt & \wedge & [b_1 1]X) \\
\vee & (\langle b_2 1\rangle tt & \wedge & [b_2 1]X) \\
\vee & (\langle b_1 2\rangle tt & \wedge & [b_1 2]X) \\
\vee & (\langle b_2 2\rangle tt & \wedge & [b_2 2]X) \\
\vee & (\langle k1\rangle tt & \wedge & [k1]X) \\
\vee & (\langle k2\rangle tt & \wedge & [k2]X)\ )
\end{array}
$$

Verifying Φ_2 for $Peterson_2$ shows that only progress conditions are not sufficient for liveness, as expected. Having tried several fairness assumptions, the following turned out to be the weakest one that is sufficient for proving liveness: in addition to the general progress assumptions, weak fairness is necessary for write access to b_1 and b_2 and for both read and write access of variable k. The probes which have to be added to the process agents now have also to distinguish between read and write access for variables b_1 and b_2 getting the set of probes $b_1 1w, b_2 1r, b_2 2w, b_1 2r, k1, k2$ (the other possibilities do not appear in the case here). We get the following process:

$$
\begin{array}{lll}
P_1 & = & req1\ .\ b_1wt.b_1 1w\ .\ kw2.k1\ .\ P_{11}\ +\ \tau.P_1 \\
P_{11} & = & b_2rf.b_2 1r\ .\ P_{12}\ +\ b_2rt.b_2 1r\ .\ (kr2.k1\ .\ P_{11}\ +\ kr1.k1\ .\ P_{12}) \\
P_{12} & = & enter1.exit1.b_1wf.b_1 1w\ .\ P_1 \\[4pt]
P_2 & = & req2\ .\ b_2wt.b_2 2w\ .\ kw1.k2\ .\ P_{21}\ +\ \tau.P_2 \\
P_{21} & = & b_1rf.b_1 2r\ .\ P_{22}\ +\ b_1rt.b_1 2r\ .\ (kr1.k2\ .\ P_{21}\ +\ kr2.k2\ .\ P_{22}) \\
P_{22} & = & enter_2.exit_2\ .\ b_2wf.b_2 2w\ .\ P_2
\end{array}
$$

$$Peterson_3\ =\ (\,P_1\,|\,P_2\,|\,K_1\,|\,B_1f\,|\,B_2f\,)\setminus L$$

The μ-calculus formula Φ_3 expressing the intended liveness property is constructed analogously to Φ_2. Note that the progress conditions for actions $b_1 2w$ etc. do not appear in the formula, simply because they do not appear in the process. Verifying Φ_3 for $Peterson_3$ gave the result **true**.

$$\begin{array}{lllll}
\Phi_3 & \equiv & \nu Z. \ [-]Z & \wedge & [req_1]\Phi'_3 \\
\Phi'_3 & \equiv & \mu X. \ [-]X & \vee & \langle enter_1\rangle tt
\end{array}$$

$$\begin{array}{llllll}
\vee & \nu Y. \ \langle\tau\rangle\langle b_1 1w\rangle tt & \wedge & [b_1 1w]X & \wedge & [-b_1 1w]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle b_1 2r\rangle tt & \wedge & [b_1 1w, b_1 2r]X & \wedge & [-b_1 1w, b_1 2r]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle b_2 1r\rangle tt & \wedge & [b_2 1r, b_2 2w]X & \wedge & [-b_2 1r, b_2 2w]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle b_2 2w\rangle tt & \wedge & [b_2 2w]X & \wedge & [b_2 2w]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle k1\rangle tt & \wedge & [k1]X & \wedge & [-k1]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle k2\rangle tt & \wedge & [k2]X & \wedge & [-k2]\ (X \vee Y) \\
\vee & \nu Y. \ \langle enter_2\rangle tt & \wedge & [enter_2]X & \wedge & [-enter_2]\ (X \vee Y) \\
\vee & \nu Y. \ \langle exit_2\rangle tt & \wedge & [exit_2]X & \wedge & [-exit_2]\ (X \vee Y)
\end{array}$$

$$\begin{array}{llll}
\vee & (\langle b_1 1w\rangle tt & \wedge & [b_1 1w]X) \\
\vee & (\langle b_2 1r\rangle tt & \wedge & [b_2 1r]X) \\
\vee & (\langle b_1 2r\rangle tt & \wedge & [b_1 2r]X) \\
\vee & (\langle b_2 2w\rangle tt & \wedge & [b_2 2w]X) \\
\vee & (\langle k1\rangle tt & \wedge & [k1]X) \\
\vee & (\langle k2\rangle tt & \wedge & [k2]X)
\end{array}$$

For the case of interpreting the wait statement **not** with busy waiting the necessary requirements turn out to be much weaker. In addition to progress only fair writing for the variables b_1 and b_2 is sufficient for liveness. Here also the position of the request-probe makes a difference. In Walker's version of Peterson's algorithm the request-probe was placed after writing b_1 to **true**. In this case we can show that only progress requirements are sufficient to prove liveness. However, one conflict is hidden in this version: process P_1 wishes to get into the critical section, but is not able to set variable b_1 to **true**. Placing the request probe before writing to b_1 leaves the solution of this problem to the fairness conditions.

The proof technique is the same as in the case above and we present only processes and formulae verified.

$$\begin{array}{lcl}
P_1 & = & req_1 \ . \ b_1wt.b_11 \ . \ kw2.k1 \ . \ P_{11} \ + \ \tau.P_1 \\
P_{11} & = & b_2rf.b_21 \ . \ P_{12} \ + \ kr1.k1 \ . \ P_{12} \\
P_{12} & = & enter_1.exit_1 \ . \ b_1wf.b_11 \ . \ P_1 \\
P_2 & = & req_2 \ . \ b_2wt.b_22 \ . \ kw1.k2 \ . \ P_{21} \ + \ \tau.P_2 \\
P_{21} & = & b_1rf.b_12 \ . \ P_{22} \ + \ kr2.k2 \ . \ P_{22} \\
P_{22} & = & enter_2.exit_2 \ . \ b_2wf.b_22 \ . \ P_2 \\
Peterson_4 & = & (\, P_1 \mid P_2 \mid K_1 \mid B_1f \mid B_2f \,) \setminus L
\end{array}$$

Φ_2 expresses simply liveness under progress assumptions. It was evaluated to **false** for $Peterson_4$ and processes P_1 and P_2 as above. For the modification of P_1, where the request probe req_1 comes after $b_1wt.b_11$ ($Peterson_5$), it is the case that Φ_2 does hold!
For the request probe req_1 in the "correct" place as above fair writing for variables b_1 and b_2 has to be guaranteed. The probes indicating write (and read) access for b_1 and b_2 have to be added. The formula Φ_4 giving **true** $Peterson_4$ is as follows:

$$\begin{array}{lcllcl}
\Phi_4 & \equiv & \nu Z. & [-]Z & \wedge & [req_1]\Phi_4' \\
\Phi_4' & \equiv & \mu X. & [-]X & \vee & \langle enter_1\rangle tt
\end{array}$$

$$\begin{array}{llclcl}
\vee & \nu Y. \ \langle\tau\rangle\langle b_11w\rangle tt & \wedge & [b_11w]X & \wedge & [-b_11w]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle b_12r\rangle tt & \wedge & [b_11w, b_12r]X & \wedge & [-b_11w, b_12r]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle b_21r\rangle tt & \wedge & [b_21r, b_22w]X & \wedge & [-b_21r, b_22w]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle b_22w\rangle tt & \wedge & [b_22w]X & \wedge & [b_22w]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle k1, k2\rangle tt & \wedge & [k1, k2]X & \wedge & [-k1, k2]\ (X \vee Y) \\
\vee & \nu Y. \ \langle\tau\rangle\langle k1, k2\rangle tt & \wedge & [k1, k2]X & \wedge & [-k1, k2]\ (X \vee Y) \\
\vee & \nu Y. \ \langle enter_2\rangle tt & \wedge & [enter_2]X & \wedge & [-enter_2]\ (X \vee Y) \\
\vee & \nu Y. \ \langle exit_2\rangle tt & \wedge & [exit_2]X & \wedge & [-exit_2]\ (X \vee Y) \\
\vee & (\langle b_11w\rangle tt & \wedge & [b_11w]X) & & \\
\vee & (\langle b_21r\rangle tt & \wedge & [b_21r]X) & & \\
\vee & (\langle b_12r\rangle tt & \wedge & [b_12r]X) & & \\
\vee & (\langle b_22w\rangle tt & \wedge & [b_22w]X) & & \\
\vee & (\langle k1\rangle tt & \wedge & [k1\,]X) & & \\
\vee & (\langle k2\rangle tt & \wedge & [k2\,]X) & &
\end{array}$$

7.3 Experimental Results.

The local version of the Gauß elimination algorithm presented in Section 6.4 was implemented by Wallner [Wal93] and the processes and formulae of this chapter have been verified using this implementation. The program is written in C and Binary Decision Diagrams (BDDs) [Bry86] have been chosen as data structure for Boolean expressions. The BBD package from Carnegie Mellon University was used. The program was run on a Sun UltraSparc 1. The transformation from CCS agents to transition systems as input for the program was performed with the Edinburgh Concurrency Workbench.

However, experiments showed that BDDs are probably not the most suitable choice for our algorithm: each substitution step during the algorithm makes a composition of BBDs necessary. The size of the BDDs grew more than expected and made frequent and time-consuming reordering necessary. Below we list the results from the verification procedures. BDD sizes are included and here and we took only into account the size of the BDD representing the right-hand side of the

Verification of Petersons's mutex algorithm					
Version of Peterson	2	3	4	4	5
states	203	203	139	139	139
formula	Φ_2	Φ_3	Φ_2	Φ_4	Φ_2
fixpoints	10	10	10	10	10
result	false	true	false	true	true
time	8 min	13 min	1 min	1 min	1 min
equations created	352	456	236	244	185
% of all equations	17 %	22 %	17 %	18 %	13 %
maximal BDD size	5689	9868	2123	2123	289
average BDD size	577	423	231	175	49
substitution steps	106508	202121	50946	51182	23313
elimination steps	11219	11464	5078	4946	4088

variable of interest ("the first equation"). Each formula introduced in the previous section was verified for the version of Peterson's algorithm containing the relevant probes for this case. All formulae express always-properties, which makes an evaluation of the formula at all states necessary. It is to be expected that local model checking is no advantage in this case. However, it turned out, that only 13-22% of the possible equations had to be created.

Chapter 8

Equivalent techniques.

The model checking problem for the modal μ-calculus has been treated also within other frameworks, and there exist reductions to problems in automata theory and theory of games. Chapter 5 contains reductions of the model checking problem to Boolean equation systems and vice versa. In this chapter we will show the equivalence of solving Boolean equation systems on one hand, and the nonemptiness problem for alternating automata as well as the decision problem for games, i.e. which player has a winning strategy, on the other hand. From the equivalence it follows that algorithms solving one problem can be transformed in order to solve the other problems. Furthermore the equivalence allows us to apply the various properties for Boolean equation systems from chapter 3 and Section 3.2 also to the kind of alternating automata and games considered.

8.1 Alternating automata.

In this section we show the equivalence of alternating automata on infinite words over a 1-letter alphabet with a parity acceptance condition and Boolean equation systems. It follows then according to the results of section 5.2 that the nonemptiness problem for these alternating automata and the model checking problem are equivalent too.

For an overview over automata on infinite words and trees see [Tho90], for alternating automata also [Var95].

Words and trees.

Let Σ be a finite nonempty alphabet. A finite word over Σ is a finite sequence $a_0, \ldots, a_n$ of elements of Σ. The set of finite words over Σ is denoted by Σ^*. An infinite word over Σ is a infinite sequence $a_0, a_1, \ldots$ of elements of Σ. The set of infinite words over Σ is denoted by Σ^ω.
A **tree** τ over the alphabet Σ is a directed, acyclic graph. Each node n is labelled by an element of Σ, written as $\tau(n) \in \Sigma$. The set of nodes may be either finite or infinite. There exists one node without predecessor, the **root** of τ. Each other node has one unique predecessor, its **parent**, and a finite number of successors, its **children**. The number of its children is the **arity** of a node. Nodes without children are called **leaves**. A **branch** b of a tree τ is a sequence $b_1 b_2 \ldots$, such that b_0 is the root of τ and each b_i is the parent of b_{i+1}. It is either finite, ending in a leaf, of infinite. Given a branch b of a tree we define the set $lim(b) \subseteq \Sigma$ as all elements a of Σ such that infinitely many nodes of b are labelled with a. Note that if b is finite, then $lim(b) = \emptyset$.

Alternating automata.

Alternating automata are a generalization of nondeterministic automata. For our purpose automata over an alphabet containing a single letter are sufficient.
An alternating automaton A is here defined as a tuple $(\{a\}, S, s^0, \rho, \Omega)$, where

- $\{a\}$ is a 1-letter alphabet,
- S is the set of states of A,
- $s^0 \in S$ is the initial state,
- $\rho : \{a\} \times S \rightarrow \mathcal{B}^+(S)$ a transition function, which maps a state of S (and the symbol a) to a negation free Boolean expression over S,
- Ω is an acceptance condition which has to be specified.

For a subset S' of S define an environment on states $\theta_{S'}$ such that for all $s \in S'$ we have that $\theta_{S'}(s) = \mathsf{true}$ and false for all other states in $S \setminus S'$. A subset S' of S **satisfies** a negation free Boolean expression f over S, if $f(\theta_{S'}) = \mathsf{true}$. For example when f is represented in disjunctive normal form, all the states occurring in one disjunct form a set which satisfies f.

A run of an automaton A over the (infinite) word $\omega = a, a, a, \ldots$ is a tree r over S with the properties:

- the root of r is labelled by the initial state s^0
- if a node n has the children $n_1, \ldots, n_k$, and n is labelled by a state s, where $\rho(a, s) = f$, then the label set $\{r(n_1), \ldots r(n_k)\}$ satisfies f.

A run r of A is **accepting** if the acceptance condition Ω holds, which here is a **parity condition**. Ω includes a labelling of the states with colours $\{1, \ldots, m\}$ for some $m \in I\!N$, and an **acceptance set** $F \subseteq S$, which contains for a subset of colours all states of these colours. The acceptance condition is:

- every finite branch ends in a leaf labelled with a state s, such that $\rho(a, s) = \mathsf{true}$
- for every infinite branch b the state with the least label in $lim(b)$ is contained in F.

An automaton is **empty** if it has no accepting run.

We may also mention now that an alternating automaton over a single-letter alphabet as defined above can be interpreted as a non-deterministic tree-automaton and vice versa. In this case a run of an automaton A over the (infinite) tree τ is a tree r over S with the properties:

- the root of r is labelled by the initial state s^0
- if a node n of r has the children $n_1, \ldots, n_k$, then for $\rho(a, r(n)) = f$ the set of labels $\{r(n_1), \ldots r(n_k)\}$ satisfies f
- define for $s \in S$ the automaton A_s as A, but with initial state s; for each node n of r with children $n_1, \ldots, n_k$ there exists a node n' in τ with children $n'_1, \ldots, n'_k$, such that every subtree of r rooted with n_i is a run of $A_{r(n_i)}$ over the subtree of τ rooted with n'_i.

The acceptance condition for a run is as above.

Proposition 8.1 An alternating automaton A over infinite strings and a 1-letter alphabet is nonempty iff the interpretation of A as nondeterministic tree-automaton is nonempty.

From Boolean equation systems to alternating automata.

Given a Boolean equation system $\mathcal{E}$ and an environment θ we construct an automaton $A_{\mathcal{E},\theta}$ as follows.
$A_{\mathcal{E},\theta} = (\{a\}, S_{\mathcal{E}}, X_i, \rho_{\mathcal{E},\theta}, \Omega_{\mathcal{E},\theta})$, where

- $S_{\mathcal{E}}$ is the set of all variables of $\mathcal{E}$, i.e. $S_{\mathcal{E}} = lhs(\mathcal{E}) \cup rhs(\mathcal{E})$.
- Some variable X_i of $\mathcal{E}$ is taken as initial state.
- If $\sigma X = f$ is an equation of $\mathcal{E}$, we define $\rho(a, X) = f$, otherwise $\rho(a, X) = \theta(X)$.
- The acceptance set F contains all states X where $\nu X = f$ is an equation with a greatest fixpoint operator in $\mathcal{E}$. The labelling of the states follows the order of the variables in $\mathcal{E}$: the first variable gets the label 1, the second 2 etc.. States which do not correspond to left-hand side variables in $\mathcal{E}$ are only labels of leaves in all runs. Hence their labelling is irrelevant.

Theorem 8.2 For a Boolean equation system $\mathcal{E}$ and an environment θ it is the case that $([\![\mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$ iff $A_{\mathcal{E},\theta}(\{a\}, S_{\mathcal{E}}, X_i, \rho_{\mathcal{E},\theta}, \Omega_{\mathcal{E},\theta})$ is nonempty. Moreover $A_{\mathcal{E},\theta}$ has size of $O(|\,\mathcal{E}\,|)$.

The proof is in the appendix.

From alternating automata to Boolean equation systems.

The transformation from an alternating automaton over a 1-letter alphabet with parity condition to a Boolean equation system is simple. Given an automaton $A(\{a\}, S, s^0, \rho, \Omega)$ we construct a Boolean equation system $\mathcal{E}_A$ as follows:

- The set of states S is interpreted as set of Boolean variables.

- For each $s \in S \cap F$ there is an equation $\nu s = \rho(a, s)$ in $\mathcal{E}_A$.
- For each $s \in S \setminus F$ there is an equation $\mu s = \rho(a, s)$ in $\mathcal{E}_A$.
- The acceptance condition Ω includes a labelling of the states of S. If for $s_i, s_j \in S$ the label of s_i is lower than the label of s_j then $s_i \lhd s_j$ in $\mathcal{E}_A$, i.e. the equation $\sigma_i s_i = \rho(a, s_i)$ is before $\sigma_i s_j = \rho(a, s_j)$ in $\mathcal{E}_A$. (If s_i and s_j carry the same label then they are in the same block and their order is irrelevant.)

Theorem 8.3 For an alternating parity automaton over a 1-letter alphabet $A(\{a\}, S, s^0, \rho, \Omega)$ there exists a Boolean equation system $\mathcal{E}_A$ of size $O(|A|)$, such that for any environment θ it is:

$A(\{a\}, S, s^0, \rho, \Omega)$ is nonempty iff $([\![\mathcal{E}_A]\!]\,\theta)(s^0) = \mathsf{true}$.

Proof: Take the Boolean equation system $\mathcal{E}_A$ as constructed above and transform it back to an automaton $A_{\mathcal{E}_A}$ as in the previous section. It is easy to see that we get the original automaton up to labelling. The equivalence follows then from theorem 8.2. □

Now the equivalence of alternating automata and model checking problems follows easily:

Theorem 8.4 For an alternating parity automaton $A(\{a\}, S, s^0, \rho, \Omega)$ over a 1-letter alphabet there exists a proposition of the modal μ-calculus Φ and a model $\mathcal{M}$ with the state space $\mathcal{S}$, such that for some renaming function $\lambda : S \to \mathcal{S}$, any environment θ and any valuation $\mathcal{V}$ it is:

$\lambda(s^0) \in \|\Phi\|^{\mathcal{T}}_{\mathcal{V}}$ iff $A(\{a\}, S, s^0, \rho, \Omega)$ is nonempty. It is $ad(\Phi) \leq |F| + 1$ and the $\mathcal{M}$ is of size $O(|A|^2)$.

Proof: Apply theorems 8.2, 8.3 and 5.2. □

Complexity and relation to other work.

From the equivalence proved above and the results from section 6.5 we know that the nonemptiness problem for alternating parity automata over a 1-letter alphabet is contained in NP ∩ co-NP. In this section we want to relate this result to other complexity results for the same problem in the theory of automata. For that purpose we consider more standard acceptance conditions, the Büchi and Rabin acceptance conditions.

The Büchi acceptance condition for a run r of an (alternating) automata consists of an acceptance set $F \subseteq S$ and the requirement, that $lim(b) \cap F \neq \emptyset$ for every branch b of r.

The Rabin condition includes a set of accepting pairs $\{(L_1, U_1), \ldots, (L_n, U_n)\}$ and the requirement for a run r to be accepted is: for each branch b of r there exists an $i \in \{1, \ldots, n\}$ such that $lim(b) \cap U_i \neq \emptyset$ and $lim(b) \cap L_i = \emptyset$.

The languages accepted by alternating automata on infinite words are the same for all these three acceptance conditions (see e.g. Lindsay [Lin88]); it is the class of ω-regular languages.

However, concerning the emptiness problem for alternating automata over a 1-letter alphabet the acceptance conditions make a difference.

There exist linear translations from Büchi automata to parity automata and from parity automata to Rabin automata, which are essentially only transformations of the acceptance conditions.

For the case of Büchi automata the states have to be equipped with labels from $\{1, 2\}$. The labels are chosen in such a way, that each state contained in the acceptance set F gets the label 1 and each other state not contained in F gets the label 2. The labelling together with the acceptance set F is then the equivalent parity acceptance condition. Thus every alternating Büchi automaton A can be mapped to an alternating parity automaton, and further with theorem 8.3 to a Boolean equation system $\mathcal{E}_A$, such that the Büchi automaton with initial state s^0 is nonempty iff $([\![\mathcal{E}_A]\!]\,\theta)(s^0) = \mathsf{true}$ for any environment θ. From the structure of Büchi acceptance conditions and the way of construction of $\mathcal{E}_A$ it follows that $\mathcal{E}_A$ has alternation depth of at most 2; the first equations have greatest fixpoint operators, the last equations have least fixpoint operators. Applying complexity results from chapter 6.2 we get the proposition below. It follows also from [Var95], prop.5 and proposition 8.1.

Proposition 8.5 The nonemptiness problem for alternating Büchi automata on infinite words over a 1-letter alphabet is decidable in quadratic time and space.

For the reduction of a parity automaton to a Rabin automaton we also just the acceptance condition needs to be transformed. For each colour $i \in \{1, \ldots, m\}$ we define $L_i \overset{\text{def}}{=} \{s \in F \mid s \text{ is labelled with } i\}$ and $U_i \overset{\text{def}}{=} \{s \in S \mid s \text{ has a label lower than } i\}$. Note that an accepting pair (L_i, U_i) with $L_i = \emptyset$ can be removed from the set of accepting pairs, because it accepts nothing. It is easy to see that this Rabin condition accepts the same runs as the original parity condition and vice versa. However, here the nonemptiness problem follows from [EJ88] and proposition 8.1.

Proposition 8.6 The nonemptiness problem for alternating Rabin automata on infinite words over a 1-letter alphabet is NP-complete.

Representing μ-calculus formulae as automata already has a long tradition (e.g. [SE84, Niw88, EJ91, Kai96]). The modal μ-calculus was shown to be expressively equivalent to automata on infinite trees. Among known results the closest to ours is the equivalence of the model checking problem and nonemptiness of nondeterministic tree-automata with parity acceptance condition from Emerson, Jutla and Sistla [EJS93]. With proposition 8.1 the equivalence presented here and their result are interderivable. Another approach (e.g. see [Var95, BVW94]) is to represent a formula of the modal μ-calculus and also the transition system as (alternating, amorphous) Rabin tree-automata. If the product-automaton of these is nonempty, then the formula holds at the initial state of the transition system. However, this emptiness problem is NP-complete, and hence the problems are not equivalent. In this approach the NP $\cap$ co-NP complexity of the model checking problem follows from complementation arguments.

8.2 Graph games.

Starting from the framework of Boolean equation systems we can derive graph games as defined in [Sti96] and show the equivalence of both approaches. In this section we assume Boolean equation systems being closed and in standard form.

A **game graph** $\mathcal{G}$ consists of a set of vertices $\{1, \ldots, n\}$, each of them carrying one label from {I, II} and another from $\{\mu, \nu\}$ [1] The graph $\mathcal{G}$ contains one or two edges of the form $i \rightarrow j$ for each vertex i. The size $|\mathcal{G}|$ is defined as usual as sum of the number of vertices and the number of edges of $\mathcal{G}$.

A **play** p on the game graph $\mathcal{G}_{\mathcal{E}}$ is an infinite sequence of vertices chosen by two players, player I and player II. The play starts at some initial vertex i. Whenever the current vertex is labelled with I then player I has to move and chooses one of the successors, which then is the current vertex. Dually, if it is labelled by II then player II has to move and chooses a successor.

A **strategy** for a player is a decision function from the play done so far to the next move.

Deciding who is the winner of a play p requires considering the set $lim(p)$ of all vertices which have been visited infinitely often. If the least vertex of all vertices in $lim(p)$ is labelled with μ then player I wins; if it is labelled with ν then player II wins.

A player has a **winning strategy** for the game on $\mathcal{G}_{\mathcal{E}}$ with initial vertex i if she can win every play.

A **history free winning strategy** is a winning strategy where the choice of a successor does not depend on the initial sequence of the play done

[1]In [Sti96] game graphs are defined in such a way that each vertex carries only one label from {I, II}. For getting from the definition there to ours we equip each I-node with a μ and each II-node with a ν. For the other way round we have to take care of two cases: vertices carrying a I and a ν, and, dually vertices carrying a II and a μ. In both cases a extra vertex has to be introduced which inherits all the successors of the one considered, but is then the only immediate successor of the original one. In the first case the original vertex gets the label II, its new successor the label I, dually in the second case. In all other cases the labels μ od ν may just be removed.

so far. This means that in a history free winning strategy for player I (II) there exists a unique choice of a successor at every I-labelled (II-labelled) vertex.

From Boolean equation systems to graph games.

Given a Boolean equation system $\mathcal{E}$ we will define a game graph $\mathcal{G}_{\mathcal{E}}$. Recall that for a given Boolean equation system $\mathcal{E}$ the dependency graph (see section 6.1) consists of a set of vertices $\{1, \ldots, n, \mathsf{true}, \mathsf{false}\}$, one for each left-hand side variable of $\mathcal{E}$ and two for true and false. If there is an equation $\sigma X_i = X_j \vee X_k$ $(X_j \wedge X_k)$ in $\mathcal{E}$ then there will be edges $i \rightarrow j$ and $i \rightarrow k$ in the dependency graph.
Essentially the game graph $\mathcal{G}_{\mathcal{E}}$ for $\mathcal{E}$ is its dependency graph where additionally each vertex carries two more labels. For every equation $\sigma X_i = f$ in $\mathcal{E}$ vertex i of $\mathcal{G}_{\mathcal{E}}$ is labelled with σ. Vertex false gets the label μ, true gets the label ν. If $\sigma X_i = X_j \wedge X_k$ is an equation of $\mathcal{E}$ then vertex i is labelled with I, and all other vertices are labelled with II. Two more edges are added to $\mathcal{G}_{\mathcal{E}}$ for technical reasons: $\mathsf{false} \rightarrow \mathsf{false}$ and $\mathsf{true} \rightarrow \mathsf{true}$.

Theorem 8.7 Player II has a winning strategy for the game on $\mathcal{G}_{\mathcal{E}}$ with initial vertex i iff $([\![\mathcal{E}]\!]\, \theta)(X_i) = \mathsf{true}$. Moreover $|\mathcal{G}_{\mathcal{E}}| = O(|\mathcal{E}|)$.

The proof can be found in the appendix.
The existence of history-free winning strategies follows easily from the corresponding properties for Boolean equation systems (see also [Sti96]).

Proposition 8.8 If player I (II) has a winning strategy for the game on $\mathcal{G}_{\mathcal{E}}$ with initial vertex i, then she has also a history free winning strategy.

Proof: Follows immediately from lemma 3.36 and theorem 8.7. $\square$

From graph games to Boolean equation systems.

From a game graph $\mathcal{G}$ we derive a Boolean equation system $\mathcal{E}_{\mathcal{G}}$.

- If vertex i of $\mathcal{G}$ is labelled with σ there will be an equation $\sigma X_i = f_i$ in $\mathcal{E}_{\mathcal{G}}$. There will be no equations for true and false.
- If vertex i has label I and $i \rightarrow j$ and $i \rightarrow k$ are edges in $\mathcal{G}$ then $\sigma X_i = X_j \vee X_k$ is an equation of $\mathcal{E}_{\mathcal{G}}$.
- If vertex i has label II and there are edges $i \rightarrow j$ and $i \rightarrow k$ in $\mathcal{G}$ then $\sigma X_i = X_j \wedge X_k$ is an equation of $\mathcal{E}_{\mathcal{G}}$. If there is just one edge $i \rightarrow j$ from i then $\sigma X_i = X_j$ is an equation of $\mathcal{E}_{\mathcal{G}}$.
- For $i < j$ it is $X_i \lhd X_j$ in $\mathcal{E}_{\mathcal{G}}$.

Theorem 8.9 Player II has a winning strategy for the game on $\mathcal{G}$ with initial vertex i iff $([\![\mathcal{E}_{\mathcal{G}}]\!]\,\theta)(X_i) = \mathsf{true}$. Moreover $|\mathcal{G}| = O(|\mathcal{E}_{\mathcal{G}}|)$.

Proof: Follows immediately from the fact, that the game graph defined by $\mathcal{E}_{\mathcal{G}}$ is again the original game graph, i.e. $\mathcal{G} = \mathcal{G}_{\mathcal{E}_{\mathcal{G}}}$, together with theorem 8.7 $\square$

With linear reductions in both directions we have shown the equivalence of determining whether there exists a winning strategy for one player in a game and solving Boolean equation systems. This is another proof that the decision problem for graph games is in NP $\cap$ co-NP. With the equivalence of the latter and the model checking problem in the modal μ-calculus we get immediately an answer to an open question in [Sti96].

Theorem 8.10 For a game graph $\mathcal{G}$ there exists a proposition of the modal μ-calculus Φ and a model $\mathcal{M}$ with the state space $\mathcal{S}$, such that for a renaming function $\lambda : \{1, \ldots, n\} \rightarrow \mathcal{S}$ and any valuation $\mathcal{V}$ it is:

$\lambda(i) \in \|\Phi\|^{\mathcal{T}}_{\mathcal{V}}$ iff player II has a winning strategy for the game on $\mathcal{G}$ with initial vertex i. Moreover $|\mathcal{M}| = O(|\mathcal{G}|^2)$.

Proof: Follows from theorems 5.2, 8.7 and 8.9. $\square$

Relation to other techniques.

In this section a play has been defined as an infinite sequence of vertices. The analogy to the automata approach is obvious: each play can be interpreted as a branch of a run on an alternating automation as defined in the previous section. The branch is accepted iff player II wins the play.
Equally a play on a game graph $\mathcal{G}$ can be defined as a finite sequence of vertices ([Sti96]). Then the termination condition for a play is, that a vertex has been visited twice. Player II wins such a finite play, if the vertex with the least label between the first and second occurrence of the one visited twice is labelled with ν, otherwise player I wins. With this definition a play is equivalent to a path in a tableau as defined in Section 6.3. The question whether there exists a winning strategy for player II or a successful tableau are the same. Consequently an algorithm which solves the decision problem for finite plays has to deal with same redundancy problem as tableaux have. One algorithm solving this problem in a top-down manner is contained in [Mad92]. However, the criteria for possible "reuse" of prior information are involved. A more efficient and simple algorithm avoiding redundancy is Gauß elimination of section 6.4.

Chapter 9

Infinite Boolean equation systems.

So far we have been concerned with model checking only for finite state systems. It has been shown that there the problems of solving Boolean equation systems and model checking are equivalent. Models with an infinite state space easily arise when e.g. considering systems with unbounded buffers or programs using recursive data-structures such as natural numbers or trees. In this chapter the framework of Boolean equation systems will be extended to the infinite case. The model checking problem for the modal μ-calculus and systems with (possibly) infinite state space on one hand, and solving infinite Boolean equation systems on the other hand will be shown to be equivalent. However, such an equivalence is only useful, if there exists a finitely representable method for solving infinite Boolean equation systems. Here approximation techniques are not applicable. We present an elimination method similar to Gauss elimination in section 6.4 based on a representation of infinite Boolean equation systems by set based Boolean equation systems. This elimination method is closely related to the tableau method of Bradfield and Stirling [BS91, Bra92]. It combines the top-down approach of the tableau with a bottom-up evaluation. In a tableau for an infinite state system the same effect can occur as in the finite case: the tableau might contain many copies of similar

subtrees. The bottom-up evaluation avoids this kind of redundancy. To determine whether a tableau is successful or not it is necessary to investigate so called extended paths. It turns out that the strategy of the elimination method simplifies the success criterion. The nondeterminism contained in the tableau method is, of course, still contained in the elimination algorithm presented here. It is intended that an intelligent prover makes use of her (not generally formalizable) knowledge about system and property to prove in order to deal with the nondeterministic parts of the algorithm.
We define infinite Boolean equation systems and show how properties for the case of finite Boolean equation systems can be transfered. Set based Boolean equation systems are introduced as a finite representation for infinite Boolean equation systems. We show a substitution step and elimination step similar to the ones in the Gauß elimination of section 6.4. With these an algorithm is formulated describing the bottom-up version of the tableau method in [BS91, Bra92]. Small examples demonstrate the technique.

9.1 Definitions.

In this section we define syntax and semantics of infinite Boolean equation systems. Furthermore, we show that for each infinite Boolean equation system there exists a system in conjunctive form such that both systems have the same solution. In terms of games this says that there exist history free winning strategies.
In the case of infinite Boolean equation systems we have to deal with two kinds of infinity: on one hand there might be an infinite number of equations, on the other hand the right-hand sides of each equation may consist of infinite conjunctions or disjunctions. However, what still has to be finite is the nesting depth of infinite Boolean equation systems. An infinite Boolean equation system therefore consists of a finite sequence of blocks, where a block is a possibly infinite set of Boolean equations all having the same fixpoint operator. The syntax of an infinite Boolean equation system is as follows.

Definition 9.1 The set of positive infinite Boolean expressions over a countable set $\mathcal{X}$ of variables is denoted by $\mathbb{B}^+_\infty(\mathcal{X})$. Each of its elements is of the form $\bigvee_{i\in I} X_i$, $\bigwedge_{i\in I} X_i$ or X_i where I is a countable index set and $X_i \in \mathcal{X} \cup \{\mathsf{true}, \mathsf{false}\}$.

An infinite Boolean equation is of the form $\sigma X = f$, where $\sigma \in \{\mu, \nu\}$, $X \in \mathcal{X}$ and $f \in \mathbb{B}^+_\infty(\mathcal{X})$.

A block $\sigma\mathcal{B}$ is a set of infinite Boolean equations $\sigma X_j = f_j$, all having the same fixpoint operator σ, $j \in J$ and J is a countable index set.

An infinite Boolean equation system $\mathcal{E}$ is a finite sequence of blocks $\sigma_1\mathcal{B}_1 \ldots \sigma_n\mathcal{B}_n$ for some $n \in \mathbb{N}$.

Again, for technical reasons, we assume that in an infinite Boolean equation system there are no two equations having the same variable on the left-hand side. The definitions of the set of left-hand side variables $lhs(\mathcal{E})$ and right-hand side variables $rhs(\mathcal{E})$ of an infinite Boolean equation system are as in the finite case. Also environments $\theta : \mathcal{X} \to \mathbb{B}$ are defined as in the finite case. We have $(\bigvee_{i\in I} X_i)(\theta) = \mathsf{true}$, if $X_i = \mathsf{true}$ or $\theta(X_i) = \mathsf{true}$ for some $i \in I$, and false otherwise. Dually $(\bigwedge_{i\in I} X_i)(\theta) = \mathsf{false}$ if for some $i \in I$ either $X_i = \mathsf{false}$ or $\theta(X_i) = \mathsf{false}$. For some index set I and $b_I \in \mathbb{B}^I$ we denote by $\theta[X_I/b_I]$ the simultaneous substitution of all $X_i \in \mathcal{X}$ by b_i for $i \in I$, such that $\theta[X_I/b_I](X_i) = b_i$ for $i \in I$ and otherwise $\theta[X_I/b_I](X_i) = \theta(X_i)$.
The semantic of an infinite Boolean equation system is defined recursively. In contrast to the finite case in each step an infinite Boolean equation system is not reduced to systems with one equation less, but with one block less.

Definition 9.2 Let $\sigma\mathcal{B}\ \mathcal{E}$ be an infinite Boolean equation system, where $lhs(\sigma\mathcal{B}) = \{X_i \in \mathcal{X} \mid i \in I\}$ for some index set I, and $b \in \mathbb{B}^I$.

$$\begin{array}{rcl}
[\![\epsilon]\!]\,\theta & = & \theta \\
[\![\sigma\mathcal{B}\ \mathcal{E}]\!]\,\theta & = & [\![\mathcal{E}]\!]\,\theta[X_I/\sigma X_I.\mathcal{B}([\![\mathcal{E}]\!]\,)] \quad \text{, where} \\
\mu X_I.\mathcal{B}([\![\mathcal{E}]\!]\,) & = & \bigcap\{b \in \mathbb{B}^I \,|\, \forall i \in I. b_i \geq f_i([\![\mathcal{E}]\!]\,\theta[X_I/b])\} \\
\nu X_I.\mathcal{B}([\![\mathcal{E}]\!]\,) & = & \bigcup\{b \in \mathbb{B}^I \,|\, \forall i \in I. b_i \leq f_i([\![\mathcal{E}]\!]\,\theta[X_I/b])\}
\end{array}$$

With this definition of the semantic we can make use of all the properties proved for fixpoint equation systems in chapter 3. In this case we interpret a block above as one vector valued fixpoint equation. However, we often want to argue about a single Boolean equation, not about a whole block. Therefore we need the property below about splitting of blocks. Then it is also possible to split a single equation from a block and consider it as one block. When arguing about infinitely many Boolean equations then blocks containing infinitely many equations should be split before applying the relevant lemmata.

Lemma 9.3 Let

- $\mu\mathcal{B}$, $\mu\mathcal{B}_1$ and $\mu\mathcal{B}_2$ be blocks, where $\mu\mathcal{B} = \mu\mathcal{B}_1 \cup \mu\mathcal{B}_2$.
- $\mathcal{E}_1$, $\mathcal{E}_2$ be infinite Boolean equation systems,
- and θ an environment.

Then $[\![\mathcal{E}_1\ \mu\mathcal{B}\ \mathcal{E}_2]\!]\,\theta = [\![\mathcal{E}_1\ \mu\mathcal{B}_1\ \mu\mathcal{B}_2\ \mathcal{E}_2]\!]\,\theta$.

Proof: Follows from Bekič's Theorem 2.24 and the transformation from fixpoint expressions to fixpoint equation systems. Details are left to the reader. □

We now show a property which is the infinite version of lemma 3.36. It says that for every Boolean equation system $\mathcal{E}$ and environment θ there exists a conjunctive Boolean equation system $\mathcal{E}'$ such that $\mathcal{E}' \leq \mathcal{E}$, and $[\![\mathcal{E}']\!]\,\theta = [\![\mathcal{E}]\!]\,\theta$. In terms of games this means that also in the infinite case there are history free winning strategies.

Theorem 9.4 Given an infinite Boolean equation system $\mathcal{E} = \sigma_1\mathcal{B}_1 \ldots \sigma_n\mathcal{B}_n$ and an environment θ there exists an infinite Boolean equation system $\mathcal{E}' = \sigma_1\mathcal{B}'_1 \ldots \sigma_n\mathcal{B}'_n$ such that $\mathcal{E}'$ contains no disjunctions on the right-hand side, such that

- If $\sigma_j X_k = \bigwedge_{i\in I} X_i$ is an equation in block $\mathcal{B}_j$ of $\mathcal{E}$ then it is also an equation in $\mathcal{B}'_j$ of $\mathcal{E}'$.
- If $\sigma_j X_k = X_i$ is an equation in block $\mathcal{B}_j$ of $\mathcal{E}$ then it is also an equation in $\mathcal{B}'_j$ of $\mathcal{E}'$.

- If $\sigma_j X_k = \bigvee_{i \in I} X_i$ is an equation in block $\mathcal{B}_j$ of $\mathcal{E}$ and I is nonempty, then for some $i \in I$ the equation $\sigma_j X_k = X_i$ is in block $\mathcal{B}'_j$ of $\mathcal{E}'$. If I is empty then $\sigma_j X_k = \mathsf{false}$ is an equation of $\mathcal{B}'_j$ of $\mathcal{E}'$.
- $[\![\mathcal{E}]\!]\,\theta = [\![\mathcal{E}']\!]\,\theta$

A proof can be found in the appendix.

9.2 Equivalence to the model checking problem.

Essentially the transformation of the model checking problem for infinite state spaces to infinite Boolean equation systems does not differ from the finite case. However, as we allow for infinite Boolean equation systems only one conjunct or one disjunct on the right-hand side of each equation we have to introduce new variables.
The transformation function $\mathbf{E}_\infty$ maps a pair $(\Phi, \mathcal{M})$ consisting of a modal μ-calculus formula Φ and a model $\mathcal{M}$ with a possibly countable state space $\mathcal{S}$ to an infinite Boolean equation system.
The function $\mathbf{E}_\infty$ performs the transformations from a nested fixpoint formula to a fixpoint equation system and creates the basic block structure of the whole system. By introduction of new variables and constants it also reduces each right-hand side expression to a single variable, constant, modality, disjunction or conjunction (and no combination of those). $\mathbf{E}_\infty$ refers to a set of functions $\{\mathbf{E_1}, \mathbf{E_2}, \ldots\}$, which create the Boolean equations within one block. Each $\mathbf{E_i}$ for $i \in I\!N$ is related to state s_i of the transition system.
We omit the argument $\mathcal{M}$ of $\mathbf{E}_\infty$ when it is clear from the context.

$$\begin{aligned}
\mathbf{E}_\infty(Q) &= \epsilon \\
\mathbf{E}_\infty(X) &= \epsilon \\
\mathbf{E}_\infty(\Phi_1 \wedge \Phi_2) &= \mathbf{E}_\infty(\Phi_1)\ \mathbf{E}_\infty(\Phi_2) \\
\mathbf{E}_\infty(\Phi_1 \vee \Phi_2) &= \mathbf{E}_\infty(\Phi_1)\ \mathbf{E}_\infty(\Phi_2) \\
\mathbf{E}_\infty([a]\Phi) &= \mathbf{E}_\infty(\Phi)
\end{aligned}$$

$$\begin{array}{rcl}
\mathbf{E}_\infty(\langle a\rangle\Phi) & = & \mathbf{E}_\infty(\Phi) \\
\mathbf{E}_\infty(\sigma X.\Phi_1 \vee \Phi_2) & = & (\sigma X_1 = \mathbf{E}_1(X' \vee X''))\ (\sigma X_2 = \mathbf{E}_2(X' \vee X''))\ldots \\
 & & \mathbf{E}_\infty(\sigma X' = \Phi_1)\ \mathbf{E}_\infty(\sigma X'' = \Phi_2) \text{ for fresh } X', X'' \\
\mathbf{E}_\infty(\sigma X.\Phi_1 \wedge \Phi_2) & = & (\sigma X_1 = \mathbf{E}_1(X' \wedge X''))\ (\sigma X_2 = \mathbf{E}_2(X' \wedge X''))\ldots \\
 & & \mathbf{E}_\infty(\sigma X' = \Phi_1)\ \mathbf{E}_\infty(\sigma X'' = \Phi_2) \text{ for fresh } X', X'' \\
\mathbf{E}_\infty(\sigma X.\Phi) & = & (\sigma X_1 = \mathbf{E}_1(\Phi))\ (\sigma X_2 = \mathbf{E}_2(\Phi))\ldots\ \mathbf{E}_\infty(\Phi) \\
 & & \text{if } \Phi \text{ is not a conjunction or disjunction}
\end{array}$$

and for $i \in I\!N$

$$\begin{array}{rcl}
\mathbf{E_i}(Q) & = & \begin{cases} \mathsf{true} & \text{if } s_i \in \mathcal{V}(Q) \\ \mathsf{false} & \text{else} \end{cases} \\
\mathbf{E_i}(X) & = & X_i \\
\mathbf{E_i}([a]\Phi) & = & \bigwedge_{s_i \xrightarrow{a} s_j} \mathbf{E_j}(\Phi) \\
\mathbf{E_i}(\langle a\rangle\Phi) & = & \bigvee_{s_i \xrightarrow{a} s_j} \mathbf{E_j}(\Phi) \\
\mathbf{E_i}(\sigma X.\Phi) & = & X_i
\end{array}$$

The transformation function $\mathbf{E}_\infty$ also maps to a valuation $\mathcal{V}$ an environment $\theta_\mathcal{V}$ defined as follows:
$\theta_\mathcal{V}(X_i) = \mathsf{true}$ iff $s_i \in \mathcal{V}(X)$.

Proposition 9.5 The property $\sigma X.\Phi$ holds at state s_i of transition system $\mathcal{T}$ in the model $\mathcal{M} = (\mathcal{T}, \mathcal{V})$, $s_i \models_\mathcal{M} \sigma X.\Phi$, iff the corresponding infinite Boolean equation system has the solution true for X_i, i.e. for all environments $\theta_\mathcal{V}$ it is the case that

$([\![\mathbf{E}_\infty((\sigma X{=}\Phi), \mathcal{M})]\!]\ \theta_\mathcal{V})\,(X_i) = \mathsf{true}$.

Proof: The proof is analogous to the one of proposition 5.1. The introduction of new variables and equations is correct due to lemma 3.25. □

Also the backwards transformation from an infinite Boolean equation system to a model checking problem works here. The construction of section 5.2 is immediately applicable to the infinite case.

Theorem 9.6 For each infinite Boolean equation system $\mathcal{E}$ there exists a proposition of the modal μ-calculus Φ and a model $\mathcal{M}$, such that for a variable renaming function λ on the variables of $\mathcal{E}$, all $X \in lhs(\mathcal{E})$ and environments θ we have

$$([\![\mathcal{E}]\!]\ \theta)(X) = ([\![\mathbf{E}(\Phi, \mathcal{M})]\!]\ \theta)(\lambda(X))$$

Proof: See proof of Theorem 5.2 □

9.3 Set based Boolean equation systems.

So far we have introduced infinite Boolean equation systems, showed that various properties of the finite case also hold for the infinite, and that the model checking problem for possibly infinite state spaces and infinite Boolean equation systems are equivalent. However this results only become useful, if we find a finite representation of infinite Boolean equation systems. This is the aim of this section.

The finite representation we give here deals only with infinite Boolean equation systems which contain no proper disjunctions (i.e. if there exist disjunctions, then they consist of not more than one disjunct). Therefore here Theorem 9.4 is crucial. For every model checking problem we get an infinite Boolean equation system, and for every infinite Boolean equation system $\mathcal{E}$ there exists another infinite Boolean equation system $\mathcal{E}'$ without proper disjunctions, but having the same solution as $\mathcal{E}$ and being finitely representable.

The kind of Boolean equation systems which will be introduced here is called "set based". Intuitively in a Boolean equation system derived from a model checking problem there is one variable for each pair consisting of a state and a fixpoint variable. The variable will be true, if the fixpoint formula corresponding to this variable holds at the state. Here this idea generalizes to variables for pairs consisting of a set of states and a fixpoint variable, and the variable will be true, if the corresponding fixpoint formula holds at all states contained in the set. The sets considered here may of course contain infinitely many states and this is the technique where finite representations can be obtained.

Encoded in a set based Boolean equation system will be a transformation to an infinite Boolean equation system. The semantics of a set based Boolean equation system is then defined by the semantics of the infinite Boolean equation system, to which it is transformed. For that purpose we need partial mappings $\rho, \rho_1, \ldots$. Each right-hand side variable in a set based Boolean equation system will be equipped with such a mapping ρ.

For the state space $\mathcal{S}$ and some $M \subseteq \mathcal{S}$ let the function ρ be $\rho : M \to \mathfrak{P}(S)$. Then we also define $\rho(N) = \bigcup_{s \in N} \rho(s)$ for $N \subseteq \mathcal{S}$. The concatenation $\rho_2 \circ \rho_1$ and union $\rho_1 \cup \rho_2$ of $\rho_2 : M_2 \to \mathfrak{P}(S)$ and $\rho_1 : M_1 \to \mathfrak{P}(S)$ are defined in the usual way:

$$\rho_2 \circ \rho_1 : \begin{cases} M_1 & \to & \mathfrak{P}(\mathcal{S}) \\ s_1 & \mapsto & \{s \in \mathcal{S} \mid \exists s_2 \in M_2 . s_2 \in \rho_1(s_1) \text{ and } s \in \rho_2(s_2)\} \end{cases}$$

and

$$\rho_1 \cup \rho_2 : \begin{cases} M_1 \cup M_2 & \to & \mathfrak{P}(\mathcal{S}) \\ s & \mapsto & \rho_1(s) \cup \rho_2(s) \end{cases}$$

Given a function $\rho : M \to \mathfrak{P}(S)$ define

$$\begin{aligned} \rho^0 &\overset{\text{def}}{=} Id, \quad \text{the identity function} \\ \rho^{i+1} &\overset{\text{def}}{=} \rho^i \circ \rho \\ \rho^* &\overset{\text{def}}{=} \bigcup_{i \in \mathbb{N}} \rho^i \end{aligned}$$

An order $<_\rho$ on $M \subseteq \mathcal{S}$ is defined by a function ρ: for $s_1, s_2 \in M$ define $s_1 < s_2$ if $s_1 \in \rho(s_2)$. We will say ρ is wellfounded, if $<_\rho$ is wellfounded.

We now define the syntax of set based Boolean equation systems.

Definition 9.7 A set based Boolean equation is of the form:

$\sigma(X, M) = \bigwedge_{j \in J}(X_j, M_j, \rho_j)$, where

- $\sigma \in \{\mu, \nu\}$,
- $M, M_j \subseteq \mathcal{S}$ for all $j \in J$,
- $(X, M) \in \mathcal{X}$ is a Boolean variable,

- J is a finite index set,
- $(X_j, M_j) \in \mathcal{X} \cup \{\mathsf{true}, \mathsf{false}\}$,
- $\rho_j : M \to M_j$ for all $j \in J$.

A set based Boolean equation system is a finite sequence of set based Boolean equations.

The semantics of a set based Boolean equation system $\mathcal{E}$ is defined via a transformation $\mathbf{T}$ of $\mathcal{E}$ to an infinite Boolean equation system. Informally, a set based equation $\sigma(X, M) = \bigwedge_{j\in J}(X_j, M_j, \rho_j)$ will be mapped to a set of infinite Boolean equations, each of the form $\sigma X_s = f_s$, where $s \in M$ and f_s is a conjunction which will be defined below.
Assume $M = \{s'_1, s'_2, \ldots\}$. Then

$$\begin{array}{rcl} \mathbf{T}(\epsilon) & = & \epsilon \\ \mathbf{T}(\ \ (\sigma(X, M) = \bigwedge\limits_{j\in J}(X_j, M_j, \rho_j))\ \ \mathcal{E}) & = & \end{array}$$

$$(\sigma X_{s'_1} = \bigwedge_{j\in J}\ \bigwedge_{t\in\rho_j(s'_1)} X_{j,t})\ (\sigma X_{s'_2} = \bigwedge_{j\in J}\ \bigwedge_{t\in\rho_j(s'_2)} X_{j,t}) \ldots \mathbf{T}(\mathcal{E})$$

where

- $X_{s'_i} \in \mathcal{X}$,
- $X_{j,t} \in \mathcal{X} \cup \{\mathsf{true}, \mathsf{false}\}$,
- $X_{j,t} = \mathsf{true}$ if $X_j = \mathsf{true}$,
- $X_{j,t} = \mathsf{false}$ if $X_j = \mathsf{false}$.

Definition 9.8 The semantics of a set based Boolean equation system $\mathcal{E}$ is defined relatively to an environment θ and is itself an environment.

$[\![\mathcal{E}]\!]\,\theta = \theta'$, where $\theta'((X, M)) = (\bigwedge_{s\in M} X_s)([\![\mathbf{T}(\mathcal{E})]\!]\,\theta)$

It is easy to see, that if in $\bigwedge_{j\in J}(X_j, M_j, \rho_j)$ for one of the disjuncts $(X_j, M_j) = \mathsf{false}$, then the infinite disjunction also gets false, i.e., $([\![(\sigma(X, M) = \bigwedge_{j\in J}(X_j, M_j, \rho_j))\ \mathcal{E}]\!]\,\theta)((X, M)) = \mathsf{false}$.

9.4 Elimination method.

In this section we present an elimination method for set based Boolean equation systems. Similarly to the finite case it can be interpreted as a bottom-up evaluation version of the tableau method of Bradfield and Stirling [BS91, Bra92]. In their method the success of a tableau requires investigation of so called extended paths. In the elimination method this task is solved by the mappings ρ in a very simple way.
Analogously to the finite case (see section 6.4) we define a substitution step and an elimination step in a set based Boolean equation system, and show that they preserve the solution.
First we show the substitution step. When performing one substitution step in a set based Boolean equation system $\mathcal{E}$ this stands for a possibly infinite number of simultaneous substitution steps in the corresponding infinite Boolean equation system $\mathbf{T}(\mathcal{E})$.

Lemma 9.9 Let

- $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ be set based Boolean equation systems,
- $M, N, N' \subseteq \mathcal{S}$, where $N \subseteq N'$
- assuming that for all $j \in J$ it is $Y \neq X_j$

$$\begin{aligned} f_M &= (Y, N, \rho_Y) \ \wedge \ \bigwedge_{j \in J} (X_j, M_j, \rho_j), \\ f_{N'} &= \bigwedge_{k \in K} (Y_k, N_k, \rho_k), \\ f'_M &= \bigwedge_{k \in K} (Y_k, N_k, \rho_Y \circ \rho_k) \ \wedge \ \bigwedge_{j \in J} (X_j, M_j, \rho_j), \end{aligned}$$

- θ an environment.

Then $[\![\mathcal{E}_1 \ (\sigma_X(X, M) = f_M) \ \mathcal{E}_2 \ (\sigma_Y(Y, N') = f_N) \ \mathcal{E}_3]\!] \, \theta$
$= [\![\mathcal{E}_1 \ (\sigma_X(X, M) = f'_M) \ \mathcal{E}_2 \ (\sigma_Y(Y, N') = f_N) \ \mathcal{E}_3]\!] \, \theta.$

The proof is in the appendix.
Next we show the elimination step. In case of finite Boolean equation systems the right-hand side occurrences of the left-hand side variable may just be substituted by true or false. Here when eliminating a variable additionally the mappings ρ of all other right-hand side variables

of this equation are extended. Intuitively this corresponds to the investigation of extended paths in the tableau method.

Lemma 9.10 Let

- $\mathcal{E}_1$ and $\mathcal{E}_2$ be set based Boolean equation systems,
- $\sigma(X, M) = (X, M, \rho) \;\wedge\; \bigwedge_{i \in I}(X_i, M_i, \rho_i)$ a set based Boolean equation,
- θ an environment, and
- $\theta' \overset{\text{def}}{=} [\![\mathcal{E}_1 \; (\sigma(X, M) = (X, M, \rho) \wedge \bigwedge_{i \in I}(X_i, M_i, \rho_i)) \; \mathcal{E}_2]\!] \, \theta$

If $\sigma = \nu$ then $\theta' = [\![\mathcal{E}_1(\sigma(X, M) = \bigwedge_{i \in I}(X_i, M_i, \rho_i \circ \rho^*))\mathcal{E}_2]\!] \, \theta$.

If $\sigma = \mu$ and ρ is wellfounded then θ' is as in the case for $\sigma = \nu$,

if ρ is not wellfounded then $\theta'((X, M)) = \mathsf{false}$.

A proof can be found in the appendix.
Based on these both lemmata is the algorithm in Figure 9.1. Its task is to show that for an infinite Boolean equation system $\mathcal{E}$ and environment θ it is $([\![\mathcal{E}]\!] \, \theta)(X_s) = \mathsf{true}$. Creating an equation $\sigma(Z, S') = g$ includes a nondeterministic choice: from each disjunction on the right-hand side of an equation $\sigma Z_s = g_s$ in $\mathcal{E}$ one disjunct is selected, whereas all other equations $\sigma Z_s = g_s$ remain unchanged. All these equations are collected in the block $\mathbf{T}(\sigma(Z, S') = g)$. Evaluation *Eval* of conjunctions is here done by the following rules:

$$
\begin{aligned}
(\mathsf{true}, \rho) \wedge \bigwedge_{i \in I}(X_i, M_i, \rho_i) &= \bigwedge_{i \in I}(X_i, M_i, \rho_i) \\
(\mathsf{false}, \rho) \wedge \bigwedge_{i \in I}(X_i, M_i, \rho_i) &= (\mathsf{false}, \rho) \\
\bigwedge \emptyset &= (\mathsf{true}, \rho) \quad \text{for any } \rho
\end{aligned}
$$

The algorithm in pseudo code is as follows:
So far we presented an algorithm for solving set based Boolean equation systems and proved it correct. The question is still, whether it is always possible to find a representation of an infinite Boolean equation system as set based Boolean equation system such that from solving the latter the solution of the first can be derived.

```
Create an equation $\sigma_X(X, M) = f_X$, such that $s \in M$;
$\mathcal{E}' := \sigma_X(X, M) = f_X$;
Apply, if possible, an elimination step;
$f_X := Eval(f_X)$;
while not $f_X = (\mathsf{true}, \rho)$ or $f_X = (\mathsf{false}, \rho)$
   do
      Select $(Y, N, \rho_Y)$ from $f_X$;
      Create an equation $\sigma_Y(Y, N') = f_Y$, where $N \subseteq N'$;
      Insert $\sigma_Y(Y, N') = f_Y$ in $\mathcal{E}'$ according to the transformation;
      Apply all possible elimination steps and substitution steps;
      Evaluate each equation $\sigma_Z(Z, M') = Eval(f_Z)$;
   od
```

Figure 9.1. Elimination algorithm for infinite Boolean equation systems.

Proposition 9.11 For an infinite Boolean equation system $\mathcal{E}$ and environment θ, where $([\![\mathcal{E}]\!]\,\theta)(X_s) = \mathsf{true}$ the algorithm in Figure 9.1 can evaluate a variable (X, M) to true, where $s \in M$.

Proof: The algorithm includes two sorts of nondeterministic choices. The one is the choice of disjuncts in the infinite Boolean equation system $\mathcal{E}$. Theorem 9.4 says that there exists a choice of disjuncts such that the solution is preserved. The other nondeterministic choice consists in the selection of a set of states when creating a new equation. (Note that this choice is comparable to the thin rule in the tableau method.) We have to make sure that there exist choices, such that the resulting set based system contains only a finite number of equations. The simplest choice is collecting all variables of a block, which have the solution true, i.e. $N_Y \stackrel{\text{def}}{=} \{s \in S \mid ([\![\mathcal{E}]\!]\,\theta)(Y_s)\} = \mathsf{true}$. For each block of $\mathcal{E}$ there exists one set of this kind, and therefore there is just a finite number of these sets. When restricting the choice of sets to these N_Y the resulting set based system $\mathcal{E}'$ is finite. Note that it contains all variables of the system $\mathcal{E}$ which have the solution true. Hence there are enough equations in order to apply the substitution and elimination steps, which are correct according to lemmata 9.10 and 9.9. $\square$

9.5 Examples.

We want to demonstrate the elimination method by two examples. The problems are both contained in [Bra92].
For the transition system $\mathcal{T}$ below we want to show the property that every path starting at $\mathbf{s}$ has only finite length. In terms of modal μ-calculus this is: $\mathbf{s} \in \| \mu Z.[-]Z \|^{\mathcal{T}}_{\mathcal{V}}$.

$$\begin{tikzcd}
s \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] & s_{00} \\
& s_{11} \arrow[r] & s_{10} \\
& s_{22} \arrow[r] & s_{21} \arrow[r] & s_{20} \\
& s_{33} \arrow[r] & s_{32} \arrow[r] & s_{31} \arrow[r] & s_{30} \\
& \vdots & & & & \ddots
\end{tikzcd}$$

In a first step we derive the infinite Boolean equation system for the model checking problem above.

$$\begin{array}{lcll}
\mu Z_s & = & \bigwedge_{i \in I\!N} Z_{s_{ii}} & \\
\mu Z_{s_{ij}} & = & Z_{s_{i(j-1)}} & \text{for } i, j \in I\!N \text{ and } 0 < i \leq j \\
\mu Z_{s_{i0}} & = & \mathsf{true} & \text{for } i \in I\!N
\end{array}$$

The next step is to find a representation as set based Boolean equation system. In general this is the part where the knowledge about system and property to prove comes in. On one hand in each disjunction of the infinite system one disjunct has to be selected, which is not necessary in the case here. On the other hand for each block of the infinite system a suitable partition of the state space has to be found.

As abbreviation we introduce the sets and mappings

$$\begin{array}{lcl}
M_1 & \stackrel{\text{def}}{=} & \{(i,i) \mid i \in I\!N \setminus \{0\}\} \\
M_2 & \stackrel{\text{def}}{=} & \{(i,j) \in I\!N \times I\!N \mid 0 < j \leq i\} \\
M_3 & \stackrel{\text{def}}{=} & \{(i,0) \in I\!N \times I\!N\} \\
\rho_0 & : & s \mapsto \{(0,0)\} \\
\rho_1 & : & s \mapsto M_1 \\
\rho_2 & : & \left\{ \begin{array}{lll} M_2 & \rightarrow \mathfrak{P}(M_2) & \\ (i,j) & \mapsto \{(i,j-1)\} & \text{for } j > 1 \\ (i,1) & \mapsto \emptyset & \end{array} \right. \\
\rho_3 & : & \left\{ \begin{array}{lll} M_2 & \rightarrow \mathfrak{P}(M_3) & \\ (i,j) & \mapsto \emptyset & \text{for } j > 1 \\ (i,1) & \mapsto \{(i,0)\} & \end{array} \right. \\
\rho_? & & \text{will denote an arbitrary mapping}
\end{array}$$

The set based Boolean equation system is then:

$$\mu(Z,\{s\}) = (Z, M_1, \rho_1) \wedge (Z, \{(0,0)\}, \rho_0) \tag{9.1}$$

$$\mu(Z, M_2) = (Z, M_2, \rho_2) \wedge (Z, M_3, \rho_3) \tag{9.2}$$

$$\mu(Z, M_3) = (\mathsf{true}, \rho_?) \tag{9.3}$$

The procedure of solving this equation system is now done in detail. We substitute the right-hand side of equation 9.3 into equation 9.2 getting for equation 9.2:

$$\mu(Z, M_2) = (Z, M_2, \rho_2) \wedge (\mathsf{true}, \rho_?) \tag{9.4}$$

Next we apply an elimination step to equation 9.4. Because ρ_2 is wellfounded we get:

$$\mu(Z, M_2) = (\mathsf{true}, \rho_?) \tag{9.5}$$

In the last step we substitute the right-hand sides of equations 9.5 and 9.3 into equation 9.1.

$$\mu(Z,\{s\}) = (\mathsf{true}, \rho_?) \wedge (\mathsf{true}, \rho_?) \tag{9.6}$$

$$= (\mathsf{true}, \rho_?) \tag{9.7}$$

which gives the expected result $Z_s = \mathsf{true}$ or $\mathbf{s} \in \|\,\mu Z.[-]Z\,\|^{\mathcal{T}}_{\mathcal{V}}$. $\lhd$
The second example is originally a Petri Net example in [Bra92]. Here we demonstrate its version based on a transition system. The property to prove is that on all paths a c-transition occurs only finitely often. This will be shown for the initial state s_{00} of the transition system below and the corresponding expression is $s_{00} \in \|\,\mu Y.\nu Z.[c]Y \wedge [-c]Z\,\|^{\mathcal{T}}_{\mathcal{V}}$.

$$\begin{tikzcd}
\mathbf{s_{00}} \arrow[r] \arrow[d, "\mathbf{c}"] & \mathbf{s_{01}} \arrow[r] \arrow[d, "\mathbf{c}"] & \mathbf{s_{02}} \arrow[r] \arrow[d, "\mathbf{c}"] & \mathbf{s_{03}} \arrow[r] \arrow[d, "\mathbf{c}"] & \ldots \\
\mathbf{s_{10}} & \mathbf{s_{11}} \arrow[l] & \mathbf{s_{12}} \arrow[l] & \mathbf{s_{13}} \arrow[l] & \ldots \arrow[l]
\end{tikzcd}$$

We immediately present a set based system, where it is assumed that $i \in \{1,2\}$, $j,k \in I\!N$ and $k > 0$. Define the mappings

$$\begin{aligned}
\rho_1((1,j)) &= \begin{cases} \{(1,j)\} & \text{for } j \geq 1 \\ \emptyset & \text{for } j = 0 \end{cases} \\
\rho_2((1,j)) &= \begin{cases} \emptyset & \text{for } j \geq 1 \\ \{(1,0)\} & \text{for } j = 0 \end{cases} \\
\rho_3((0,j)) &= \{(0,j+1)\} \\
\rho_4((0,j)) &= \{(1,j)\} \\
\rho_5((1,k)) &= \begin{cases} \{(1,k-1)\}\ ) & \text{for } k > 1 \\ \emptyset & \text{for } k = 1 \end{cases} \\
\rho_6((1,1)) &= \{(1,0)\}
\end{aligned}$$

Then a set based Boolean equation system equivalent to the model checking problem is:

$$\mu\ (Y,\ \{(0,j)\}) \;=\; (Z,\ \{(0,j)\},\ id\ ) \tag{9.8}$$
$$\mu\ (Y,\ \{(1,j)\}) \;=\; (Z,\ \{(1,k)\},\ \rho_1\ )\ \wedge\ (Z,\ \{(1,0)\},\ \rho_2\ ) \tag{9.9}$$
$$\nu\ (Z,\ \{(0,j)\}) \;=\; (Z,\ \{(0,j)\},\ \rho_3\ )\ \wedge\ (Y,\ \{(1,j)\},\ \rho_4\ ) \tag{9.10}$$

$$\nu\ (Z,\ \{(1,k)\}) \;=\; (Z,\ \{(1,k)\},\ \rho_5\ ) \wedge (Z,\ \{(1,0)\},\ \rho_6\ ) \quad (9.11)$$
$$\nu\ (Z,\ \{(1,0)\}) \;=\; (\mathsf{true},\ \rho_?\ ) \quad (9.12)$$

After substitution of equation 9.12 into equations 9.11 and 9.9 and elimination steps in equations 9.10 and 9.11 we get:

$$\mu\ (Y,\ \{(0,j)\}) \;=\; (Z,\ \{(0,j)\},\ id\ ) \quad (9.13)$$
$$\mu\ (Y,\ \{(1,j)\}) \;=\; (Z,\ \{(1,k)\},\ \rho_1\ ) \wedge (\mathsf{true},\ \rho_?\ ) \quad (9.14)$$
$$\nu\ (Z,\ \{(0,j)\}) \;=\; (Y,\ \{(1,j)\},\ \rho_4 \circ \rho_3^*\ ) \quad (9.15)$$
$$\nu\ (Z,\ \{(1,k)\}) \;=\; (\mathsf{true},\ \rho_?\ ) \quad (9.16)$$
$$\nu\ (Z,\ \{(1,0)\}) \;=\; (\mathsf{true},\ \rho_?\ ) \quad (9.17)$$

Now we substitute the right-hand side of equation 9.15 in equation 9.13 and also 9.16 in 9.14.

$$\mu\ (Y,\ \{(0,j)\}) \;=\; (Y,\ \{(1,j)\},\ id \circ \rho_4 \circ \rho_3^*\ ) \quad (9.18)$$
$$\mu\ (Y,\ \{(1,j)\}) \;=\; (\mathsf{true},\ \rho_?\ ) \quad (9.19)$$
$$\ldots$$

The last substitution of 9.19 in 9.18 gives the result

$$\mu\ (Y,\ \{(0,j)\}) \;=\; (\mathsf{true},\ \rho_?\ )$$

and it is proved that $s_{00} \in \| \mu Y.\nu Z.[c]Y \wedge [-c]Z \|^{\mathcal{T}}_{\mathcal{V}}$. ◁

9.6 Conclusion.

In this chapter we showed that the techniques for finite Boolean equation systems can be extended in order to deal also with infinite state spaces. The model checking problem for infinite state spaces and solving Boolean equation systems were shown to be equivalent. Whilst the theoretical approach differs very much from the one in [BS91, Bra92], the tableau method there and the elimination method presented here are closely related. The main advantages of the tableau method are the ones of local model checking and computer assisted proving in contrast to fully automatic proving. The first allows to consider only a relevant part of the state space, which is possibly a much smaller subset. The latter gives the possibility to set up a proof following the knowledge about the special structure and properties of a system in contrast to traversing a whole state space and trying to prove every subformula at every state, which makes proving properties impossible for infinite systems. The elimination algorithm combines the top-down strategy of the tableau with a bottom-up evaluation. With this combination we get the advantages of the tableau method, but we are also able to avoid the inherent redundancy of tableaux as well as exploration of extended paths for the success criterion.

Andersen [And94b] described another method for performing model checking on infinite state systems, presented as a set of rewriting rules and also similar to the tableau system of [BS91, Bra92]. It improves the tableau method in the sense that the success criterion for a leaf is derivable from the path leading to that leaf rather than by an exploration of possibly the whole tableau.

Already Wallner [Wal94] transformed model checking for the modal μ-calculus to infinite Boolean equation systems, but did not derive a finite representation.

Chapter 10

Conclusion.

In this thesis we attacked model checking in the modal μ-calculus. The approach was an algebraic one: model checking was transformed to solving Boolean equation systems and both problems have been shown to be equivalent for both, models with infinite and with finite state space.

10.1 Finite state space model checking

Equivalence to solving Boolean equation systems

Other people have reduced the model checking problem for finite state spaces to solving Boolean equation systems. Further we have shown the equivalence of both problems by a reduction which maps a model checking problem to a Boolean equation system. With this result any algorithm solving one of the problems also solves the other one. Boolean equation systems as used here have a simpler structure than the model checking problem: right-hand sides of equations are negation free Boolean expressions, the equations are ordered linearly, and each equation is equipped with a minimality or maximality condition; the logical modalities disappear, and the model is encoded in the equation system. Boolean equation systems are interpreted over complete lattices and results of lattice theory give support in finding new algorithms.

Algorithms and complexity

There exist several algorithms which solve the model checking problem for finite state spaces. However, they all have exponential time complexity. The model-checking problem is known to be in NP $\cap$ co-NP, and it is believed that there exists an algorithm soving the problem in polynomial time. But so far, no polynomial algorithm has been found. Existing model-checking algorithms use various settings, and interpreting all of them within one framework helped to get a clearer understanding. We introduced a new algorithm, similar to Gauß elimination for linear equation systems, in a global and a local version. It took a long time to find an example where this algorithm has exponential behaviour, i.e. the expressions created have exponential size. While looking for it many examples occurred where the tableau method and the approximation technique have exponential time complexity (and also space for the tableau), but Gauß elimination solves them in linear time and space. The difficulty in finding an exponential example might indicate that the average complexity of the problem is much better than exponential. Furthermore, we showed that complexity of Gauß elimination is independent of the alternation depth of a Boolean equation system or a μ-calculus formula (but depends on the structure of the expressions). Approximation algorithms are exponential in the alternation depth. Obviously, is not inherent to the problem that algorithms solving it are exponential in the alternation depth. This gives an argument, that there could be a polynomial algorithm combining ideas of approximation and elimination approach.

Application

Fairness properties are quite difficult to express in the modal μ-calculus. Usually statements are restricted to the fact that the modal μ-calculus allows to express "infinitely often" and this is a necessary ingredient for fairness properties. We gave in section* 7 examples which allow the derivation of a scheme for engineering "real" liveness properties with fairness assumptions. Some fairness and liveness properties can also be expressed in other temporal logics, such as CTL*, but translation from these logics to modal μ-calculus is for all interesting logics exponential or even worse. Therefore it is useful to formulate properties directly in the modal μ-calculus and our examples help with engineering of new formulae.

Other frameworks

Model checking in the modal μ-calculus has already been treated in other frameworks. We looked at them from the perspective of Boolean equation systems and could show equivalences for automata-theoretic and game-theoretic problems.

Automata theory

Mapping μ-calculus formulae to automata already has a long tradition. We were able to show a new result: the equivalence of solving Boolean equation systems and the emptiness problem for alternating automata on infinite strings over a 1-letter alphabet and parity acceptance condition. The equivalence to model checking follows immediately with results from chapter 5. In other work there are various reductions of model checking in the modal μ-calculus to automata-theoretic problems. However, all automata previously considered have been tree-automata. There is a strong claim, that modal μ-calculus expressions correspond to tree-automata, and this idea has been transferred to model checking work. Our result demonstrates that this is not a necessary feature. No new complexity results follow directly from our equivalence, but now also results of alternating ω-automata may help to find a solution.

Game theory

We have shown the equivalence of solving Boolean equation systems and graph games. In doing this we gave an answer to the open question of whether graph games are reducible to model checking problems. Game theory is an active area of research and there exist reductions of graph games to e.g. simple stochastic game, for which there exists a "subexponential" algorithm ($2^{O\sqrt{n}}$) (see [Sti96]). There is some hope that answers to open complexity questions in game theory will give an answer to the complexity of the model checking problem.

10.2 Infinite state space model checking

Equivalence to solving Boolean equation systems

For the case of infinite state spaces we introduced infinite Boolean equation systems and showed the equivalence of model checking in the modal μ-calculus and solving infinite Boolean equation systems. Translating into game-theoretic terms, we also showed the existence of history-free winning strategies for the case of infinite state spaces.

Algorithm

Analogously to Gauß elimination for the finite case we derived an elimination algorithm for infinite Boolean equation systems. Here, the existence of history-free winning strategies was a crucial condition for representing infinite Boolean equation systems by finite, set based Boolean equation systems. The algorithm is closely related to the tableau method of Bradfield and Stirling [BS91, Bra92], but, like in the finite case, avoiding redundancy of tableaux. The bottom-up strategy for solving set based Boolean equation systems gave another advantage: the complicated exploration of extended paths to determine success of a leaf is replaced by iterative function compositions which seems to be easier treatable for an implementation.
Like in the tableau system there is a high grade of nondeterminism

inherent in the elimination algorithm. The idea of making use of knowledge about a system and a property to direct a proof is quite attractive. If the supposed properties about a system and the system do not coincide then the solution of the set based system constructed will be false. This also immediately gives diagnostic information. It would be interesting to try this approach with real world examples.

Appendix A

Appendix

A.1 Proofs of Chapter 3.

Proposition 3.5 The solution of $[\epsilon]\,\theta$ is θ.

The solution of $[(\sigma X = f)\ \mathcal{E}]\,\theta$ is the lexicographically least (w.r.t $(\sigma X = f)\ \mathcal{E}$) environment θ_1 satisfying:

(1) $f(\theta_1) = \theta_1(X)$ and

(2) θ_1 is the solution of $[\mathcal{E}]\,\theta\,[\,X\,/\,\theta_1(X)\,]$.

Proof: Assume that $\sigma = \mu$. The case $\sigma = \nu$ is dually.

$$\begin{array}{llcll}
(3) & \theta_1 & = & [\mathcal{E}]\,\theta\,[X/\theta_1(X)] & \text{from (2)} \\
(4) & \theta_1(X) & = & f([\mathcal{E}]\,\theta\,[\,X\,/\,\theta_1(X)\,]) & \text{from (1)} \\
(5) & \theta_1(X) & \geq & \bigcap\{a \mid a \geq f([\mathcal{E}]\,\theta\,[X/a])\} & \text{from (3) and (4)}
\end{array}$$

on the other hand for

$$\begin{array}{llcll}
(6) & \theta_1' & \overset{\text{def}}{=} & [\mathcal{E}]\,\theta\,[\,X\,/\,\mu X.f([\mathcal{E}]\,\theta)\,] & \\
(7) & f(\theta_1') & = & \theta_1'(X) & \\
(8) & \theta_1'(X) & \geq & \theta_1(X) & \theta_1 \text{ is lex. least env.} \\
 & & & & \text{fulfilling (1) and (2)} \\
(9) & \theta_1(X) & = & \mu X.f([\mathcal{E}]\,\theta)\,] & \text{from (5) and (8)}
\end{array}$$

□

Corollary 3.7 If $[\mathcal{E}]\,\theta\ = \theta'$ then $[\mathcal{E}^{(i)}]\,\theta\,' = \theta'$ for $1 \leq i \leq n$.

Proof: Follows directly from proposition 3.5 □

Proposition 3.9 Let $\sigma X.f$ be a fixpoint expression over a lattice $(A, \leq)$ and θ an arbitrary environment.

Then $(\sigma X.f)(\theta) = ([\mathbf{E}(\sigma X.f)]\,\theta)(X)$.

Proof: For the proof of this proposition we need a property of Boolean equation systems concerning the independence of equations with different variables.

Lemma 3.10 Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be fixpoint-equation systems, such that

- $lhs(\mathcal{E}_1) \cap lhs(\mathcal{E}_2) = \emptyset$,
- $lhs(\mathcal{E}_1) \cap rhs(\mathcal{E}_2) = \emptyset$,
- $lhs(\mathcal{E}_2) \cap rhs(\mathcal{E}_1) = \emptyset$.

Then $[\mathcal{E}_1][\mathcal{E}_2]\theta = [\mathcal{E}_1\mathcal{E}_2]\theta$.

The proof of proposition 3.9 is now by induction on the structure of $\mathcal{E}$.

- Assume $\nu X.f$ is an unnested expression, i.e. $\mathbf{E}(\nu X.f) = \nu X.f$, and θ an environment.

$$\begin{aligned}
([\nu X.f]\,\theta\,)(X) &= ([\epsilon]\,\theta\,[X/\nu X.f([\epsilon]\,\theta)])(X)\\
&= (\theta\,[X/\nu X.f(\theta)])(X)\\
&= (\nu X.f)(\theta)
\end{aligned}$$

  The same holds for unnested $\mu X.f$

- Now assume that $\sigma_1 X_1.f_1, \ldots, \sigma_l X_l.f_l$ are the direct fixpoint subformulae of $\nu X.f$ (and by assumption the names of variables in fixpoint expressions are unique, such that X_1 does not occur in $\sigma_2 X_2.f_2, \ldots, \sigma_l X_l.f_l$ etc.). Furthermore let for $1 \leq i \leq l$ and $S \subseteq A$

$$[\mathbf{E}(\sigma_i X_i.f_i)]\,\theta\,[X/S] \ \overset{\text{def}}{=}\ [\mathcal{E}_i]\,\theta\,[X/S] = (\sigma_i X_i.f_i)(\theta[X/S])$$

$$\begin{aligned}
([\nu X.f]\,\theta\,)(X) &= ([(\nu X.\mathbf{E}'(f))\mathbf{E}(\sigma_1 X_1.f_1)\ldots\mathbf{E}(\sigma_l X_l.f_l)]\,\theta\,)(X)\\
&= ([(\nu X.\mathbf{E}'(f))\mathcal{E}_1\ldots\mathcal{E}_l]\,\theta\,)(X)\\
&= ([\mathcal{E}_1\ldots\mathcal{E}_l]\,\theta\,[X/\nu X.(\mathbf{E}'(f)([\mathcal{E}_1\ldots\mathcal{E}_l]\,\theta))])(X)
\end{aligned}$$

$$
\begin{aligned}
&= \nu X.(\mathbf{E}'(f)([\mathcal{E}_1 \ldots \mathcal{E}_l]\,\theta)) \quad (\text{ lemma } 3.10)\\
&= \nu X.(\mathbf{E}'(f)([\mathcal{E}_1] \ldots [\mathcal{E}_l]\,\theta)\\
&= \nu X.(\mathbf{E}'(f)(\,\theta\,[X_1/\sigma_1 X_1.f_1, \ldots, \sigma_l X_l.f_l]))\\
&= (\nu X.f)(\theta)
\end{aligned}
$$

Again the same holds for nested $\mu X.f$. □

Lemma 3.11 If $\theta_1 \leq \theta_2$ then $[\mathcal{E}]\,\theta_1 \leq [\mathcal{E}]\,\theta_2$.

Proof: by induction.

For all $\theta_1 \leq \theta_2$ it is the case that $[\epsilon]\,\theta_1 = \theta_1 \leq \theta_2 = [\epsilon]\,\theta_2$.

induction hypothesis: Assume that for all $\theta_1 \leq \theta_2$ it is $[\mathcal{E}]\,\theta_1 \leq [\mathcal{E}]\,\theta_2$.

induction step:

$$
\begin{array}{rcll}
[\mathcal{E}]\,\theta_1 & \leq & [\mathcal{E}]\,\theta_2 & \\
\sigma X.f([\mathcal{E}]\,\theta_1) & \leq & \sigma X.f([\mathcal{E}]\,\theta_2) & (f \text{ and } \sigma X. \text{ are monotone})\\
\theta_1[X/\sigma X.f([\mathcal{E}]\,\theta_1)] & \leq & \theta_2[X/\sigma X.f([\mathcal{E}]\,\theta_1)] & (\text{ definition of } \leq)\\
[\mathcal{E}]\,\theta_1[X/\sigma X.f([\mathcal{E}]\,\theta_1)] & \leq & [\mathcal{E}]\,\theta_2[X/\sigma X.f([\mathcal{E}]\,\theta_1)] & (\text{ind. hyp.})\\
[(\sigma X = f)\ \mathcal{E}]\,\theta_1 & \leq & [(\sigma X = f)\ \mathcal{E}]\,\theta_2 & (\text{definition of semantics})
\end{array}
$$
□

Lemma 3.14 If $\mathcal{E}_1 \sim \mathcal{E}_2$ then $\mathcal{E}\mathcal{E}_1 \sim \mathcal{E}\mathcal{E}_2$.

If $\mathcal{E}_1 \precsim \mathcal{E}_2$ then $\mathcal{E}\mathcal{E}_1 \precsim \mathcal{E}\mathcal{E}_2$.

Proof: For the second part we show $[(\sigma X = f)\ \mathcal{E}_1]\,\theta \leq [(\sigma X = f)\ \mathcal{E}_2]\,\theta$. The lemma as stated follows then from iterative application of the weaker statement.

$$
\begin{aligned}
[(\sigma X = f)\ \mathcal{E}_1]\,\theta &= [\mathcal{E}_1]\,\theta\,[X/\sigma X.f([\mathcal{E}_1]\,\theta)]\\
&\leq [\mathcal{E}_1]\,\theta\,[X/\sigma X.f([\mathcal{E}_2]\,\theta)]\\
&\leq [\mathcal{E}_2]\,\theta\,[X/\sigma X.f([\mathcal{E}_2]\,\theta)]\\
&= [(\sigma X = f)\ \mathcal{E}_2]\,\theta\,.
\end{aligned}
$$

The first part follows the immediately. □

Lemma 3.16 If $\mathcal{E}_1 \leq \mathcal{E}_2$ then also $\mathcal{E}_1 \precsim \mathcal{E}_2$

Proof: by structural induction
Assume $f \leq g$. Then

$$\begin{array}{rcl}
[\sigma X = f]\,\theta & = & \theta[X/(\sigma X.f)(\theta)] \\
& \leq & \theta[X/(\sigma X.g)(\theta)] \\
& = & [\sigma X = g]\,\theta
\end{array}$$

Assume $f \leq g$ and $\mathcal{E}_1 \leq \mathcal{E}_2$ with $[\mathcal{E}_1]\,\theta \leq [\mathcal{E}_2]\,\theta$.

$$\begin{array}{rcl}
[(\sigma X = f)\ \mathcal{E}_1]\,\theta & = & [\mathcal{E}_1]\,\theta\,[X/\sigma X.f([\mathcal{E}_1]\,\theta)] \\
& \leq & [\mathcal{E}_1]\,\theta\,[X/\sigma X.g([\mathcal{E}_1]\,\theta)] \\
& \leq & [\mathcal{E}_1]\,\theta\,[X/\sigma X.g([\mathcal{E}_2]\,\theta)] \\
& \leq & [\mathcal{E}_2]\,\theta\,[X/\sigma X.g([\mathcal{E}_2]\,\theta)] \\
& = & [(\sigma X = g)\ \mathcal{E}_2]\,\theta
\end{array}$$ □

Lemma 3.18 If $([(\sigma X = f)\ \mathcal{E}]\,\theta)(X) = ([(\sigma X = g)\ \mathcal{E}]\,\theta)(X)$
then $[(\sigma X = f)\ \mathcal{E}]\,\theta = [(\sigma X = g)\ \mathcal{E}]\,\theta.$

Proof: Follows directly from proposition 3.5. □

Lemma 3.19 Let

- $\mathcal{E} \equiv \mathcal{E}_1\ (\sigma X = f)\ \mathcal{E}_2$,
- $([\mathcal{E}]\,\theta)(X) = a$, and
- $\mathcal{E}' \equiv \mathcal{E}_1\ (\sigma X = a)\ \mathcal{E}_2$.

Then $[\mathcal{E}]\,\theta = [\mathcal{E}']\,\theta$.

Proof: Note that here we can not simply apply proposition 3.5 or lemma 3.14, because the equivalence $[(\sigma X = f)\ \mathcal{E}_2]\,\theta = [(\sigma X = a)\ \mathcal{E}_2]\,\theta$ does not hold for all environments θ.
The proof is done by contradiction. We assume that for $[\mathcal{E}']\,\theta \overset{\text{def}}{=} \theta'_1$ it is $\theta'_1 \neq \theta_1$ and derive an infinite number of subsystems of $\mathcal{E}$ and $\mathcal{E}'$, which must have different solutions.

θ_1 and θ'_1 coincide in all variables which are not bound in $\mathcal{E}$, or $\mathcal{E}'$ respectively. Let n be the number of equations of $\mathcal{E}$. For all i, $1 \leq i \leq n$ holds, $[\mathcal{E}^{(i)}]\,\theta_1 = \theta_1$.

Now choose the first variable Y of $var(\mathcal{E})$ (first with respect to the order of equations in $\mathcal{E}$), for which holds $\theta_1(Y) \neq \theta'_1(Y)$, such that for all previous variables θ_1 and θ'_1 coincide. Fix the i such that $\mathcal{E}^{(i)} \equiv (\sigma_i Y = g)\mathcal{E}^{(i+1)}$, and $\mathcal{E}'^{(i)} \equiv (\sigma_i Y = g)\mathcal{E}'^{(i+1)}$.

$$\begin{aligned}\theta_1 &= [\mathcal{E}^{(i)}]\,\theta_1\\ &= [(\sigma_i Y = g)\,\mathcal{E}^{(i+1)}]\,\theta_1\\ &= [\mathcal{E}^{(i+1)}]\,\theta_1[Y/\sigma_i Y.g([\mathcal{E}^{(i+1)}]\,\theta_1)]\end{aligned}$$

$$\begin{aligned}\theta'_1 &= [\mathcal{E}'^{(i)}]\,\theta_1\\ &= [(\sigma_i Y = g)\,\mathcal{E}'^{(i+1)}]\,\theta_1\\ &= [\mathcal{E}'^{(i+1)}]\,\theta_1[Y/\sigma_i Y.g([\mathcal{E}'^{(i+1)}]\,\theta_1)]\end{aligned}$$

Hence, because $\theta_1(Y) \neq \theta'_1(Y)$ also

$\sigma_i Y.g([\mathcal{E}^{(i+1)}]\,\theta_1)] \neq \sigma_i Y.g([\mathcal{E}'^{(i+1)}]\,\theta_1)]$,

and therefore $[\mathcal{E}^{(i+1)}]\,\theta_1 \neq [\mathcal{E}'^{(i+1)}]\,\theta_1 \overset{\text{def}}{=} \theta''_1$

On the other hand still $\theta_1(X) = a$ and also $\theta''_1(X) = a$. Therefore we can apply the same argumentation to $\mathcal{E}^{(i+1)}$, θ_1, $\mathcal{E}'^{(i+1)}$ and θ''_1, and so on. Altogether we can derive that there must be an infinite number of subsystems $\mathcal{E}^{(i)}$ and $\mathcal{E}'^{(i)}$ having different solutions relative to θ_1. $\square$

Lemma 3.20 $[\mathcal{E}_1\ (\sigma X = a)\ \mathcal{E}_2]\,\theta = [\mathcal{E}_1\ \mathcal{E}_2]\,\theta\,[X/a]$.

Proof: For all environments θ we have $[(\sigma X = a)\mathcal{E}_2]\,\theta = [\mathcal{E}_2]\,\theta\,[X/a]$. For some $\mathcal{E}, \mathcal{E}'$ and all environments θ let $[\mathcal{E}]\,\theta = [\mathcal{E}']\,\theta\,[X/a]$. Then

$$\begin{aligned}[(\sigma Y = f)\ \mathcal{E}]\,\theta &= [\mathcal{E}]\,\theta\,[Y/\sigma Y.f([\mathcal{E}]\,\theta)]\\ &= [\mathcal{E}']\,\theta\,[X/a][Y/\sigma Y.f([\mathcal{E}']\,\theta\,[X/a])]\\ &= [(\sigma Y = f)\ \mathcal{E}']\,\theta\,[X/a].\end{aligned}$$ $\square$

Lemma 3.21 Let

- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1\ (\sigma X_1 = f_1)\ (\sigma X_2 = f_2)\ \mathcal{E}_2]\,\theta$, and
- $\theta_2 \overset{\text{def}}{=} [\mathcal{E}_1\ (\sigma X_2 = f_2)\ (\sigma X_1 = f_1)\ \mathcal{E}_2]\,\theta$.

Then $\theta_1 = \theta_2$.

Proof: follows from Bekič's theorem and the transformation from nested fixpoints to fixpoint-equation systems in proposition 3.9. $\square$

Lemma 3.22 If

- X_1 is not free in f_2,
- X_2 is not free in f_1,
- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1\ (\sigma_1 X_1 = f_1)\ (\sigma_2 X_2 = f_2)\ \mathcal{E}_2]\,\theta$
- $\theta_2 \overset{\text{def}}{=} [\mathcal{E}_1\ (\sigma_2 X_2 = f_2)\ (\sigma_1 X_1 = f_1)\ \mathcal{E}_2]\,\theta$

Then $\theta_1 = \theta_2$.

Proof: Straightforward application of the definition of the semantics shows that
$[(\sigma_1 X_1 = f_1)\ (\sigma_2 X_2 = f_2)\ \mathcal{E}_2]\theta = (\sigma_2 X_2 = f_2)\ (\sigma_1 X_1 = f_1)\ \mathcal{E}_2]\theta$
for all environments θ. Then lemma 3.14 can be applied. $\square$

Lemma 3.23 Let

- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1\ (\mu X_1 = f_1)\ (\nu X_2 = f_2)\ \mathcal{E}_2]\,\theta$, and
- $\theta_2 \overset{\text{def}}{=} [\mathcal{E}_1\ (\nu X_2 = f_2)\ (\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta$.

Then it is $\theta_1 \leq \theta_2$, and moreover, if the inequality is strict then $\theta_1(X_1) < \theta_2(X_1)$ and $\theta_1(X_2) < \theta_2(X_2)$.

Proof: According to lemma 3.14 it suffices for the first part of the proposition to show that for all environments θ it is
$[(\mu X_1 = f_1)\ (\nu X_2 = f_2)\ \mathcal{E}_2]\,\theta \leq [(\nu X_2 = f_2)\ (\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta$.
Let $[(\mu X_1 = f_1)\ (\nu X_2 = f_2)\ \mathcal{E}_2]\,\theta \overset{\text{def}}{=} \theta_1'$. We know that

- $f_2(\theta_1') = \theta_1'(X_2)$ (proposition 3.5),
- $[(\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta_1' = \theta_1'$ (lemmata 3.19, 3.20)

Due to proposition 3.5 these are the two properties which the solution θ_2' of $[(\nu X_2 = f_2)\ (\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta$ must have, and furthermore the

solution θ_2' is the lexicographic least one of those environments $\theta\,'$ having these properties. Hence the solution θ_2' is lexicographically lower or equal to θ_1, i.e. $\theta_2'(X_2) \geq \theta_1'(X_2)$.
If $\theta_2'(X_2) = \theta_1'(X_2)$ then applying lemmata 3.19, 3.20 shows that both solutions must be equal.

$$\begin{aligned}
\theta_1' &= [(\mu X_1 = f_1)\ (\nu X_2 = f_2)\ \mathcal{E}_2]\,\theta \\
&= [(\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta\,[X_2/\,\theta_1'(X_2)] \\
&= [(\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta\,[X_2/\,\theta_2'(X_2)] \\
&= [(\nu X_2 = f_2)\ (\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta\,.
\end{aligned}$$

If $\theta_2'(X_2) > \theta_1'(X_2)$ then $\theta_2' \geq \theta_1'$ and $\theta\,[X_2/\,\theta_2'(X_2)] > \theta\,[X_2/\theta_1'(X_2)]$ and with lemma 3.11 also

$$\begin{aligned}
\theta_2' &= [(\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta\,[X_2/\,\theta_2'(X_2)] \\
&\leq [(\mu X_1 = f_1)\ \mathcal{E}_2]\,\theta\,[X_2/\,\theta_1'(X_2)] \\
&= \theta_1'.
\end{aligned}$$

□

Lemma 3.24 Let

- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1\ (\mu X = f)\ \mathcal{E}_2]\,\theta$, and
- $\theta_2 \overset{\text{def}}{=} [\mathcal{E}_1\ (\nu X = f)\ \mathcal{E}_2]\,\theta$.

Then it is $\theta_1 \leq \theta_2$, and moreover, if the inequality is strict then $\theta_1(X) < \theta_2(X)$.

Proof: In order to prove the first part of the lemma and according to lemma 3.14 it suffices to show that $[(\mu X = f)\ \mathcal{E}_2]\,\theta \leq [(\nu X = f)\ \mathcal{E}_2]\,\theta$.

$$\begin{aligned}
[(\mu X = f)\ \mathcal{E}_2]\,\theta &= [\mathcal{E}_2]\,\theta\,[X/\mu X_0.f([\mathcal{E}_2]\,\theta\,[X/X_0])] \\
&\leq [\mathcal{E}_2]\,\theta\,[X/\nu X_0.f([\mathcal{E}_2]\,\theta\,[X/X_0])] \\
&= [(\nu X = f)\ \mathcal{E}_2]\,\theta\,.
\end{aligned}$$

For the second part of the lemma assume that the solutions coincide at X and show that then they must be identical. Substitute the solution $\theta_1(X) = \theta_2(X) = a$ in the equation systems due to lemma 3.19 and eliminate it with lemma 3.20:

$$\begin{aligned}
[\mathcal{E}_1\ (\mu X = f)\ \mathcal{E}_2]\,\theta &= [\mathcal{E}_1\ (\mu X = a)\ \mathcal{E}_2]\,\theta \\
&= [\mathcal{E}_1\ \mathcal{E}_2]\,\theta\,[X/a] \\
&= [\mathcal{E}_1\ (\nu X = a)\ \mathcal{E}_2]\,\theta \\
&= [\mathcal{E}_1\ (\nu X = f)\ \mathcal{E}_2]\,\theta. \qquad \square
\end{aligned}$$

Lemma 3.25

$$([(\sigma X = f_1 \wedge f_2)\ \mathcal{E}]\,\theta)(Y) = ([(\sigma X = f_1 \wedge X')\ (\sigma' X' = f_2)\ \mathcal{E}]\,\theta)(Y),$$

$$([(\sigma X = f_1 \vee f_2)\ \mathcal{E}]\,\theta)(Y) = ([(\sigma X = f_1 \vee X')\ (\sigma' X' = f_2)\ \mathcal{E}]\,\theta)(Y),$$

where X' is a new variable, i.e. (*) X' does not occur on the right hand side of $\mathcal{E}$ or in f_1 or f_2, and (**) $Y \neq X'$.

Proof: by straightforward application of the definition of the semantics.

$$\begin{aligned}
&([(\sigma X = f_1 \wedge X')\ (\sigma' X' = f_2)\ \mathcal{E}]\,\theta)(Y) \\
= \quad & ([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta\,[X/\sigma X.(f_1 \wedge X')([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta)])(Y) \\
= \quad & ([(\sigma' X' = f_2)\ \mathcal{E}] \\
& \qquad \theta\,[X/\sigma X.(f_1([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta) \wedge X'([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta))])(Y) \\
= \quad & ([(\sigma' X' = f_2)\ \mathcal{E}] \\
& \qquad \theta\,[X/\sigma X.(f_1([\mathcal{E}]\,\theta\,[X'/\_]) \wedge X'([\mathcal{E}]\,\theta[X'/\sigma' X'.f_2([\mathcal{E}]\,\theta)]))])(Y) \\
= \quad & ([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta\,[X/\sigma X.(f_1([\mathcal{E}]\,\theta) \wedge \sigma' X'.f_2([\mathcal{E}]\,\theta))])(Y) \\
= \quad & ([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta\,[X/\sigma X.(f_1([\mathcal{E}]\,\theta) \wedge f_2([\mathcal{E}]\,\theta))])(Y) && (*) \\
= \quad & ([(\sigma' X' = f_2)\ \mathcal{E}]\,\theta\,[X/\sigma X.(f_1 \wedge f_2)([\mathcal{E}]\,\theta)])(Y) && (*) \\
= \quad & ([\mathcal{E}]\,\theta\,[X/\sigma X.(f_1 \wedge f_2)([\mathcal{E}]\,\theta)][X'/\ldots])(Y) \\
= \quad & ([\mathcal{E}]\,\theta\,[X/\sigma X.(f_1 \wedge f_2)([\mathcal{E}]\,\theta)](Y) && (*)(**) \\
= \quad & ([(\sigma X = f_1 \wedge f_2)\ \mathcal{E}]\,\theta)(Y)
\end{aligned}$$

The proof for $\vee$ is analogous. $\square$

Lemma 3.26 Let

- $\theta_1 \overset{\text{def}}{=} [\mathcal{E}_1\ (\sigma X_1 = f)\ (\sigma X_2 = f)\ \mathcal{E}_2]\,\theta$
- $\theta_2' \overset{\text{def}}{=} [\mathcal{E}_1[X_1/X_2]\ (\sigma X_2 = f[X_1/X_2])\ \mathcal{E}_2[X_1/X_2]]\,\theta$
- $\theta_2 \overset{\text{def}}{=} \theta_2'[X_1/\theta_2'(X_2)]$

Then $\theta_1 = \theta_2$.

Proof: We will show the lemma for the case of $\sigma = \mu$. The other case of $\sigma = \nu$ is dual. Moreover the proof is done for $\mathcal{E}_1 \equiv \epsilon$. The generalization to arbitrary $\mathcal{E}_1$ follows then by lemma 3.14 and 3.19. For the proof here the alternative characterization of the solution of a fixpoint-equation system in proposition 3.5 turned out to be more suitable.
Show that $\theta_2 \leq \theta_1$:

(1)	$\theta_1(X_1)$	$=$	$\theta_1(X_2)$	proposition 3.5
(2)	$\theta_1(X_2)$	$=$	$f(\theta_1)$	proposition 3.5
(3)	$[\mathcal{E}_2[X_1/X_2]]\,\theta_1$	$=$	$[\mathcal{E}_2]\,\theta_1$	(1),(2), proposition 3.5
(4)	$[\mathcal{E}_2]\,\theta_1$	$=$	θ_1	corollary 3.7

Hence, with proposition 3.5, it is $\theta_2 \leq \theta_1$.
Show that $\theta_1 \leq \theta_2$:

(1)	$[\mathcal{E}_2[X_1/X_2]]\,\theta_2$	$=$	θ_2	proposition 3.5
(2)	$\theta_2(X_1)$	$=$	$\theta_2(X_2)$	by definition
(3)	$[\mathcal{E}_2]\,\theta_2$	$=$	θ_2	(1),(2), lemma 3.19
(4)	$f(\theta_2)$	$=$	$\theta_2(X_2)$	proposition 3.5
(5)	$[(\mu X_2 = f)\ \mathcal{E}_2]\,\theta_2$	$\leq$	θ_2	(3), (4), proposition 3.5
(6)	$f([[(\mu X_2 = f)\ \mathcal{E}_2]\,\theta_2)$	$\leq$	$f(\theta_2)$	(5), monotonicity of f
(7)	$\mu X_1.f([[(\mu X_2 = f)\ \mathcal{E}_2]\,\theta_2)$	$\leq$	$\theta_2(X_1)$	(2), (4), (6), Theo. 2.16
(8)	$\theta_1(X_1)$	$\leq$	$\theta_2(X_1)$	(7)
(9)	$[(\mu X_2 = f)\ \mathcal{E}_2]\,\theta_1$	$\leq$	$[(\mu X_2 = f)\ \mathcal{E}_2]\,\theta_2$	(8) & proposition 3.11
(10)	$\theta_1 = [(\mu X_2 = f)\ \mathcal{E}_2]\,\theta_1$	$\leq$	θ_2	(9), (5), prop 3.5 □

Proposition 3.30 Let $\mathcal{E}$ be a Boolean equation system, $\sigma X = f$ a Boolean equation, θ an environment, $b_\mu = \mathsf{false}$ and $b_\nu = \mathsf{true}$. Then for the solution of a Boolean equation system holds:

$$[\![\epsilon]\!]\,\theta = \theta$$

$$[\![(\sigma X{=}f)\ \mathcal{E}]\!]\,\theta = [\![\mathcal{E}]\!]\,\theta[X\,/\,f(\,[\![\mathcal{E}]\!]\,\theta[X/b_\sigma]\,)\,].$$

Proof: Apply lemma 3.29 to definition 3.3. □

Proposition 3.31 For each Boolean equation system $\mathcal{E}$ there exists a Boolean equation system $\mathcal{E}'$ in standard form and a renaming function λ, such that $(\llbracket\mathcal{E}\rrbracket\,\theta)(X) = (\llbracket\mathcal{E}'\rrbracket\,\theta)(\lambda(X))$, and $\mathcal{E}'$ has size linear in the size of $\mathcal{E}$.

Proof: The transformation from a Boolean equation system $\mathcal{E}$ into standard form is performed by introduction of additional variables (proposition 3.25). The number of additional variables is linear in the size of the right-hand side expressions of $\mathcal{E}$. The size of the right-hand side expressions of $\mathcal{E}'$ is linear in the size of the right-hand side expressions of $\mathcal{E}$. Renaming does not influence the size. □

Lemma 3.35 $(\llbracket\mathcal{E}\rrbracket\,\theta)(X) = \mathsf{false}$ iff $(\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta})(X) = \mathsf{true}$.

Proof: by induction on the structure of $\mathcal{E}$

$$\begin{aligned}
\overline{(\llbracket\epsilon\rrbracket\,\theta)(X)} &= \overline{\theta(X)}\\
&= \overline{\theta}(X)\\
&= (\llbracket\epsilon\rrbracket\,\overline{\theta})(X)\\
&= (\llbracket\overline{\epsilon}\rrbracket\,\overline{\theta})(X)
\end{aligned}$$

induction hypothesis: $\overline{(\llbracket\mathcal{E}\rrbracket\,\theta)(X)} = (\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta})(X)$
Show $\overline{(\llbracket(\mu Y = f)\ \mathcal{E}\rrbracket\,\theta)(X)} = (\llbracket(\nu Y = \overline{f})\ \overline{\mathcal{E}}\rrbracket\,\overline{\theta})(X)$

$$\overline{(\llbracket(\mu Y = f)\ \mathcal{E}\rrbracket\,\theta)(X)}$$

$$\begin{aligned}
&= \overline{(\llbracket\mathcal{E}\rrbracket\,\theta[Y/f(\llbracket\mathcal{E}\rrbracket\,\theta[Y/\mathsf{false}])])(X)} && \text{definition of semantics}\\
&= (\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta[Y/f(\llbracket\mathcal{E}\rrbracket\,\theta[Y/\mathsf{false}])]})(X) && \text{induction hypothesis}\\
&= (\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta}[Y/\overline{f(\llbracket\mathcal{E}\rrbracket\,\theta[Y/\mathsf{false}])}])(X) && \text{complementation of } \theta\\
&= (\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta}[Y/\overline{f}(\overline{\llbracket\mathcal{E}\rrbracket\,\theta[Y/\mathsf{false}]})])(X) && \text{de Morgan}\\
&= \llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta}[Y/\overline{f}(\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta[Y/\mathsf{false}]})])(X) && \text{induction hypothesis}\\
&= \llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta}[Y/\overline{f}(\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta}[Y/\mathsf{true}])(X) && \text{complementation of } \theta\\
&= (\llbracket(\nu Y = \overline{f})\ \overline{\mathcal{E}}\rrbracket\,\overline{\theta})(X) && \text{definition of semantics}
\end{aligned}$$
□

Proposition 3.36 Given a Boolean equation system $\mathcal{E}$ and an environment θ there exist Boolean equation systems $\mathcal{E}'$ and $\mathcal{E}''$ with the properties:

- $\mathcal{E}'$ is in conjunctive form,
- $\mathcal{E}' \leq \mathcal{E}$, and
- $[\![\mathcal{E}']\!]\,\theta = [\![\mathcal{E}]\!]\,\theta$.

For $\mathcal{E}''$ the dual properties hold:

- $\mathcal{E}''$ is in disjunctive form,
- $\mathcal{E}'' \geq \mathcal{E}$, and
- $[\![\mathcal{E}'']\!]\,\theta = [\![\mathcal{E}]\!]\,\theta$.

For the proof of this proposition we need lemmata A.1 and A.2.

Lemma A.1 Given Boolean equation systems $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ with the properties:

(1) $\mathcal{E}_1, \mathcal{E}_2$ are in conjunctive form,

(2) $\mathcal{E}_1 \leq \mathcal{E}$, $\mathcal{E}_2 \leq \mathcal{E}$,

(3) $[\![\mathcal{E}_1]\!]\,\theta\,[X/\mathsf{false}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{false}]$,

(4) $[\![\mathcal{E}_2]\!]\,\theta\,[X/\mathsf{true}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{true}]$.

Then there exists a Boolean equation system $\mathcal{E}_3$ in conjunctive form, such that $\mathcal{E}_3 \leq \mathcal{E}$ and

- $[\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{false}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{false}]$,
- $[\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{true}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{true}]$.

Proof: Assume
$\mathcal{E}_1 = (\sigma_1 X_1{=}f_1)\ldots(\sigma_n X_n{=}f_n)$ and
$\mathcal{E}_2 = (\sigma_1 X_1{=}g_1)\ldots(\sigma_n X_n{=}g_n)$.
Let $\sigma_i X_i{=}f_i$ be an equation of $\mathcal{E}_3$, if $([\![\mathcal{E}_1]\!]\,\theta\,[X/\mathsf{false}])(X_i) = \mathsf{true}$ and $\sigma_i X_i{=}g_i$, if $([\![\mathcal{E}_1]\!]\,\theta\,[X/\mathsf{false}])(X_i) = \mathsf{false}$.
By construction of $\mathcal{E}_3$ follows

(5) $[\![\mathcal{E}_1]\!]\,\theta\,[X/\mathsf{false}] \leq [\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{false}]$,

(6) $\mathcal{E}_3 \leq \mathcal{E}$, and

(7) $\mathcal{E}_3$ is in conjunctive form.

From (3), (5), (6) and proposition 3.16 follows that
$[\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{false}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{false}]$.
We also know that $[\![\mathcal{E}_2]\!]\,\theta\,[X/\mathsf{true}] \leq [\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{true}]$, because at the variables where $\mathcal{E}_2$ and $\mathcal{E}_3$ differ $\mathcal{E}_3$ has the solution true for $\theta\,[X/\mathsf{false}]$ and hence also for $\theta\,[X/\mathsf{true}]$.
With (4) and (6) follows that $[\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{true}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{true}]$. □

Lemma A.2 Given Boolean equation systems $\mathcal{E}, \mathcal{E}_1, \mathcal{E}_2$ with the properties:

(1) $\mathcal{E}_1, \mathcal{E}_2$ are in disjunctive form,

(2) $\mathcal{E}_1 \geq \mathcal{E}$, $\mathcal{E}_2 \geq \mathcal{E}$,

(3) $[\![\mathcal{E}_1]\!]\,\theta\,[X/\mathsf{false}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{false}]$,

(4) $[\![\mathcal{E}_2]\!]\,\theta\,[X/\mathsf{true}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{true}]$.

Then there exists a Boolean equation system $\mathcal{E}_3$ in disjunctive form, such that $\mathcal{E}_3 \leq \mathcal{E}$ and

- $[\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{false}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{false}]$,
- $[\![\mathcal{E}_3]\!]\,\theta\,[X/\mathsf{true}] = [\![\mathcal{E}]\!]\,\theta\,[X/\mathsf{true}]$.

Proof analogous to the proof of lemma A.1 □
Proof of proposition 3.36: by induction
Here we assume that the Boolean equation system is in normal form, i.e. each right hand side expression is either a conjunction or a disjunction of two variables. Then we have to investigate the equations which have a disjunction as right hand side and show that we can select one of the disjuncts preserving the solution.

$$\begin{aligned}
[\![\sigma X{=}(X_i \vee X_j)]\!]\,\theta &= \theta\,[X/(X_i \vee X_j)(\theta\,[X/b_\sigma])] \\
&= \theta\,[X/\theta\,[X/b_\sigma](X_i) \vee \theta\,[X/b_\sigma](X_j)] \\
&= \theta\,[X/\theta\,[X/b_\sigma](X_i)] \\
&\quad \text{(if } (X_i \vee X_j)(\theta') = \mathsf{true} \\
&\quad \text{then assume wlog } X_i(\theta') = \mathsf{true}) \\
&= [\![\sigma X{=}X_i]\!]\,\theta.
\end{aligned}$$

Now assume that for $\mathcal{E}, \theta$ there exists $\mathcal{E}_1$ such that $[\![\mathcal{E}]\!]\,\theta = [\![\mathcal{E}_1]\!]\,\theta$. Let

$$\begin{aligned}
\theta_1 &\overset{\text{def}}{=} \theta\,[X/b_\sigma] \\
\theta_2 &\overset{\text{def}}{=} \theta\,[X/(X_i \vee X_j)\ ([\![\mathcal{E}]\!]\,\theta\,[X/b_\sigma])] \\
\theta_3 &\overset{\text{def}}{=} \theta\,[X/\overline{b_\sigma}], \qquad \text{where } \overline{\mathsf{true}} = \mathsf{false} \text{ and } \overline{\mathsf{false}} = \mathsf{true}
\end{aligned}$$

$$\begin{aligned}
[\![(\sigma X{=}X_i \vee X_j)\ \mathcal{E}]\!]\,\theta &= [\![\mathcal{E}]\!]\ \theta\,[X/(X_i \vee X_j)([\![\mathcal{E}]\!]\,\theta\,[X/b_\sigma])] \\
&= [\![\mathcal{E}]\!]\ \theta\,[X/(X_i \vee X_j)([\![\mathcal{E}]\!]\,\theta_1)] \\
&= [\![\mathcal{E}]\!]\,\theta_2 \\
&= (*)
\end{aligned}$$

We have to consider two cases:

(i) $(X_i \vee X_j)([\![\mathcal{E}]\!]\,\theta_1) = b_\sigma$, and hence $\theta_1 = \theta_2$. Then there exists $\mathcal{E}_1$ such that $[\![\mathcal{E}]\!]\,\theta_i = [\![\mathcal{E}_1]\!]\,\theta_i$ for $i = 1, 2$.

$$\begin{aligned}
(*) &= [\![\mathcal{E}_1]\!]\,\theta\,[X/(X_i \vee X_j)([\![\mathcal{E}_1]\!]\,\theta\,[X/b_\sigma])] \\
&= [\![\mathcal{E}_1]\!]\,\theta\,[X/(X_i)([\![\mathcal{E}_1]\!]\,\theta\,[X/b_\sigma])] \\
&\quad \text{(as in the base case:} \\
&\quad \text{choose a disjunct which gives the correct result)} \\
&= [\![(\sigma X{=}X_i)\mathcal{E}_1]\!]\,\theta
\end{aligned}$$

(ii) $(X_i \vee X_j)([\![\mathcal{E}]\!]\,\theta_1) \neq b_\sigma$, and hence $\theta_2 = \theta_3$. Now there exists a different equation system for either θ_i, $\mathcal{E}_1$ with $[\![\mathcal{E}]\!]\,\theta_1 = [\![\mathcal{E}_1]\!]\,\theta_1$ and $\mathcal{E}_3$ with $[\![\mathcal{E}]\!]\,\theta_3 = [\![\mathcal{E}_3]\!]\,\theta_3$.

Then due to proposition A.1 there exists $\mathcal{E}_4$ with $[\![\mathcal{E}_4]\!]\,\theta_1 = [\![\mathcal{E}]\!]\,\theta_1$ and $[\![\mathcal{E}_4]\!]\,\theta_3 = [\![\mathcal{E}]\!]\,\theta_3$. Hence

$$\begin{aligned}
(*) &= [\![\mathcal{E}_4]\!]\,\theta\,[X/(X_i \vee X_j)([\![\mathcal{E}_4]\!]\,\theta\,[X/b_\sigma])] \\
&= [\![\mathcal{E}_4]\!]\,\theta\,[X/(X_i)([\![\mathcal{E}_4]\!]\,\theta\,[X/b_\sigma])] \\
&\quad \text{(again choose a suitable disjunct)} \\
&= [\![(\sigma X{=}X_i)\mathcal{E}_4]\!]\,\theta.
\end{aligned}$$

The proof for the dual fact, that there exists a conjunctive system which has the same solution as $\mathcal{E}$ works analogously. □

A.2 Proofs of Chapter 5.

Theorem 5.1 Let $\sigma X.\Phi$ be a formula of the modal μ-calculus, $\mathcal{M} = (\mathcal{T}, \mathcal{V})$ a model and s_i a state of $\mathcal{T}$.

Then for all environments $\theta_{\mathcal{V}}$ it is the case that

$s_i \models_{\mathcal{M}} \sigma X.\Phi$ iff $(\llbracket\, \mathbf{E}(\,(\sigma X = \Phi), \mathcal{M}\,)\,\rrbracket\; \theta_{\mathcal{V}})\,(X_i) = \mathsf{true}$.

Proof of theorem 5.1: The mapping $\mathbf{E}$ is divided in three steps: the first leads from a μ-calculus formula to a μ-calculus equation system, the second to a equation system over the power space of the state space, the last one to Boolean equation systems. For each domain we give a semantics and show that in each case the problems to be solved are reduced stepwise.

The first transformation, $\mathbf{E}_\mu$, leads from the set of μ-calculus formulae, μL to sequences of unnested μ-calculus formulae, denoted by $\mu {L_1}^*$. This transformation was already given and proved in definition 3.8 and proved in proposition 3.9. Here we just present the transformation for the actual scenario.

$\mathbf{E}_\mu : \mu L \to \mu {L_1}^*$ is based on a mapping $\mathbf{E}'_\mu$ and is defined as follows:

$$\begin{array}{rcl}
\mathbf{E}_\mu(Q) & = & \epsilon \\
\mathbf{E}_\mu(X) & = & \epsilon \\
\mathbf{E}_\mu(\Phi_1 \wedge \Phi_2) & = & \mathbf{E}_\mu(\Phi_1)\, \mathbf{E}_\mu(\Phi_2) \\
\mathbf{E}_\mu(\Phi_1 \vee \Phi_2) & = & \mathbf{E}_\mu(\Phi_1)\, \mathbf{E}_\mu(\Phi_2) \\
\mathbf{E}_\mu([a]\Phi) & = & \mathbf{E}_\mu(\Phi) \\
\mathbf{E}_\mu(\langle a\rangle\Phi) & = & \mathbf{E}_\mu(\Phi) \\
\mathbf{E}_\mu(\sigma X.\Phi) & = & (\sigma X.\mathbf{E}'_\mu(\Phi))\ (\mathbf{E}_\mu(\Phi))
\end{array}$$

$$\begin{array}{rcl}
\mathbf{E}'_\mu(Q) & = & Q \\
\mathbf{E}'_\mu(X) & = & X \\
\mathbf{E}'_\mu(\Phi_1 \wedge \Phi_2) & = & \mathbf{E}'_\mu(\Phi_1) \wedge \mathbf{E}'_\mu(\Phi_2) \\
\mathbf{E}'_\mu(\Phi_1 \vee \Phi_2) & = & \mathbf{E}'_\mu(\Phi_1) \vee \mathbf{E}'_\mu(\Phi_2) \\
\mathbf{E}'_\mu([a]\Phi) & = & [a]\mathbf{E}'_\mu(\Phi)
\end{array}$$

$$\begin{array}{rcl}
\mathbf{E}'_{\mu}(\langle a\rangle\Phi) & = & \langle a\rangle\mathbf{E}'_{\mu}(\Phi) \\
\mathbf{E}'_{\mu}(\sigma X.\Phi) & = & X
\end{array}$$

From proposition 3.9 follows: $s \in \|\sigma X.\Phi\|_{\mathcal{V}}$ iff $s \in ([\![\mathbf{E}_{\mu}(\sigma X.\Phi)]\!]\,\mathcal{V})(X)$. Note that here we interpret the valuation function $\mathcal{V}$ as an environment.

The second transformation, $\mathbf{E}_{\mathcal{M}}$, maps a sequence of μ-calculus formulae to a fixpoint-equation system over the powerset of the state space. Formally, this is the step from the logical formulae to their semantic domain. Technically, we perform only a syntactical transformation from logical variables to set variables, from the Boolean connectives $\vee$ and $\wedge$ to the set operations $\cup$ and $\cap$, from the modal operators $[a]$ and $\langle a\rangle$ to set operators $[\![a]\!]^{\mathcal{T}}$ and $\langle\!\langle a\rangle\!\rangle^{\mathcal{T}}$.

Let $\sigma X.\Phi$ be an unnested μ-calculus formula and $\mathcal{E}$ a sequence of unnested μ-calculus formulae. The transformation $\mathbf{E}_{\mathcal{M}} : \mu L_1{}^* \rightarrow \mu\mathcal{P}(\mathcal{S})^*$ is based on a mapping $\mathbf{E}'_{\mathcal{M}}$ and defined as follows.

$$\begin{array}{rcl}
\mathbf{E}_{\mathcal{M}}(\epsilon) & = & \epsilon \\
\mathbf{E}_{\mathcal{M}}((\sigma X.\Phi)\,\mathcal{E}) & = & (\sigma X = \mathbf{E}'_{\mathcal{M}}(\Phi))\,\mathbf{E}_{\mathcal{M}}(\mathcal{E})
\end{array}$$

$$\begin{array}{rcl}
\mathbf{E}'_{\mathcal{M}}(Q) & = & \mathcal{V}(Q) \\
\mathbf{E}'_{\mathcal{M}}(X) & = & X \\
\mathbf{E}'_{\mathcal{M}}(\Phi_1 \wedge \Phi_2) & = & \mathbf{E}'_{\mathcal{M}}(\Phi_1) \cap \mathbf{E}'_{\mathcal{M}}(\Phi_2) \\
\mathbf{E}'_{\mathcal{M}}(\Phi_1 \vee \Phi_2) & = & \mathbf{E}'_{\mathcal{M}}(\Phi_1) \cup \mathbf{E}'_{\mathcal{M}}(\Phi_2) \\
\mathbf{E}'_{\mathcal{M}}([a]\Phi) & = & [\![a]\!]^{\mathcal{T}}(\mathbf{E}'_{\mathcal{M}}(\Phi)) \\
\mathbf{E}'_{\mathcal{M}}(\langle a\rangle\Phi) & = & \langle\!\langle a\rangle\!\rangle^{\mathcal{T}}(\mathbf{E}'_{\mathcal{M}}(\Phi)) \\
\mathbf{E}'_{\mathcal{M}}(\sigma X.\Phi) & = & \sigma X = \mathbf{E}'_{\mathcal{M}}(\Phi)
\end{array}$$

Recall that the semantics of a fixpoint-equation system was given in definition 3.3. Here f denotes a monotone set function on $\mathcal{P}(\mathcal{S})$.

$$\begin{array}{rcl}
[\![\epsilon]\!]\,\mathcal{V} & = & \mathcal{V} \\
[\![(\nu X = f)\,\mathcal{E}]\!]\,\mathcal{V} & = & [\![\mathcal{E}]\!]\,\mathcal{V}[X/\bigcup\{S \subseteq \mathcal{S} \mid S \supseteq f([\![\mathcal{E}]\!]\,\mathcal{V}[X/S])\}] \\
[\![(\mu X = f)\,\mathcal{E}]\!]\,\mathcal{V} & = & [\![\mathcal{E}]\!]\,\mathcal{V}[X/\bigcap\{S \subseteq \mathcal{S} \mid S \subseteq f([\![\mathcal{E}]\!]\,\mathcal{V}[X/S])\}]
\end{array}$$

Correctness of the transformation follows immediately from the definitions of the semantics: $[\![(\sigma X.\Phi)\ \mathcal{E}]\!]\ \mathcal{V} = [\![(\sigma X = \mathbf{E}'_{\mathcal{M}}(\Phi))\ \mathbf{E}_{\mathcal{M}}(\mathcal{E})]\!]\ \mathcal{V}$
In the last step the isomorphism between the powerset of the state space and a Boolean vector space allows to represent a set expression as a Boolean vector expression and equivalently as a vector of Boolean expressions. According to Bekič's theorem such a simultaneous fixpoint expression can be eliminated and substituted by a sequence of simple fixpoint expressions. In addition the set operators $[\![a]\!]^{\mathcal{T}}$ and $\langle\!\langle a \rangle\!\rangle^{\mathcal{T}}$ can be eliminated by evaluation, because here each Boolean expression describes a set expression at a particular state of the underlying transition system and at each single state the set operators can be evaluated easily.
Altogether the transformation function $\mathbf{E}_{\mathbb{B}} : \mu\mathcal{P}(\mathcal{S})^* \to \mu\mathbb{B}^*$ maps a fixpoint-equation system over sets of states to a Boolean equation system. It refers to a set of functions $\{\mathbf{E}_{\mathbb{B},1}, \ldots, \mathbf{E}_{\mathbb{B},n}\}$, where $n = |\mathcal{S}|$ is the size of the state space.

$$\begin{array}{rcl}
\mathbf{E}_{\mathbb{B}}(\epsilon) & = & \epsilon \\
\mathbf{E}_{\mathbb{B}}((\sigma X = f)\ \mathcal{E}) & = & (\sigma X_1 = \mathbf{E}_1(f)) \ldots (\sigma X_n = \mathbf{E}_n(f))\ \mathbf{E}_{\mathbb{B}}(\mathcal{E}) \\
\mathbf{E}_{\mathbb{B},i}(S) & = & \begin{cases} \mathsf{true} & \text{if } s_i \in S \\ \mathsf{false} & \text{else} \end{cases} \\
\mathbf{E}_{\mathbb{B},i}(X) & = & X_i \\
\mathbf{E}_{\mathbb{B},i}(A_1 \cup A_2) & = & \mathbf{E}_{\mathbb{B},i}(A_1) \vee \mathbf{E}_{\mathbb{B},i}(A_2) \\
\mathbf{E}_{\mathbb{B},i}(A_1 \cap A_2) & = & \mathbf{E}_{\mathbb{B},i}(A_1) \wedge \mathbf{E}_{\mathbb{B},i}(A_2) \\
\mathbf{E}_{\mathbb{B},i}(\langle\!\langle a \rangle\!\rangle^{\mathcal{T}} A) & = & \bigvee_{s_i \xrightarrow{a} s_j} \mathbf{E}_{\mathbb{B},j}(A) \\
\mathbf{E}_{\mathbb{B},i}([\![a]\!]^{\mathcal{T}} A) & = & \bigwedge_{s_i \xrightarrow{a} s_j} \mathbf{E}_{\mathbb{B},j}(A)
\end{array}$$

The semantic of a Boolean equation system was already given in section 3.2. The environment $\theta_{\mathcal{V}}$ derived from the valuation $\mathcal{V}$ is defined as above:
$\theta_{\mathcal{V}}(X_i) = \mathsf{true}$ iff $s_i \in \mathcal{V}(X)$
In order to show the correctness of the transformation $\mathbf{E}_{\mathbb{B}}$ we have to

prove for a set equation system $\mathcal{E}$ and a valuation $\mathcal{V}$:
$s_i \in ([\![\mathcal{E}]\!]\, \mathcal{V})(X)$ iff $([\![\mathbf{E}_{I\!B}(\mathcal{E})]\!]\, \theta_{\mathcal{V}})(X_i) = \mathsf{true}$.
The proof here requires Bekič's theorem 2.24 for the transformation of an n-ary simultaneous fixpoint to a nested fixpoint and the transformation of a nested fixpoint to a fixpoint-equation system given in definition 3.8 and proposition 3.9.
Altogether the transformation function $\mathbf{E}$ from a μ-calculus expression and a model to a Boolean equation system can be composed by the transformations $\mathbf{E}_\mu, \mathbf{E}_\mathcal{M}$, and $\mathbf{E}_{I\!B}$ as defined above, and it holds: $\mathbf{E}(\Phi) = (\mathbf{E}_{I\!B} \circ \mathbf{E}_\mathcal{M} \circ \mathbf{E}_\mu)(\Phi)$ where $\circ$ is the usual composition of functions, and from the correctness of these transformation with respect to the semantics given we can conclude that
$[\![\mathbf{E}(\sigma X.\Phi)]\!]\ \theta_\mathcal{V}\ (X_i) = \mathsf{true}$ iff $s_i \in \|\Phi\|^\mathcal{T}_\mathcal{V}$. □

A.3 Proofs of Chapter 8.

Theorem 8.2 For a Boolean equation system E and an environment θ it is $([\![\mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$ iff $A_{\mathcal{E},\theta}(\{a\}, S_\mathcal{E}, X_i, \rho_{\mathcal{E},\theta}, \Omega_{\mathcal{E},\theta})$ is nonempty. Moreover $A_{\mathcal{E},\theta}$ has size of $O(|\,\mathcal{E}\,|)$.

Proof: In the following we often argue with automata which differ only in their initial state, but coincide in the set of states $S_\mathcal{E}$, the transition relation $\rho_{\mathcal{E},\theta}$ and the accepting condition $\Omega_{\mathcal{E},\theta}$. Then we will explicitly talk about the automaton $A_{\mathcal{E},\theta}$ with initial state X_i.
(*) Note, if we have an accepting run r of the automaton $A_{\mathcal{E},\theta}$ with initial state X_i and a run r' of $A_{\mathcal{E},\theta}$ with initial state X_j, such that every branch b' of r' consists of a finite initial part b_f continued by a branch b of r, i.e. $b' = b_f b$, then b' fulfills also the acceptance condition $\Omega_{\mathcal{E},\theta}$ and hence r' is an accepting run of $A_{\mathcal{E},\theta}$ with initial state X_j.
Now th proof is by induction on $\mathcal{E}$.

$$\begin{array}{lll} & ([\![\epsilon]\!]\,\theta)(X_i) & = \mathsf{true} \\ \text{iff} & \theta(X_i) & = \mathsf{true} \\ \text{iff} & \rho_{\epsilon,\theta}(a, X_i) & = \mathsf{true} \\ \text{iff} & A_{\epsilon,\theta} \text{ with initial state } X_i \text{ has an accepting run.} & \end{array}$$

induction hypothesis: $\forall X_i, \mathcal{E}$ of length n, θ : $([\![\mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$ iff $A_{\mathcal{E},\theta}$ with initial state X_i has an accepting run.
Show $\forall X_i, \mathcal{E}$ of length n, θ, σ, X, f :
$([\![(\sigma X = f)\ \mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$ iff $A_{(\sigma X = f)\ \mathcal{E},\theta}$ with initial state X_i is nonempty.
$(\Longrightarrow)$
case 1 $([\![(\nu X = f)\ \mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true} = ([\![\mathcal{E}]\!]\,\theta[X/f([\![\mathcal{E}]\!]\,\theta[X/\mathsf{true}])])(X_i)$

1.1 $([\![\mathcal{E}]\!]\,\theta[X/\mathsf{false}])(X_i) = \mathsf{true}$

Then there exists an accepting run r on $A_{\mathcal{E},\theta[X/\mathsf{false}]}$ with initial state X_i and no node of r is labelled with X, because otherwise this node would be a leaf and this branch not accepted.

The tree r is then also an accepted run of $A_{(\nu X = f)\ \mathcal{E},\theta}$ with initial state X_i, because $\rho_{\mathcal{E},\theta[X/\mathsf{false}]}$ and $\rho_{(\nu X = f)\ \mathcal{E},\theta}$ coincide on all states different from X and no node of r is labelled with X. Furthermore, if a run is accepted by $\Omega_{\mathcal{E},\theta[X/\mathsf{false}]})$ then it is also accepted by the "weaker" acceptance condition $\Omega_{(\nu X = f)\ \mathcal{E},\theta}$.

1.2 $([\![\mathcal{E}]\!]\,\theta[X/\mathsf{false}])(X_i) = \mathsf{false}$

(1) Then it must be the case that $f([\![\mathcal{E}]\!]\,\theta[X/\mathsf{true}]) = \mathsf{true}$, i.e. for a satisfying set of f $\{X_{j1}, \ldots, X_{jk}\}$ it is that

$([\![\mathcal{E}]\!]\,\theta[X/\mathsf{true}])(X_{jl}) = \mathsf{true}$ for $1 \le l \le k$. For each X_{jl} there is according to the induction hypothesis an accepting run r_{jl} of the automaton $A_{\mathcal{E},\theta[X/\mathbf{true}]}$ with initial state X_{jl}.

(2) We show now that there exists an accepting run r_X on $A_{(\nu X = f)\ \mathcal{E},\theta}$ with initial state X.

Consider a tree $r_{X'}$ where the root is labelled with X and the successors of the root are $r_{j1}, \ldots, r_{jk}$ from (1). Let $r_{X''}$ be the tree $r_{X'}$ where all leaves labelled with X are substituted by a copy of $r_{X'}$. Continue substitution of X-labelled leaves by $r_{X'}$ getting finally the tree r_X. It is easy to see that r_X follows the transition function $\rho_{(\nu X = f)\ \mathcal{E},\theta}$ because it coincides on all $r_{j1}, \ldots, r_{jk}$ with $\rho_{\mathcal{E},\theta[X/\mathbf{true}]}$ on all states apart from X and at the nodes labelled with X the sucessors are labelled with a satisfying set of f according to the transition function $\rho_{(\nu X=f)\ \mathcal{E},\theta}(a,X) = f$. It remains to show that the run r_X is also accepted. Note that each branch b_X of r_X consists either of a finite initial part followed by a branch from some r_{jl}, where $1 \leq l \leq k$, in which no node is labelled with X, or b_X contains infinitely many nodes labelled with X. In the first case b_X is accepted by the fact that each branch containing no X-labelled node which is accepted by $A_{\mathcal{E},\theta[X/\mathbf{true}]}$ with initial state X_{jl} is also accepted by $A_{(\nu X=f)\ \mathcal{E},\theta}$ with initial state X_{jl} and argument (*) above. In the latter case b_X is accepted by the acceptance condition $\Omega_{(\nu X=f)\ \mathcal{E},\theta}$, because X is a ν-variable and gets the least index.

(3) We finally have to show that there is an accepting run r on $A_{(\nu X=f)\ \mathcal{E},\theta}$ with initial state X_i. According to the assumptions it must be the case that $([\![\mathcal{E}]\!]\ \theta[X/\mathsf{true}])(X_i) = \mathsf{true}$ and with the induction hypothesis we know that there must be an accepting run r' of $A_{\mathcal{E},\theta[X/\mathbf{true}]}$ with initial state X_i. Now take the run r' and substitute each leaf labelled with X by the run r_X from (2). It is easy to see that r_X follows the transition function $\rho_{(\nu X=f)\ \mathcal{E},\theta}$. Each branch of r' containing no X and accepted bt $\Omega_{\mathcal{E},\theta[X/\mathbf{true}]}$ is also a branch of r and accepted by $\Omega_{(\nu X=f)\ \mathcal{E},\theta}$. All other branches are accepted by argument (*) above.

case 2 $([\![(\mu X=f)\ \mathcal{E}]\!]\ \theta)(X_i) = ([\![\mathcal{E}]\!]\ \theta[X/([\![\mathcal{E}]\!]\ \theta[X/\mathsf{false}])])(X_i) = \mathsf{true}$

2.1 $([\![\mathcal{E}]\!]\ \theta[X/\mathsf{false}])(X_i) = \mathsf{true}$

According to the induction hypothesis there is an accepting run r on

$A_{\mathcal{E},\theta[X/\mathbf{false}]}$ with initial state X_i. No node of r is labelled with X, since such a node would be a leaf of a not accepted branch. Hence r is also an accepting run of $A_{(\mu X=f)\ \mathcal{E},\theta}$ with initial state X_i, because $\rho_{(\mu X=f)\ \mathcal{E},\theta}$ and $\rho_{\mathcal{E},\theta[X/\mathbf{false}]}$ coincide on all states apart from X and X does not appear in r. Then $\Omega_{(\mu X=f)\ \mathcal{E},\theta}$ accepts every branch that is accepted by $\Omega_{\mathcal{E},\theta[X/\mathbf{false}]}$.

2.2 $([\![\mathcal{E}]\!]\ \theta[X/\mathsf{false}])(X_i) = \mathsf{false}$

(1) Then it must be the case that $f(\llbracket\mathcal{E}\rrbracket\,\theta[X/\mathbf{false}]) = \mathsf{true}$, i.e. there must be a satifying set $\{X_{j1}, \ldots, X_{jk}\}$ for some $k \in I\!N$ of f such that $(\llbracket\mathcal{E}\rrbracket\,\theta[X/\mathbf{false}])(X_{jl}) = \mathsf{true}$ for $1 \leq l \leq k$. According to the induction hypothesis for each $1 \leq l \leq k$ there is an accepting run r_{jl} on $A_{\mathcal{E},\theta[X/\mathbf{false}]}$ with initial state X_{jl}. Since no node is labelled with X each tree r_{jl} is also an accepting run of $A_{(\mu X=f)\ \mathcal{E},\theta}$ with initial state X_{jl}, because the transition functions $\rho_{\mathcal{E},\theta[X/\mathbf{false}]}$ and $\rho_{(\mu X=f)\ \mathcal{E},\theta}$ coincide on all states apart from X and $\Omega_{(\mu X=f)\ \mathcal{E},\theta}$ accepts every infinite branch that $\Omega_{\mathcal{E},\theta[X/\mathbf{false}]}$ accepts.

(2) Show now that $A_{(\mu X=f)\ \mathcal{E},\theta}$ with initial state X has an accepting run r_X. Let r_X be the tree where the root is labelled with X and the successors of the root are the trees $r_{j1}, \ldots, r_{jk}$ from (1). Since $\{X_{j1}, \ldots, X_{jk}\}$ is an accepting set of f, r_X follows the transition function $\rho_{(\mu X=f)\ \mathcal{E},\theta}(a, X) = f$, which coincides with $\rho_{\mathcal{E},\theta[X/\mathbf{false}]}$ on all states other than X. With argument (*) follows that r_X is also accepted by $A_{(\mu X=f)\ \mathcal{E},\theta}$.

(3) It remains to construct an accepting run r of $A_{(\mu X=f)\ \mathcal{E},\theta}$ with initial state X_i. We know that $(\llbracket\mathcal{E}\rrbracket\,\theta[X/\mathbf{true}])(X_i) = \mathsf{true}$ and according to the induction hypothesis there must be an accepting run r' of $A_{\mathcal{E},\theta[X/\mathbf{true}]}$ with initial state X_i Let r be as r' where all leaves labelled with X are substituted by r_X from (2). Note that all branches of r' containing no X are also accepted by $A_{(\mu X=f)\ \mathcal{E},\theta}$ with initial state X_i. All other branches are accepted by argument (*).

($\Longleftarrow$) We make use of complementation of Boolean equation systems and alternating automata with parity condition.
Assume $(\llbracket\mathcal{E}\rrbracket\,\theta)(X_i) = \mathsf{false}$, then by lemma 3.35 $(\llbracket\overline{\mathcal{E}}\rrbracket\,\overline{\theta})(X_i) = \mathsf{true}$ and according to the first part of the proof we know that $A_{\overline{\mathcal{E}},\overline{\theta}}$ with initial state X_i is nonempty. The complementation of alternating automata with parity condition is easy (see [EJ91]): the complement of $A_{\overline{\mathcal{E}},\overline{\theta}}$ with initial state X_i is $A_{\mathcal{E},\theta}$ with initial state X_i, and if $A_{\overline{\mathcal{E}},\overline{\theta}}$ has an accepting run, then $A_{\mathcal{E},\theta}$ is empty. □

Theorem 8.7 Player II has a winning strategy for the game on $\mathcal{G}_\mathcal{E}$ with initial vertex i iff $([\![\mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$. Moreover $|\mathcal{G}_\mathcal{E}| = O(|\mathcal{E}|)$.

Proof: It follows immediately from construction that the size of $\mathcal{G}_\mathcal{E}$ is linear in the size of $\mathcal{E}$.

$(\Longleftarrow)$

Assume $([\![\mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$. According to lemma 3.36 there exists a Boolean equation system $\mathcal{E}'$ in conjunctive form, where $\mathcal{E}' \leq \mathcal{E}$ and and $[\![\mathcal{E}']\!]\,\theta = [\![\mathcal{E}]\!]\,\theta$. All conjunctions of $\mathcal{E}$ are contained in $\mathcal{E}'$, but from each disjunction of $\mathcal{E}$ there is only one disjunct in the corresponding equation of $\mathcal{E}'$. Consider the game graph $\mathcal{G}_{\mathcal{E}'}$. In every play on $\mathcal{G}_{\mathcal{E}'}$ player II never takes a move, because there are no vertices labelled with $\vee$.

We now want to show by contradiction that for the game on $\mathcal{G}_{\mathcal{E}'}$ with initial vertex labelled with i player II wins every play. Then a winning strategy for player II is to choose in every I-labelled vertex this successor which is also contained in $\mathcal{G}_{\mathcal{E}'}$.

Assume p is a play of $\mathcal{G}_{\mathcal{E}'}$ with initial vertex i which is won by player I. Let j be the least vertex in $lim(p)$. For each vertex in $lim(p)$ it must be the case that there is (at least) one of its successors in $lim(p)$ and also (at least) one of its predecessors. Moreover there must be a subsequence $p' = v_0, v_1, \ldots, v_n$ of p, where $v_0 = v_n = j$ and $v_k \in lim(p) \setminus \{j\}$ for $0 < j < n$. We now want to show that in the Boolean equation system $\mathcal{E}'$ the variable X_i has the solution false. Assume $j \neq \mathsf{false}$. Consider all equations $\sigma X_k = f_k$ in $\mathcal{E}'$ where k is a vertex in $lim(p)$. We know that $\mu X_j = f_j$, the equation corresponding to vertex j, is the least one with respect to $\trianglelefteq$ among these equations. Now p' defines a sequence of substitution steps (lemma 6.3) in $\mathcal{E}'$: first the equation corresponding to vertex v_1 is substituted into f_j giving $\mu X_j = f_j^1$, then the equation corresponding to vertex v_2 into f_j^1 giving $\mu X_j = f_j^2$ and so on. After $n-1$ substitution steps we have an occurrence of the variable X_j on the right-hand side of $\mu X_j = f_j^{n-1}$ and may apply an elimination step (lemma 6.2). Because $f_j^{(n-1)}$ can only consist of a disjunction or a single variable the equation evaluates to $\mu X_j = \mathsf{false}$ and it is the case that $([\![\mathcal{E}']\!]\,\theta)(X_j) = ([\![\mathcal{E}]\!]\,\theta)(X_j) = \mathsf{false}$. The initial

part of p defines a sequence from X_i to the first occurrence of X_j in p and going this initial sequence backwards applying substitution steps for constants (lemma 3.19) we get that $([\![\mathcal{E}']\!]\,\theta)(X_i) = ([\![\mathcal{E}]\!]\,\theta)(X_i) =$ false which contradicts the assumption. For the case $j =$ false we just have to apply the last argument above and get the same contradiction.
$(\Longrightarrow)$
The other direction follows by duality arguments. Analogously to the first case of the proof we can show that from $([\![\mathcal{E}]\!]\,\theta)(X_i) =$ false it follows that player I has a winning strategy. Since only one of the players can have a winning strategy and $([\![\mathcal{E}]\!]\,\theta)(X_i)$ must be either true or false the proof is complete. □

A.4 Proofs of Chapter 9.

Theorem 9.4 Given an infinite Boolean equation system $\mathcal{E} = \sigma_1\mathcal{B}_1 \ldots \sigma_n\mathcal{B}_n$ and an environment θ there exists an infinite Boolean equation system $\mathcal{E}' = \sigma_1\mathcal{B}'_1 \ldots \sigma_n\mathcal{B}'_n$ such that $\mathcal{E}'$ contains no disjunctions on the right-hand side. In particular:

- If $\sigma_k X_j = \bigwedge_{i\in I} X_i$ is an equation in block $\mathcal{B}_j$ of $\mathcal{E}$ then it is also an equation in $\mathcal{B}'_j$ of $\mathcal{E}'$.
- If $\sigma_k X_j = X_i$ is an equation in block $\mathcal{B}_j$ of $\mathcal{E}$ then it is also an equation in $\mathcal{B}'_j$ of $\mathcal{E}'$.
- If $\sigma_k X_j = \bigvee_{i\in I} X_i$ is an equation in block $\mathcal{B}_j$ of $\mathcal{E}$ and I is nonempty, then for some $k \in I$ the equation $\sigma_k X_j = X_i$ is in block $\mathcal{B}'_j$ of $\mathcal{E}'$. If I is empty then $\sigma_k X_j =$ false is an equation of $\mathcal{B}'_j$ of $\mathcal{E}'$.
- $[\![\mathcal{E}]\!]\,\theta = [\![\mathcal{E}']\!]\,\theta$

Proof: by induction on the structure of $\mathcal{E}$.

The argumentation here is similar to the one in the proof for the finite case (proposition 3.36). There in the induction step we have to construct one Boolean equation system based on two others (lemma A.1). In contrast to the finite case here we have to construct one Boolean equation system based on countable number of other ones. However, the idea and technique is very much the same.

base case: Let $\mathcal{E} = \epsilon$ and θ be an environment. Then $\mathcal{E}' = \epsilon$ fulfills the requirements.
induction hypothesis: for each infinite Boolean equation system $\mathcal{E}$ with fewer blocks than n and environment θ we can find an infinite Boolean equation system $\mathcal{E}'$ having no disjunctions with more than one disjunct on its right-hand side and $[\![\mathcal{E}]\!]\,\theta = [\![\mathcal{E}']\!]\,\theta$.
induction step: assume that $\mu\mathcal{B}\ \mathcal{E}$ is an infinite Boolean equation system, that for some index set I $lhs(\mathcal{B}) = \{X_i \mid i \in I\}$, and that θ is an environment. Then
$[\![\mu\mathcal{B}\ \mathcal{E}]\!]\,\theta = [\![\mathcal{E}]\!]\,\theta[X_I/\mu X_I.\mathcal{B}([\![\mathcal{E}]\!]\,\theta)]$
Now we proceed as follows:
Define

$$\begin{aligned} b^0 &\stackrel{\text{def}}{=} \mathsf{false}^I \\ b^{\alpha+1} &\stackrel{\text{def}}{=} \mathcal{B}([\![\mathcal{E}]\!]\,\theta[X_I/b^\alpha \\ b^\lambda &\stackrel{\text{def}}{=} \bigvee_{\alpha<\lambda} b^\alpha \end{aligned}$$

for α an ordinal, λ a limit ordinal. By proposition 2.20, $b = b^\kappa$ for some κ, and since the b^α form an ascending chain in the product lattice $\mathbb{B}^I$, κ is countable.
Climbing up the b^α we first construct a system $\mathcal{E}'$ having the same solution as $\mathcal{E}$ for all $\theta[X_I/b^\alpha]$. Then $\mathcal{E}'$ and $\mathcal{E}$ also have the same solution for the least fixpoint b, i.e. $[\![\mathcal{E}]\!]\,\theta[X_I/b] = [\![\mathcal{E}']\!]\,\theta[X_I/b]$. Afterwards we construct a block $\mu\mathcal{B}'$, also climbing up the b^α, such that $\mathcal{B}([\![\mathcal{E}]\!]\,\theta[X_I/b^\alpha]) = \mathcal{B}'([\![\mathcal{E}]\!]\,\theta[X_I/b^\alpha])$. Then we get also $\mathcal{B}([\![\mathcal{E}]\!]\,\theta[X_I/b] = \mathcal{B}'([\![\mathcal{E}']\!]\,\theta[X_I/b] = b$ The theorem follows then by application of the definition of the semantic.
We first construct a system $\mathcal{E}'$, such that
(1) $[\![\mathcal{E}]\!]\,\theta[X_I/b^\alpha] = [\![\mathcal{E}']\!]\,\theta[X_I/b^\alpha]$ for all α.
For this we use the fact that according to the induction hypothesis for each α there exists an $\mathcal{E}'_\alpha$ having the required form and $[\![\mathcal{E}]\!]\,\theta[X_I/b^\alpha] = [\![\mathcal{E}'_\alpha]\!]\,\theta[X_I/b^\alpha]$.
The construction of $\mathcal{E}'$ works as follows:
For $\alpha = 0$ select all $X \in lhs(\mathcal{E})$ where $([\![\mathcal{E}]\!]\,[X_I/b^0])(X) = \mathsf{true}$. We know that then also $([\![\mathcal{E}'_0]\!]\,[X_I/b^0])(X) = \mathsf{true}$. For each of these X let

the equation $\sigma X = f'$ from $\mathcal{E}'_0$ be an equation of $\mathcal{E}'$ in the corresponding block. Whatever the remaining equations of $\mathcal{E}'$ will be (they might all be false, see 3.19), we have
(*) $(\llbracket \mathcal{E}' \rrbracket [X_I/b^0])(X) = \mathsf{true} = (\llbracket \mathcal{E} \rrbracket [X_I/b^0])(X)$.
For each α select all variables $X_j \in lhs(\mathcal{E})$ such that

$$\begin{aligned} (\llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha+1}] \quad)(X_j) &= \mathsf{true} \\ (\llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha}])(X_j) &= \mathsf{false} \end{aligned}$$

and for all these X_j let the equation $\sigma X_j = f'_j$ in $\mathcal{E}'_\alpha$ be an equation of $\mathcal{E}'$, such that $\sigma X_j = f'_j$ is contained in the corresponding block to the one of $\mathcal{E}$ containing $\sigma X_j = f_j$.
The argument now is by induction. Assume that for all other X_k (having lower signature)

$$\begin{aligned} \text{if} \quad (\llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha}])(X_k) &= \mathsf{true} \\ \text{then } (\llbracket \mathcal{E}' \rrbracket \theta[X_I/b^{\alpha}])(X_k) &= \mathsf{true}. \end{aligned}$$

This is, because of monotonicity, for all $\beta \geq \alpha$

$$\begin{aligned} \text{if} \quad (\llbracket \mathcal{E}' \rrbracket \theta[X_I/b^{\alpha}])(X_k) &= \mathsf{true} \\ \text{then } (\llbracket \mathcal{E}' \rrbracket \theta[X_I/b^{\beta}])(X_k) &= \mathsf{true}, \end{aligned}$$

Hence we know that for all these X_k, where
$(\llbracket \mathcal{E}'_\alpha \rrbracket \theta)(X_k) = \mathsf{true} = (\llbracket \mathcal{E} \rrbracket \theta)(X_k)$ that also $(\llbracket \mathcal{E}' \rrbracket \theta)(X_k) = \mathsf{true}$.
With the base case (*) we can now conclude (1). (See also the argumentation for lemma A.1 in combination with lemma 9.3).
Furthermore

$$\begin{aligned} \text{if} \quad \llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha}] &= \llbracket \mathcal{E}' \rrbracket \theta[X_I/b^{\alpha}] \\ \text{then } \mathcal{B}(\llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha}]) &= \mathcal{B}(\llbracket \mathcal{E}' \rrbracket \theta[X_I/b^{\alpha}]) = b^{\alpha+1} \end{aligned}$$

From the above we also can conclude
(2) $\mu X_I.\mathcal{B}(\llbracket \mathcal{E} \rrbracket \theta) = \mu X_I.\mathcal{B}(\llbracket \mathcal{E}' \rrbracket \theta)$
Next we construct $\mu\mathcal{B}'$ in such a way, that for each α
$\mathcal{B}(\llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha}]) = \mathcal{B}'(\llbracket \mathcal{E} \rrbracket \theta[X_I/b^{\alpha}])$.

- Let each equation $\mu X = \bigwedge_{j \in J} X_j$ for some index set J in $\mu\mathcal{B}$ be also an equation of $\mu\mathcal{B}'$.

- If there is an equation in $\mu\mathcal{B}$ of the form $\mu X = X_i$ or $\mu X = \bigvee X_i$, where the disjunction contains only a single disjunct, then let $\mu X = X_i$ be an equation of $\mu\mathcal{B}'$.
- If there is an equation in $\mu\mathcal{B}$ of the form $\mu X = \bigvee_{j\in J} X_j$ for some index set J and there is one of the disjuncts true, then let $\mu X = \mathsf{true}$ be an equation of $\mu\mathcal{B}'$.
- For each equation $\mu X = \bigvee_{j\in J} X_j$ in $\mu\mathcal{B}$ where $([\![\mu\mathcal{B}\ \mathcal{E}]\!]\,\theta)(X) = \mathsf{false}$ choose any of the disjuncts from $\bigvee_{j\in J} X_j$, say X_j, and let $\mu X = X_j$ be an equation of $\mu\mathcal{B}'$.
- In all other cases for $X_i \in lhs(\mathcal{B})$ we have $([\![\mu\mathcal{B}\ \mathcal{E}]\!]\,\theta)(X_i) = \mathsf{true}$ and the equation for X_i is of the form $\mu X_i = \bigvee_{j\in J} X_j$ for some index set J. For each of these X_i there exists an α such that $(\theta[X_I/b^{\alpha+1}])(X_i) = \mathsf{true}$ and $(\theta[X_I/b^{\alpha}])(X_i) = \mathsf{false}$. Select from $\bigvee_{j\in J} X_j$ a variable X_j, such that $([\![\mathcal{E}]\!]\,\theta[X_I/b^{\alpha}])(X_j) = \mathsf{true}$ and let $\mu X = X_j$ an equation in $\mu\mathcal{B}'$. Hence $(\mathcal{B}'([\![\mathcal{E}]\!]\,\theta[X_I/b^{\alpha}]))_i = \mathsf{true}$ according to the choice of X_j.

It follows from the construction that
(3) $\mathcal{B}([\![\mathcal{E}]\!]\,\theta[X_I/b^{\alpha+1}]) = \mathcal{B}'([\![\mathcal{E}]\!]\,\theta[X_I/b^{\alpha+1}])$
Altogether we have then that

$$\begin{array}{rcll}
[\![\mu\mathcal{B}\ \mathcal{E}]\!]\,\theta & = & [\![\mathcal{E}]\!]\,\theta[X_I/\mu X_I.\mathcal{B}([\![\mathcal{E}]\!]\,\theta)] & \\
& = & [\![\mathcal{E}]\!]\,\theta[X_I/\mu X_I.\mathcal{B}'([\![\mathcal{E}]\!]\,\theta)] & (3) \\
& = & [\![\mathcal{E}']\!]\,\theta[X_I/\mu X_I.\mathcal{B}'([\![\mathcal{E}']\!]\,\theta)] & (1),(2) \\
& = & [\![\mu\mathcal{B}'\ \mathcal{E}']\!]\,\theta &
\end{array}$$

The dual case for $\nu\mathcal{B}\ \mathcal{E}$ works similarly. □

Lemma 9.9 Let

- $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ be set based Boolean equation systems,
- $M, N, N' \subseteq \mathcal{S}$, where $N \subseteq N'$
- assuming that for all $j \in J$ it is $Y \neq X_j$

$$\begin{array}{rcl}
f_M & = & (Y, N, \rho_Y) \ \wedge \ \bigwedge_{j\in J} (X_j, M_j, \rho_j), \\
f_{N'} & = & \bigwedge_{k\in K} (Y_k, N_k, \rho_k),
\end{array}$$

$$f'_M = \bigwedge_{k \in K} (Y_k, N_k, \rho_Y \circ \rho_k) \wedge \bigwedge_{j \in J} (X_j, M_j, \rho_j),$$

- θ an environment.

Then $[\![\mathcal{E}_1 \ (\sigma_X(X, M) = f_M) \ \mathcal{E}_2 \ (\sigma_Y(Y, N') = f_N) \ \mathcal{E}_3]\!] \, \theta$

$= [\![\mathcal{E}_1 \ (\sigma_X(X, M) = f'_M) \ \mathcal{E}_2 \ (\sigma_Y(Y, N') = f_N) \ \mathcal{E}_3]\!] \, \theta.$

Proof: Transform both equation systems to infinite Boolean equation systems $\mathcal{E}_4$ and $\mathcal{E}_5$.

For $m \in M$ and $n \in N$ $\mathcal{E}_4$ contains the equations

$$\begin{aligned} \sigma_X X_m &= \bigwedge_{n \in \rho_Y(m)} Y_n \wedge \bigwedge_{j \in J} \bigwedge_{t \in \rho_j(m)} X_{j,t} \quad \text{and} \\ \sigma_Y Y_n &= \bigwedge_{k \in K} \bigwedge_{n' \in \rho_k(n)} Y_{k,n'} \end{aligned}$$

in $\mathcal{E}_4$. According to lemmata 9.3, 6.3 we can apply infinitely many substitution steps in the infinite Boolean equation system $\mathcal{E}_4$ substituting all the Y_n, and getting the new equation

$$\begin{aligned} \sigma_X X_m &= \bigwedge_{n \in \rho_Y(m)} \bigwedge_{k \in K} \bigwedge_{n' \in \rho_k(n)} Y_{k,n'} \wedge \bigwedge_{j \in J} \bigwedge_{t \in \rho_j(m)} X_{j,t} \\ &= \bigwedge_{k \in K} \bigwedge_{n' \in (\rho_k \circ \rho_Y)(m)} Y_{k,n'} \wedge \bigwedge_{j \in J} \bigwedge_{t \in \rho_j(m)} X_{j,t} \end{aligned}$$

This is an equation of the infinite Boolean equation system $\mathcal{E}_5$. □

Lemma 9.10 Let

- $\mathcal{E}_1$ and $\mathcal{E}_2$ be set based Boolean equation systems,
- $\sigma(X, M) = (X, M, \rho) \wedge \bigwedge_{i \in I} (X_i, M_i, \rho_i)$ a set based Boolean equation,
- θ an environment, and
- $\theta' \overset{\text{def}}{=} [\![\mathcal{E}_1 \ (\sigma(X, M) = (X, M, \rho) \wedge \bigwedge_{i \in I} (X_i, M_i, \rho_i)) \ \mathcal{E}_2]\!] \, \theta$

If $\sigma = \nu$ then $\theta' = [\![\mathcal{E}_1 (\sigma(X, M) = \bigwedge_{i \in I} (X_i, M_i, \rho_i \circ \rho^*)) \mathcal{E}_2]\!] \, \theta.$

If $\sigma = \mu$ and ρ is wellfounded then θ' is as in the case for $\sigma = \nu$,

if ρ is not wellfounded then $\theta'((X, M)) = \mathsf{false}$.

Proof: In a first step the set based equation system is transformed to an infinite Boolean equation system, where the set equation

$$\sigma(X, M) = (X, M, \rho) \ \wedge \ \bigwedge_{i \in I} (X_i, M_i, \rho_i)$$

for $s \in M$ is mapped to a block $\sigma\mathcal{B}$ containing the equations

$$\sigma X_s = \bigwedge_{s' \in \rho(s)} X_{s'} \ \wedge \ \bigwedge_{i \in I} \bigwedge_{s' \in \rho_i(s)} Z_{i,s'}.$$

The equation

$$\sigma(X, M) = \bigwedge_{i \in I} (X_i, M_i, \rho_i \circ \rho^*)$$

is mapped to a block $\sigma\mathcal{B}^*$ in an infinite Boolean equation system, containing the equations

$$\sigma X_s = \bigwedge_{i \in I} \bigwedge_{s' \in (\rho_i \circ \rho^*)(s)} Z_{i,s'}$$

We will abbreviate the (infinite) vector of all X_i for $i \in I$ by X.
Let $\mathcal{E}_1' \overset{\text{def}}{=} \mathbf{T}(\mathcal{E}_1)$ and $\mathcal{E}_2' \overset{\text{def}}{=} \mathbf{T}(\mathcal{E}_2)$. For the cases that ρ is wellfounded or $\sigma = \nu$ we have to show that $[\![\mathcal{E}_1' \ \sigma\mathcal{B} \ \mathcal{E}_2']\!]\,\theta = [\![\mathcal{E}_1' \ \sigma\mathcal{B}^* \ \mathcal{E}_2']\!]\,\theta$ and according to lemma 3.14 we just have to show the equivalence above only for the case $\mathcal{E}_1 = \epsilon$, i.e. $[\![\sigma\mathcal{B} \ \mathcal{E}_2']\!]\,\theta = [\![\sigma\mathcal{B}^* \ \mathcal{E}_2']\!]\,\theta$, and according to the definition of the semantics it suffices to show that $\sigma X.\mathcal{B}([\![\mathcal{E}_2']\!]\,\theta) = \sigma X.\mathcal{B}^*([\![\mathcal{E}_2']\!]\,\theta)$
Now we want to apply a substitution step to each $X_{s'}$. For applying infinitely many substitutions within block $\sigma\mathcal{B}$ we need proposition 2.17(6) and lemma 9.3 rather than lemma 9.9.

$$\begin{aligned}
\sigma X_s \quad &= \quad \bigwedge_{s' \in \rho(s)} \Big(\bigwedge_{s'' \in \rho(s')} X_{s''} \ \wedge \ \bigwedge_{i \in I} \bigwedge_{s'' \in \rho_i(s')} Z_{i,s''} \Big) \ \wedge \ \bigwedge_{i \in I} \bigwedge_{s' \in \rho_i(s)} Z_{i,s'} \\
&= \quad \bigwedge_{s'' \in \rho(\rho(s))} X_{s''} \ \wedge \ \bigwedge_{i \in I} \bigwedge_{s'' \in (\rho_i \circ \rho)(s)} Z_{i,s''} \ \wedge \ \bigwedge_{i \in I} \bigwedge_{s' \in \rho_i(s)} Z_{i,s'} \\
&= \quad \bigwedge_{s' \in \rho(\rho(s))} X_{s'} \ \wedge \ \bigwedge_{i \in I} \bigwedge_{s' \in ((\rho_i \circ \rho) \cup \rho_i)(s)} Z_{i,s'}
\end{aligned}$$

$\ldots$ applying these substitution steps $log_2(n)$ times

$$= \bigwedge_{s' \in \rho^n(s))} X_{s'} \wedge \bigwedge_{i \in I} \bigwedge_{s' \in (\rho_i \circ (\rho^0 \cup \rho^1 \cup \ldots \rho^{n-1}))(s)} Z_{i,s'}$$

Let these equations be collected in a block $\mathcal{B}^n$ for $n \in 2^m, m \in \mathbb{N}$
It follows from proposition 2.17(6) that $[\![\sigma \mathcal{B}^n\ \mathcal{E}'_2]\!]\,\theta = [\![\sigma \mathcal{B}\ \mathcal{E}'_2]\!]\,\theta$ for all $\mathcal{B}^n$.
Define

$$\begin{aligned} b &\stackrel{\text{def}}{=} \sigma X.\mathcal{B}([\![\mathcal{E}'_2]\!]\,\theta \\ \theta_{sol} &\stackrel{\text{def}}{=} [\![\sigma \mathcal{B}\ \mathcal{E}'_2]\!]\,\theta \\ b^* &\stackrel{\text{def}}{=} \sigma X.\mathcal{B}^*([\![\mathcal{E}'_2]\!]\,\theta \\ \theta^*_{sol} &\stackrel{\text{def}}{=} [\![\sigma \mathcal{B}^*\ \mathcal{E}'_2]\!]\,\theta \end{aligned}$$

and it follows that

$$b = \sigma X.\mathcal{B}^n([\![\mathcal{E}]\!]\,'_2\theta)$$

With lemmata 3.19 and 3.20 it is the case that $\theta_{sol} = [\![\sigma\mathcal{B}]\!]\,\theta_{sol}$ and $b = \mathcal{B}(\theta_{sol})$, and also $\theta^*_{sol} = [\![\sigma\mathcal{B}^*]\!]\,\theta^*_{sol}$ and $b^* = \mathcal{B}^*(\theta^*_{sol})$
We abbreviate $\theta_{sol}[X/b](X_s)$ by b_s and $\theta_{sol}[X/\mathcal{B}(\theta_{sol})](X_s)$ by $(\mathcal{B}(\theta_{sol}))(X_s)$, and analogously for b^* and $\mathcal{B}^*$. Now we assume $\sigma = \nu$.
It suffices to show that $b = b^*$ and for that purpose we show that
(1) $\nu X.\mathcal{B}^*(\theta_{sol}) = b$ and (2) $\nu X.\mathcal{B}(\theta^*_{sol}) = b^*$
(1) implies that $\mathcal{B}^*([\![\mathcal{E}]\!]\,'_2\theta[X/b]) = b$ and hence $b \leq b^*$,
(2) implies that $\mathcal{B}([\![\mathcal{E}]\!]\,'_2\theta[X/b^*]) = b^*$ and hence $b^* \leq b$.
Show now $\nu X.\mathcal{B}^*(\theta_{sol}) = b$
(i) Because in $\mathcal{B}^*$ there is no free X_s on the right-hand side, it is the case that $\nu X.\mathcal{B}^*(\theta_{sol}) = \mathcal{B}^*(\theta_{sol})$
If for $\mathcal{B}^*(\theta_{sol})$ and an equation $\nu X_s = \bigwedge_{i\in I} \bigwedge_{s' \in (\rho_i \circ \rho^*)(s)} Z_{i,s'}$ in $\mathcal{B}^*$ we have that $(\bigwedge_{i\in I} \bigwedge_{s' \in (\rho_i \circ \rho^*)(s)} Z_{i,s'})(\theta_{sol}) = \mathsf{false}$ then there for some $Z_{i,s'}$ it must be that $\theta_{sol}(Z_{i,s'}) = \mathsf{false}$. Then we can find a $\mathcal{B}^n$, where the equation for X_s has this $Z_{i,s'}$ on its right-hand side and also $(\mathcal{B}^n(\theta_{sol}))(X_s) = \mathsf{false}$ and hence we have also $(\nu X.\mathcal{B}^n(\theta_{sol}))(X_s) = \mathsf{false}$ and also $(\nu X.\mathcal{B}(\theta_{sol}))(X_s) = b_s = \mathsf{false}$. Therefore is $b \leq \mathcal{B}^*(\theta_{sol})$.
(ii) Define $b^0 \stackrel{\text{def}}{=} \mathsf{true}^I$ and $b^{\alpha+1} \stackrel{\text{def}}{=} \mathcal{B}(\theta_{sol}[X/b^\alpha])$.
Assume that $(\nu X.\mathcal{B}(\theta_{sol}))(X_s) = \mathsf{false}$.

Show that then also $(\mathcal{B}^*(\theta_{sol}))(X_s) = \mathsf{false}$. Then there must be some α such that $b_s^\alpha = \mathsf{false}$ and $b_s^{\alpha-1} = \mathsf{true}$. ($\alpha$ is called the signature of X_s.)
If $\alpha = 1$ then there must be a $Z_{i,s'}$ for some $i \in I$, $s' \in \rho_i(s)$, where $\theta_{sol}(Z_{i,s'}) = \mathsf{false}$. But then it is also $(\mathcal{B}^*(\theta_{sol}))(X_s) = \mathsf{false}$.
If $\alpha > 1$ then there must be a X_s' for some $s' \in \rho(s)$ with $\theta_{sol}(X_{s'}) = \mathsf{false}$ and X_s' having a signature $\alpha' < \alpha$. Applying this argument repeatedly then the signature eventually reaches 0, and then we have a $Z_{i,s'}$ for some $i \in I$, $s' \in (\rho_i \circ \rho^n)(s)$, for some n, such that $\theta_{sol}(Z_{i,s'}) = \mathsf{false}$. Hence it is $(\mathcal{B}^*(\theta_{sol}))(X_s) = \mathsf{false}$.
Altogether from $(\nu X.\mathcal{B}(\theta_{sol}))(X_s) = \mathsf{false}$ follows that $(\mathcal{B}^*(\theta_{sol}))(X_s) = \mathsf{false}$ and hence $\mathcal{B}^*(\theta_{sol}) \leq b$.
From (i) and (ii) we can conclude that $\mathcal{B}^*(\theta_{sol}) = b$
When showing that $b^* = \mathcal{B}(\theta^*_{sol})$ apply the same arguments as above to θ^*_{sol} instead of θ_{sol}. From (i) follows then that $\nu X.\mathcal{B}(\theta^*_{sol}) \leq b^*$, from (ii) that $b^* \leq \nu X.\mathcal{B}(\theta^*_{sol})$.
For the case $\sigma = \mu$ note that if ρ is wellfounded for each $S \in M$ there exists some $n \in \mathbb{N}$ such that $\rho^n(s) = \emptyset$ and the equivalence of $\mathcal{B}^n$ and $\mathcal{B}^*$ follows immediately.
If ρ is not wellfounded then define $b^0 = \mathsf{false}^I$ and $b^{\alpha+1} = \mathcal{B}(\theta_{sol}[X/b^\alpha])$. Assume an $X_{s'}$ being true at the least fixpoint and let α be its signature. For all $s'' \in \rho(s')$ $X_{s''}$ must be true and have a lower signature. Repeat this argument for $X_{s''}$. Because ρ is not wellfounded we can find an infinite chain of decreasing signatures, which is a contradiction.

□

Bibliography

[AC88] A. Arnold and P. Crubille. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29:57–66, 1988.

[AKM95] S. Ambler, M. Kwiatkowska, and N. Measor. Duality and the completeness of the modal mu-calculus. *Theoretical Computer Science*, 151(1):3–27, 1995.

[And92] H.R. Andersen. Model checking and boolean graphs. In *Proceedings of 4th European Symposium on Programming, ESOP'92*, volume 582 of *Lecture Notes on Computer Science*, 1992.

[And93] H.R. Andersen. *Verification of temporal properties of concurrent systems.* PhD thesis, Aarhus University, 1993.

[And94a] H.R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.

[And94b] H.R. Andersen. On model checking infinite-state systems. In *Proceedings of LFCS'94*, volume 813 of *Lecture Notes in Computer Science*, pages 8–17. Springer, 1994.

[BC96] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proceedings of TACAS'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 1996.

[BCM$^+$92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[Bek84] H. Bekič. *Hans Bekič: Programming Languages and Their Definition*, volume 177 of *Lecture Notes in Computer Science*, chapter Definable operations in general algebras, and the theory of automata and flow charts. Springer, 1984.

[BK95] M. Bonsangue and M. Kwiatkowska. Re-interpreting the modal μ-calculus. In *Modal Logic and Process Algebra*, CSLI Lecture Notes, pages 65–83, 1995.

[BM93] D. Barnard and A. Mader. Model checking for the modal mu-calculus using Gauß elimination. Technischer Bericht 342/12/93 A, Technische Universität München, 1993.

[Boc70] I.M. Bocheński. *A History of Formal Logic*. Chelsea Publishing Company, New York, second edition, 1970.

[Bra92] J. C. Bradfield. *Verifying Temporal Properties of Systems*. Birkhäuser, 1992.

[Bra96] J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. In *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 1996.

[Bri96] E. Brinksma. personal communication. 1996.

[Bry86] R. E. Bryant. Graph based algorithms for Boolean function manipulation. *IEE Transactions on Computers*, C-35(8):677–691, 1986.

[BS90] J. C. Bradfield and C. Stirling. Verifying temporal properties of processes. In *Proceedings of CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 115–125. Springer, 1990.

[BS91] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 1991.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Proceedings of CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 1994.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[CKS92] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G. v. Bochmann and D.K. Probst, editors, *Proceedings of CAV'92*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422. Springer, 1992.

[Cle90] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27:725–747, 1990.

[CS91] R. Cleaveland and B. Steffen. A linear time model-checking algorithm for the alternation free modal mu-calculus. *Proceedings of the Third Workshop on Computer Aided Verification*, 2:79–92, July 1991.

[Dam92] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. Technical report, University of Edinburgh, June 1992.

[DP90] B. Davey and H. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1990.

[EH86] E.A. Emerson and J. Halpern. "sometimes" and "not never" revisited: On branching versus linear time. *Journal of the ACM*, 33:151–178, 1986.

[EJ88] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the 29th IEEE FOCS*, pages 328–337, 1988.

[EJ91] E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd FOCS*, pages 368–377, 1991.

[EJS93] E. Emerson, C. Jutla, and A. Sistla. On model checking for fragments of μ-calculus. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 1993.

[EL86] A. Emerson and C. Lei. Efficient model checking in fragments of the propositional mu-calculus. *Proceedings of 1st Annual Symposium on Logic in Computer Science, LICS'86*, pages 267–278, 1986.

[Eme91] E.A. Emerson. Temporal and modal logic. In J. van Leuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier / North-Holland, 1991.

[Eme96] E. Emerson. *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*, chapter Automated Temporal Reasoning about Reactive Systems, pages 41–101. Springer, 1996.

[EN94] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *J. Inform. Process. Cybernet.*, 30(3):143–160, 1994.

[Flo67] R. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Mathematical Aspects of Computer Science*, pages 19–32. American Mathematical Society, 1967.

[FR79] M.J. Fischer and Ladner R.E. Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, 18:194–211, 1979.

[Har95] C. Hartonas. Stone duality for modal μ-logics. 1995.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–162, 1985.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communication of the ACM*, 12:576–580, 1969.

[Hüt90] H. Hüttel. *SnS* can be modally characterized. *Theoretical Computer Science*, 74:239–248, 1990.

[Kai96] R. Kaivola. *Using automata to characterise fixed point temporal logics.* PhD thesis, University of Edinburgh, 1996.

[Kal96] K. Kalorkoti. Model checking in the modal μ-calculus by substitutions. 1996. submitted for publication.

[KM] E. Kindler and A. Mader. Trapping fairness. to appear.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Koz88] D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47:233–241, 1988.

[KP83] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Second Workshop on Logics of Programs*, 1983.

[KW97] E. Kindler and R. Walter. Mutex needs fairness. *Information Processing Letters*, 62(31–39), 1997.

[Lar92] K. Larsen. Efficient local correctness checking. In *Proceedings of CAV'92*, volume 663 of *Lecture Notes in Computer Science*. Springer, 1992.

[Lar95] K.G. Larsen. Proof system for Hennessy–Milner logic with recursion. In *Proceedings of CAAP'88*, volume 299 of *Lecture Notes in Computer Science*, pages 215–230, 1995.

[LBC$^+$94] D. Long, A. Browne, E. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In *Proceedings of 6th International Conference of Computer-Aided Verification, CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 338–350, 1994.

[Len96] G. Lenzi. A hierarchy theorem for the μ-calculus. In *Proceedings of ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 1996.

[Lin88] P. Lindsay. On alternating ω-automata. *Journal of Computer and System Sciences*, 36:16–24, 1988.

[LNS82] J.-L. Lassez, V.L. Nguyen, and E.A. Sonenberg. Fixed point theorems and semantics: a folk tale. *Information Processing Letters*, 14(3):112–116, May 1982.

[Mad92] A. Mader. Tableau recycling. In *Proceedings of CAV'92*, volume 663 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 1992.

[Mad95] A. Mader. Modal μ-calculus, model checking and Gauß elimination. In *Proceedings of TACAS'95*, volume 1019 of *Lecture Notes in Computer Science*, pages 72–88. Springer, 1995.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MP69] Z. Manna and A. Pnueli. Formalization of properties of recursively defined functions. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 201–210, 1969.

[MP83] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *Proceedings of the 10th ACM on Principles of Programming Languages*, pages 141–154, 1983.

[Niw86] D. Niwiǹski. On fixed point clones. In *Proceedings of the 13th ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 402–409. Springer, 1986.

[Niw88] D. Niwiński. Fixed-points vs. infinite generation. In *Proceedings of the third IEEE Symposium on Logic in Computer Science*, pages 402–409, 1988.

[Par70] D. M. R. Park. Fixpoint induction and proof of program semantics. *Machine Intelligence*, 5:59–78, 1970.

[Pra76] V. Pratt. Semantical considerations of Floyd-Hoare logic. In *Proceedings of the 1st IEEE Symposium on Foundations of Computer Science*, pages 109–121, 1976.

[Ros96] P. Rossmanith. personal communication. 1996.

[Rud74] S. Rudeanu. *Boolean Functions and Equations.* North-Holland Publishing Company, 1974.

[SE84] R.S. Streett and E.A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1984.

[Sti93] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 447–463. Oxford University Press, 1993.

[Sti96] C. Stirling. Model checking and other games. Notes for mathfit workshop on finite model theory, University of Wales, Swansea, 1996.

[Str81] R.S. Street. Propositional dynamic logic of looping and converse. In *Proceedings 13th Symposium on Theory of Computing*, pages 375–383, 1981.

[Str82] R.S. Street. Propositional dynamic logic of looping and converse is elementary decidable. *Information and Control*, 54:121–141, 1982.

[SW89] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In J. Díaz and F. Orejas, editors, *Proceedings of TAPSOFT*, volume 351 of *Lecture Notes in Computer Science*, pages 369–383, 1989.

[Tar55] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tho90] W. Thomas. *Handbook of Theoretical Computer Science*, volume 2, chapter Automata on infinite objects, pages 133–191. Elsevier/North-Holland, 1990.

[Var95] M.Y. Vardi. *Computer Science Today. Recent Trends and Developments.*, volume 1000 of *Lecture Notes in Computer Science*, chapter Alternating automata and program verification, pages 471–484. Springer, 1995.

[Ver95] B. Vergauwen. manuscript. 1995.

[VL92] B. Vergauwen and J. Lewi. A linear algorithm for solving fixed-point equations on transition systems. In J.-C. Raoult, editor, *Proceedings of 17th Colloquium on Trees in Algebra and Programming, CAAP'92*, volume 581 of *Lecture Notes in Computer Science*, pages 322–341. Springer, 1992.

[VL94] B. Vergauwen and J. Lewi. Efficient local correctness checking for single and alternating boolean equation systems. In *Proceedings of ICALP'94*, volume 820 of *Lecture Notes in Artificial Intelligence*, pages 302–315. Springer, 1994.

[VLAP94] B. Vergauwen, J. Lewi, I. Avau, and A. Poté. Efficient computation of nested fix-points with applications to model checking. In D. Gabbay and H.J. Ohlbach, editors, *Proceedings of ICTL'94*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 165–179. Springer, 1994.

[Vog96] W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. Technical report, Universität Augsburg, 1996.

[VW83] M. Vardi and P. Wolper. Yet another process logic. In *Proceedings of the Workshop on Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 501–512. Springer, 1983.

[Wal91] D. Walker. Automated analysis of mutual exclusion algorithms using CCS. Technical Report ECS-LFCS-89-91, University of Edinburgh, 1991.

[Wal93] F. Wallner. Ein lokaler model checker mit Gauß-Elimination. Fortgeschrittenenpraktikum, 1993.

[Wal94] F. Wallner. Model Checking im Modalen μ-Kalkül für unendliche Systeme mit Hilfe symbolischer Gauß-Elimination. Master's thesis, TU München, 1994. Diplomarbeit.

[Wal95a] R. Walter. *Petrinetzmodelle verteilter Algorithmen*, volume 2 of *Edition Versal*. Bertz Verlag, 1995. Dissertation.

[Wal95b] I. Walukiewicz. Completeness of Kozen's axiomatization of the propositional μ-calculus. In *Proceedings of LICS'95*, 1995.

[Win89] G. Winskel. A note on model checking the modal ν-calculus. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings of 16th ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772, 1989.

[ZSS94] S. Zhang, O. Sokolsky, and S.A. Smolka. On the parallel complexity of model checking in the modal mu-calculus. In *Proceedings of the 9th IEEE Symposium on Logic in Computer Science*, pages 154–163, 1994.