Theoretical Computer Science 126 (1994) 3–30
Elsevier

Model checking and boolean graphs*

Henrik Reif Andersen

Department of Computer Science, Building 344, Technical University of Denmark, DK-2800 Lyngby, Denmark

Abstract

Andersen, H.R., Model checking and boolean graphs, Theoretical Computer Science 126 (1994) 3–30.

We describe a method for translating a satisfaction problem of the modal μ-calculus into a problem of finding a certain marking of a boolean graph. By giving algorithms to solve the graph problem, we present a global model checking algorithm for a subset of the modal μ-calculus, which has time-complexity $O(|A||T|)$, where $|A|$ is the size of the assertion and $|T|$ is the size of the model (a labelled transition system). This algorithm is extended to an algorithm for the full modal μ-calculus running in time $O(|A|^{ad}|S|^{ad-1}|T|)$, where *ad* is the alternation depth and $|S|$ is the number of states in the transition system, improving on earlier presented algorithms. Moreover, a local algorithm is presented for alternation depth one. This algorithm runs in time $O(|A||T|\log(|A||T|))$ and is also an improvement over earlier algorithms.

1. Introduction

Model checking is the problem of deciding whether a given structure constitutes a valid model for a logical assertion. Viewing the structure as describing a system of, for example, interacting processes and the logical assertion as a specification, model checking can be viewed as the process of verifying that a system meets its specification. We will use a generalization of the modal μ-calculus presented by Kozen [15] as the assertion language and as models we take labelled transition systems (essentially

Correspondence to: H.R. Andersen, Department of Computer Science, Building 344, Technical University of Denmark, DK-2800 Lyngby, Denmark. Email: hra@id.dth.dk.

*This work is supported by the Danish Natural Science Research Council and the Danish Research Academy. An extended abstract with the same title appeared in Proc. of 4th European Symp. on Programming, ESOP'92, B. Krieg-Brückner, ed., Lecture Notes in Computer Science, Vol. 582 (Springer, Berlin, 1992). The main part of this work was carried out at the Computer Science Department, Aarhus University and while visiting the Computer Laboratory, University of Cambridge.

0304-3975/94/$07.00 © 1994—Elsevier Science B.V. All rights reserved
SSDI 0304-3975(93)E0121-J

equivalent to labelled Kripke models). The modal μ-calculus is a very expressive modal logic (see, e.g. [15, 14, 12]) allowing a wide range of properties to be expressed, including what is often called liveness, safety and fairness properties. Examples of such expressible properties are "eventually an a-action will happen", "it is always possible to do a b-action" and "infinitely often a c-action can happen". Labelled transition systems arise naturally in, for example, the operational semantics of process algebras as describing the behaviour of communicating concurrent systems.

This paper presents four results. First, it shows that the problem of finding the set of states in a finite labelled transition system satisfying a given formula with just one fixed-point operator, can be reduced to the problem of finding a fixed point of a monotone function on a boolean lattice consisting of a product of simple two-point lattices. Secondly, it is shown how this fixed point can be found in linear time using a simple graph algorithm, thereby giving an $O(|A||T|)$ model checking algorithm. Thirdly, this algorithm will be extended to the full calculus, giving an algorithm running in time $O(|A|^{ad}|S|^{ad-1}|T|)$, ad being the alternation depth of A – a measure of how intertwined minimum and maximum fixed points are. Finally, a local algorithm, searching potentially only a part of the transition system, will be presented for the modal μ-calculus of alternation depth one. This algorithm will run in time $O(|A||T|\log(|A||T|))$.

Previous work can be found in Emerson and Lei [14] which describes a global $O((|A||T|)^{ad+1})$ algorithm and defines the notion of alternation depth, and in Arnold and Crubille [6] which describes a global $O(|A|^2|T|)$ algorithm for the case of one simultaneous fixed point. Independently, Cleaveland and Steffen [10] and Vergauwen and Lewi [23] discovered a global algorithm for alternation depth one running in time $O(|A||T|)$, which, apart from differences in way of presentation, is very similar to the global algorithm presented in this paper.

Local algorithms have been described by Larsen [16], Stirling and Walker [21], Cleaveland [8] and Winskel [24], but they all have at least exponential running time – even for alternation depth one. Larsen [17] has recently improved his local algorithm (for one fixed point) to run in polynomial time. However, it is still not as efficient as the algorithm presented in this paper.

The local as well as the global algorithm will be given in an Algol-like imperative language. We use an imperative language because complexity of operations in imperative languages are well-understood, which makes complexity arguments simpler and more convincing. For the same reason the level of detail is very high compared to the previously mentioned papers on local model checkers (which do not contain complexity analyses).

2. Logic and models

We will consider a version of the modal μ-calculus with simultaneous fixed points. The expressive power will be equivalent to the modal μ-calculus with just unary fixed

points, in the sense that every assertion in our calculus has a logical equivalent containing only unary fixed points. The simultaneous fixed points will, however, be central to the development of efficient model checking algorithms as they allow one to express *sharing* of subexpressions.

The version of the modal μ-calculus we will use is given by the following grammar:

$$A ::= F \mid T \mid A_0 \vee A_1 \mid A_0 \wedge A_1 \mid \langle\alpha\rangle A \mid [\alpha]A \mid X \mid (\mu \underline{X}.\underline{A})_i \mid (\nu \underline{X}.\underline{A})_i.$$

The assertion variable X ranges over a set of variables *Var*. The usual notions of free variables and open and closed assertions will be used. The notation $\underline{X}$ is shorthand for $(X_1, \ldots, X_n)$, $\underline{A}$ for $(A_1, \ldots, A_n)$, where n should be clear from context. The assertion $(\mu \underline{X}.\underline{A})_i$ will denote the ith component of the simultaneous minimum fixed point $\mu \underline{X}.\underline{A}$. Dually, $(\nu \underline{X}.\underline{A})_i$ denotes the ith component of the maximum fixed point $\nu \underline{X}.\underline{A}$. The usual unary fixed point $\mu X.A$ corresponds to the case where $n=1$, and for notational convenience we simply write $\mu X.A$ instead of $(\mu X.A)_1$.

As models, we take *labelled transition systems* $T=(S, L, \rightarrow)$, where S is a set of states, L a set of labels, and $\rightarrow \subseteq S \times L \times S$ a transition relation. We write $(s, \alpha, s') \in \rightarrow$ as $s \xrightarrow{\alpha} s'$. Given a transition system T, an assertion A will denote a subset of the states S of T. Recall that the set of subsets ordered by inclusion $(\mathscr{P}(S), \subseteq)$ forms a complete lattice, which, by taking pointwise ordering, extends to a complete lattice $(\mathscr{P}(S)^n, \subseteq^n)$ on the n-ary product of $\mathscr{P}(S)$. Let $\pi_i : \mathscr{P}(S)^n \rightarrow \mathscr{P}(S)$ denote the projection onto the ith component.

For the fixed points, recall that a map ψ on $\mathscr{P}(S)^n$ is *monotonic* if, for all $\underline{U} \subseteq^n \underline{V}$, we have $\psi(\underline{U}) \subseteq^n \psi(\underline{V})$. According to Tarski's theorem [22], any monotonic ψ will have a minimum prefixed point given by

$$\bigcap \{\underline{U} \in \mathscr{P}(S)^n \mid \psi(\underline{U}) \subseteq^n \underline{U}\},$$

which we denote $\mu\psi$. Similarly, ψ will have a maximum postfixed point $\nu\psi$ given by

$$\bigcup \{\underline{U} \in \mathscr{P}(S)^n \mid \underline{U} \subseteq^n \psi(\underline{U})\}.$$

Due to the possibility of free variables, the interpretation of assertions will be given relative to an environment ρ assigning a subset of S to each variable. We will use $\rho[\underline{U}/\underline{X}]$ to denote the environment which is like ρ except that X_i is mapped to U_i. The interpretation of A denoted $[\![A]\!]_T\rho$ is defined inductively on the structure of A as follows:

$$[\![F]\!]_T\rho = \emptyset,$$

$$[\![T]\!]_T\rho = S,$$

$$[\![A_0 \vee A_1]\!]_T\rho = [\![A_0]\!]_T\rho \cup [\![A_1]\!]_T\rho,$$

$$[\![A_0 \wedge A_1]\!]_T\rho = [\![A_0]\!]_T\rho \cap [\![A_1]\!]_T\rho,$$

$$[\![\langle\alpha\rangle A]\!]_T = \{s \in S \mid \exists s' \in S.\ s \xrightarrow{\alpha} s' \ \&\ s' \in [\![A]\!]_T \rho\},$$

$$[\![[\alpha] A]\!]_T = \{s \in S \mid \forall s' \in S.\ s \xrightarrow{\alpha} s' \Rightarrow s' \in [\![A]\!]_T \rho\},$$

$$[\![X]\!]_T \rho = \rho(X),$$

$$[\![(\mu \underline{X}.\underline{A})_i]\!]_T \rho = \pi_i(\mu\psi), \quad \text{where } \psi : (U_1, \ldots, U_n) \mapsto ([\![A_1]\!]_T \rho', \ldots, [\![A_n]\!]_T \rho'), \text{ and } \rho' = \rho[U_1/X_1, \ldots, U_n/X_n],$$

$$[\![(\nu \underline{X}.\underline{A})_i]\!]_T \rho = \pi_i(\nu\psi), \quad \text{where } \psi \text{ is as above.}$$

Given a transition system $T = (S, L, \rightarrow)$, we will say that a state $s \in S$ *satisfies* the assertion A if $s \in [\![A]\!]_T \rho$ for all environments ρ and write $s \models_T A$.

For the rest of this section, we will concentrate on *unnested* fixed points and finite transition systems, and we will describe how to transform the problem of satisfaction into a problem of finding a marking of a particular kind of graph. The transformation proceeds in two steps: First the unnested fixed point is transformed into an equivalent *simple* fixed point. Secondly, this fixed point is transformed into a modality-free fixed point from which we eventually construct a boolean graph.

We will say that $\mu \underline{X}.\underline{A}$ is an *unnested fixed-point assertion* if no fixed points appear in the body $\underline{A}$. Furthermore, we will say that an unnested fixed point $\mu \underline{X}.\underline{A}$ is *simple* if each of the components A_j of $\underline{A}$ contains at most one operator, i.e. A_j is of one of the forms

$$F,\ T,\ X_{j_0} \vee X_{j_1},\ X_{j_0} \wedge X_{j_1},\ \langle\alpha\rangle X_{j'},\ [\alpha] X_{j'},\ X_{j'}.$$

Any unary, unnested fixed-point assertion $\mu X.A$ can be translated into an equivalent simultaneous, simple fixed-point assertion, with n variables, where $n = |A|$ is the size of A, measured as the number of operators and variables. The translation proceeds as follows: To each subexpression, we associate a variable. This gives n variables $\{X_1, \ldots, X_n\}$. Define the n-ary fixed point $\mu \underline{X}.\underline{A}$ by

$$A_i = \text{the expression associated with } X_i \text{ where all proper subexpressions are replaced by their associated variables and } X \text{ is replaced by } X_1,$$

assuming that X_1 is associated with A. Using Bekić's theorem [7] relating simultaneous and unary fixed points, one can now show the following proposition.

Proposition 2.1. *Let $\mu X.A$ be a closed, unary, unnested fixed point and let $\mu \underline{X}.\underline{A}$ be the translated simple fixed point. Then*

$$[\![(\mu \underline{X}.\underline{A})_1]\!]_T \rho = [\![\mu X.A]\!]_T \rho$$

for all environments ρ.

As an example $\mu X.[\alpha]X \wedge \langle\beta\rangle T$ will give rise to the 4-ary simple fixed point

$$\mu\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix}.\begin{pmatrix} X_2 \wedge X_3 \\ [\alpha]X_1 \\ \langle\beta\rangle X_4 \\ T \end{pmatrix}.$$

The translation and Proposition 2.1 generalises easily to unnested fixed points of arbitrary arity. The number of variables of the resulting simple fixed point will still be equal to the size of the original fixed-point assertion.

We now proceed to the second part of the translation. Given a transition system T and an assertion A, we will, for each state s, describe a method of finding an assertion A/s without modalities, which intuitively (when ignoring variables) has the property that $s \models_T A$ if and only if A/s denotes true.[1] In order to state this formally, we will interpret assertions without modalities – assertions built from the propositional fragment of our calculus – over the trivial one-state transition system $\bullet = (\{\bullet\}, \emptyset, \emptyset)$ with no transitions. Hence, every closed assertion A will either denote $\{\bullet\}$ or $\emptyset$ of the complete two-point lattice $\mathscr{P}(\{\bullet\})$. The lattice $\mathscr{P}(\{\bullet\})$ is nothing else but a distinct copy of the well-known two-point lattice, $\mathbb{O} = \{0, 1\}$ with the partial ordering $0 \leqslant 0$, $0 \leqslant 1$, $1 \leqslant 1$, also known as Sierpinski space. Hence, we shall often use 0 and 1 instead of $\emptyset$ and $\{\bullet\}$.

Assume that the set of states of T is numbered such that $S = \{s_1, \ldots, s_n\}$. Observe that the Sierpinski space $\mathbb{O}$ extends to a complete lattice $\mathbb{O}^n$ by extending the ordering pointwise, and note that there is an obvious isomorphism on lattices *in*: $\mathbb{O}^n \cong \mathscr{P}(S)$ defined by $in(x_1, \ldots, x_n) = \{s_i \in S \mid x_i = 1\}$.

Now, to be precise: Given a closed assertion A, we will find modality-free assertions $(A/s_1, \ldots, A/s_n)$ such that

$$[\![A]\!]_T \rho = in([\![A/s_1]\!]_\bullet \rho, \ldots, [\![A/s_n]\!]_\bullet \rho)$$

for all environments ρ. Having found such assertions, we have by the definition of the *in*-map that $s_j \in [\![A]\!]_T \rho$ if and only if $[\![A/s_j]\!]_\bullet \rho = 1$; hence, we have found the wanted modality-free assertions.

We will define A/s_i by structural induction on A, so due to the fixed points we will be confronted with open assertions. In order to handle these open assertions, we will need a notion of *change of variables* which will relate the variables of A to the variables of the A/s_i's. Consider an assertion A with variables $\{X^1, \ldots, X^m\}$ and assume that to each variable X^i, σ associates a new set of variables $\sigma(X^i) = (Y^i_1, \ldots, Y^i_n)$ such that there are no name-clashes between any of the new and any of the old variables. Say that a pair of environments (ρ, ρ') is *appropriate* for *in* and the change of variables σ if

[1] Larsen and Xinxin [20] describe a similar translation.

$\rho: Var \to \mathscr{P}(S)$ and $\rho': Var \to \mathbb{O}$, and

$$\rho(X) = in(\rho'(Y_1), \ldots, \rho'(Y_n))$$

for all variables X with $\sigma(X) = (Y_1, \ldots, Y_n)$. For two such appropriate environments, assume inductively that we have found modality-free assertions A/s_i corresponding to A such that

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

where $\psi(U) = [\![A]\!]_T \rho[U/X]$ and

$$\theta(U_1, \ldots, U_n)$$
$$= ([\![A/s_1]\!]_\bullet \rho'[U_1/Y_1, \ldots, U_n/Y_n], \ldots, [\![A/s_n]\!]_\bullet \rho'[U_1/Y_1, \ldots, U_n/Y_n]).$$

As in is an isomorphism of lattices, we immediately get that $\mu\psi = in(\mu\theta)$. (Alternatively, the reduction lemma from [5] could be applied.) Hence,

$$[\![\mu X.A]\!]_T \rho = in([\![(\mu \underline{Y}.\underline{A})_1]\!]_\bullet \rho', \ldots, [\![(\mu \underline{Y}.\underline{A})_n]\!]_\bullet \rho'),$$

where $\underline{A} = (A/s_1, \ldots, A/s_n)$, and we have found modality-free assertions corresponding to $\mu X.A$.

We are now able to state the full definition of A/s. Define for each state s, the quotient A/s by structural induction on A as follows:

$$F/s = F,$$

$$T/s = T,$$

$$(A_0 \vee A_1)/s = (A_0/s) \vee (A_1/s),$$

$$(A_0 \wedge A_1)/s = (A_0/s) \wedge (A_1/s),$$

$$([\alpha]A)/s = \bigwedge \{A/s' \mid s \xrightarrow{\alpha} s'\},$$

$$(\langle\alpha\rangle A)/s = \bigvee \{A/s' \mid s \xrightarrow{\alpha} s'\},$$

$$X/s = Y_i, \quad \text{where } s = s_i, \sigma(X) = (Y_1, \ldots, Y_n).$$

For the k-ary fixed-point $\mu X.\underline{A}$, assume that we have a change of variables σ with $\sigma(X_j) = (Y_1^j, \ldots, Y_n^j)$, and let the nk-ary fixed point $\mu \underline{Y}.\underline{B}$, $\underline{Y} = (Y_1^1, \ldots, Y_i^j, \ldots, Y_n^k)$, $\underline{B} = (B_1^1, \ldots, B_i^j, \ldots, B_n^k)$ be defined by

$$B_i^j = A_j/s_i,$$

where $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant k$. Take

$$(\mu \underline{X}.\underline{A})_j/s_i = (\mu \underline{Y}.\underline{B})_i^j,$$

and similarly for the maximum fixed point.[2] Now, we have the following theorem.

Theorem 2.2 (Quotienting theorem). *For an arbitrary assertion A, change of variables σ and appropriate pair of environments (ρ, ρ'), we have*

$$[\![A]\!]_T \rho = in([\![A/s_1]\!]_\bullet \rho', \ldots, [\![A/s_n]\!]_\bullet \rho').$$

The original problem of deciding whether a particular state s satisfies the closed assertion A can now be recast by applying the quotienting theorem:

$$\begin{aligned} s \models_T A \quad &\text{iff} \quad s \in [\![A]\!]_T \rho \text{ for all } \rho \\ &\text{iff} \quad s \in in([\![A/s_1]\!]_\bullet \rho', \ldots, [\![A/s_n]\!]_\bullet \rho') \\ &\text{iff} \quad [\![A/s]\!]_\bullet \rho' = 1, \end{aligned}$$

where (ρ, ρ') is appropriate for σ and in. In other words, model checking can be reduced to deciding whether the assertion A/s denotes the top element of $\mathbb{O}$. An important point about the quotienting is that the resulting assertion consists entirely of disjunctions, conjunctions, variables and fixed-point operators (viewing F and T as empty disjunctions and conjunctions). In particular, for an unnested k-ary fixed-point $\mu \underline{X}.\underline{A}$, we end up with an unnested fixed point $\mu \underline{Y}.\underline{B}$ in the lattice $\mathbb{O}^{k|S|}$. Moreover, if $\mu \underline{X}.\underline{A}$ is *simple* (hence, $k = |A|$), the total size of $\mu \underline{Y}.\underline{B}$ will be bounded by $|A||T|$, where $|T| = |S| + |L| + |\rightarrow|$, as simple calculations show:

$$\begin{aligned} |B| &= \sum_{j=1}^{|S|} \sum_{i=1}^{k} |B_j^i| \\ &= \sum_{i=1}^{k} \sum_{j=1}^{|S|} |A_i/s_j| \\ &\leqslant \sum_{i=1}^{k} \sum_{j=1}^{|S|} \max(1, |\{ j' \mid \exists \alpha.\ s_j \xrightarrow{\alpha} s_{j'} \}|) \\ &\leqslant \sum_{i=1}^{k} |T| = k|T| = |A||T|. \end{aligned}$$

If $\mu \underline{X}.\underline{A}$ is *not* simple, this bound would not hold. As an example consider the assertion $\mu X.\langle\alpha\rangle[\alpha] \ldots \langle\alpha\rangle[\alpha]X$ (l modalities), and assume that T is a transition system with n states, all connected to each other by α-transitions. Then the size of a single right-hand side of the resulting assertion will be:

$$|\langle\alpha\rangle[\alpha] \ldots \langle\alpha\rangle[\alpha]X/s_j| = |\bigvee_{i_1} \bigwedge_{i_2} \cdots \bigwedge_{i_{l-1}} \bigvee_{i_l} X_{i_l}| = n^l,$$

where all indices range over all states. The significance of making the fixed points simple is that values of subexpressions are shared across the disjunctions and conjunctions. In this example, we will get a resulting assertion of size ln^2 – and not n^l.

In the analysis of time and space complexities we will make use of some general assumptions about the representations of assertions and transition systems. First,

[2] We use double indexing as a convenient shorthand although the syntax formally only allows simple indexing.

variables will be assumed to be represented by natural numbers, which in turn will be assumed to be representable in a constant amount of memory.[3] Secondly, functions from an interval of the natural numbers to a set of "simple" values, e.g. numbers, will be represented efficiently such that access to the value at one particular element in the domain can be performed in constant time (like "arrays" in many programming languages). Thirdly, the transition relations are represented as functions from the set of states (assumed to be an interval of natural numbers) into sets of pairs consisting of a label and a state. Labels are also assumed to be represented in a constant amount of memory.

Often, we will use statements like, this algorithm runs "in time and space $K(n)$", where it actually should be "in time and space asymptotically bounded by $K(n)$". We will also use the notation $O(K(n))$ for this statement. All these assumptions and slight abuses of language are standard when analysing complexities of algorithms.

With these assumptions, it is easy to see that the translations into simple fixed points and boolean graphs can be performed in time and space $O(|A||T|)$.

3. Boolean graphs

In Section 2, we described how to transform an unnested fixed point $\mu X.A$ (of arity 1 or more) into first a simple k-ary fixed-point $\mu \underline{X}.\underline{A}$ and then, given a transition system with n states, into an nk-ary fixed-point $\mu \underline{Y}.\underline{B}$ consisting of only conjunctions and disjunctions. By these transformations, we have reduced the problem of finding a fixed point over the lattice $\mathscr{P}(S)$ to a problem of finding a fixed point of a boolean function over the lattice $\mathbb{O}^{nk}$. Viewing the variables as vertices of a graph and the dependencies between variables as directed edges, $\mu \underline{Y}.\underline{B}$ defines a directed boolean graph, which essentially is nothing else but another representation of the function of $\underline{Y}$ defined by $\underline{B}$.

Formally, a *boolean graph* G is a triple (V, E, L) where V is a set of vertices, $E \subseteq V \times V$ a set of directed edges, and $L: V \to \{\vee, \wedge\}$ is a total function labelling the vertices as disjunctive or conjunctive. The set $S(v)$ of *successors* and the set $P(v)$ of *predecessors* of a vertex v are defined by $S(v) = \{w \mid (v, w) \in E\}$ and $P(v) = \{w \mid (w, v) \in E\}$. Given a simple k-ary fixed point $\mu \underline{Y}.\underline{B}$ consisting of disjunctions and conjunctions, we can define a graph $G_{\underline{B}} = (V, E, L)$ where

$$V = \{Y_i \mid 1 \leqslant i \leqslant k\},$$

$$E = \{(Y_i, Y_j) \mid (B_i = \bigvee_{l \in I} Y_l \text{ or } B_i = \bigwedge_{l \in I} Y_l) \,\&\, j \in I\},$$

[3] As usual in complexity analysis we make the assumptions that integers can be stored in a constant amount of memory and that an arbitrary memory address can be accessed in constant time, the "uniform cost criterion", cf. Aho et al. [1].

$$L(Y_i)=\begin{cases}\vee & \text{if } B_i=\bigwedge_{l\in I} Y_l,\\ \wedge & \text{if } B_i=\bigvee_{l\in I} Y_l.\end{cases}$$

Note that there is an edge from i to j iff Y_j is one of the disjuncts/conjuncts in B_i, expressing the fact that the value of Y_i "depends" on the value of Y_j.

A *marking* of a boolean graph G is a function $m: V \to \mathbb{O}$ assigning values 0 and 1 to the vertices. The graph G induces a function g taking a marking m to a new marking $g(m)$ which is "what can be computed from m", i.e. for a marking m define the marking $g(m)$ as

$$g(m)(v)=\begin{cases}1 & \text{if } L(v)=\wedge \ \ \& \ \ \forall w\in S(v).m(w)=1\\ & \text{or } L(v)=\vee \ \ \& \ \ \exists w\in S(v).m(w)=1,\\ 0 & \text{otherwise.}\end{cases}$$

When G is constructed from a fixed point $\mu \underline{Y}.\underline{B}$, the function g is exactly the function defined by the body of the fixed point $\mu \underline{Y}.\underline{B}$, and m is nothing else but an element of $\mathbb{O}^n$; but thinking of m as a marking will be helpful in the development of the algorithms. As the set of markings $\mathbb{O}^V$ is just an isomorphic copy of $\mathbb{O}^n$, $\mathbb{O}^V$ will be a complete lattice when equipped with the same ordering as $\mathbb{O}^n$, i.e. the pointwise extension of the Sierpinski ordering. The problem we have to solve now is: Given a boolean graph G defining the monotonic map $g: \mathbb{O}^V \to \mathbb{O}^V$, what is the minimum prefixed point $\mu g \in \mathbb{O}^V$?

4. Algorithms

In this section, we will describe two algorithms for computing the minimum fixed point of a boolean graph. The first will be global in the sense that it computes the complete fixed point of the graph, and, on a graph G, it will have time and space complexity $O(|G|)$. If G is constructed from an unnested fixed-point formula $\mu X.A$ and a transition system T as described in the previous section, the size of G will be $O(|A||T|)$; hence, we have a global model checking algorithm that in the worst case is linear in the size of the assertion and linear in the size of the transition system.

The second will be local, in the sense that starting from a particular node x, it will only compute an approximation to the fixed point, and in doing so traverse only a necessary subset of the graph. The approximation will be correct on x and on all nodes visited. This algorithm will have, on a graph G, worst-case space complexity $O(|G|)$ and worst-case time complexity $O(G|\log|G|)$.

Both algorithms will be presented in the version for finding minimum fixed points, the case of maximum fixed points being completely dual.

4.1. A global algorithm

The global algorithm will start with the bottom element of the lattice $\mathbb{O}^V$ and gradually increase it until (eventually) the minimum fixed point is reached. Pictorially, one can think of the algorithm as "chasing ones" around the graph: Starting with nodes that are trivially forced to be one (conjunctive nodes with no successors), it will look for dependent nodes that are forced to be one, continuing until no further nodes can be forced to one – thereby having found the minimum fixed point.

Figure 1 describes the algorithm. The function $st: V \to \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers, denotes the "strength" of a node, i.e. the number of successors that must be one before this node will be forced to be one. The function g induced by G can be "extended" to a function on strengths by taking, for all $v \in V$,

$$\bar{g}(st)(v) = \begin{cases} |S(v) \cap st_{>0}| & \text{if } L(v) = \wedge, \\ 1 - |S(v) \cap st_{\leqslant 0}| & \text{if } L(v) = \vee, \end{cases}$$

where $st_{>0} = \{v \mid st(v) > 0\}$, i.e. the set of nodes which still needs some successors to become one, and $st_{\leqslant 0} = \{v \mid st(v) \leqslant 0\}$, i.e. the set of nodes which have enough successors that are one (the negative value indicates the "excess" of ones). A strength defines a marking $\hat{st}$ by

$$\hat{st}(v) = \begin{cases} 1 & \text{if } st(v) \leqslant 0, \\ 0 & \text{if } st(v) > 0. \end{cases}$$

It is now easy to see that $(\hat{}) \circ \bar{g} = g \circ (\hat{})$; hence, if $\bar{g}(st) = st$ then $g(\hat{st}) = \hat{st}$, implying that $\hat{st}$ is a fixed point of g.

The set A denotes an "active" set of nodes marked with ones, for which the consequences of becoming one has not yet been computed. Correctness can be shown

---

Input: Boolean graph $G = (V, E, L)$, defining the function g.
Output: A marking $m: V \to \{0, 1\}$ equal to μg.

for all $v \in V$ **do** $st(v) := \begin{cases} |S(v)| & \textbf{if } L(v) = \wedge \\ 1 & \textbf{if } L(v) = \vee \end{cases}$
$A := st_{\leqslant 0}$
while $A \neq \emptyset$ **do**
  choose some $v \in A$; $A := A \setminus \{v\}$
  for all $w \in P(v)$ **do**
    $st(w) := st(w) - 1$
    if $st(w) = 0$ **then** $A := A \cup \{w\}$
$m := \hat{st}$

Fig. 1. A global algorithm: Chasing 1's.

from the invariant I:

$$I \Leftrightarrow_{\text{def}} A \subseteq st_{\leqslant 0} \;\&$$

$$\hat{st} \leqslant \mu g \;\&$$

$$\forall v \in V.\ st(v) = \begin{cases} |S(v) \cap (st_{>0} \cup A)| & \text{if } L(v) = \wedge \\ 1 - |S(v) \cap (st_{\leqslant 0} \setminus A)| & \text{if } L(v) = \vee . \end{cases}$$

Theorem 4.1. *The algorithm of Fig.* 1 *correctly computes the minimum fixed point* μg *and it can be implemented to run in time* $O(|G|)$.

Proof. It is a simple exercise to show that the invariant I holds immediately before the while-loop, and that it is preserved by the body. When the while-loop terminates we have $A = \emptyset$ which from the invariant implies that $st = \bar{g}(st)$ and $\hat{st}$ is a fixed point, which by the second conjunct of the invariant is less than or equal to the minimum fixed point; hence, $\hat{st} = \mu g$.

For the time complexity, first note that whenever a node has been removed from the set A, it will never be inserted again as this only happens when its strength equals zero, and strengths always decrease. Hence, the body of the while-loop will be executed at most once for each node v of the graph. Each execution of the innermost for-all-loop takes time proportional to the size of $P(v)$, i.e. the number of predecessors for the node v. In total, the while-loop takes time proportional to the sum of the number of predecessors, i.e. the total number of edges in G, and is thus bounded by $|G|$. The first loop and the last assignment are also bounded by $|G|$.

As the algorithm looks at predecessors of nodes, the graph must initially be reversed, which can easily be done in linear time (see, e.g. [11]). □

4.2. A local algorithm

Model checking is usually involved with deciding satisfaction for just one particular state, so it might seem overwhelming to have to compute the complete fixed point in order to decide the value at just one particular state. This observation is central to the development of *local* model checkers with the idea being that starting from one particular state, only a "necessary" part of the transition system will be investigated in order to determine satisfaction. Larsen [16] describes such an algorithm for the case of one fixed point, which in an improved version is used in the TAV system [18]. Stirling and Walker [21] and Cleaveland [8] describe a similar method for the full modal μ-calculus based on tableaux, which has been used in the implementation of the Concurrency Workbench [9]. Using a single key-property of maximum fixed points, Winskel, in [24], develops a very similar and quite simple model checker. Unfortunately, they all have very bad worst-case behaviours. Even for formulas with one fixed point, they have worst-case time complexity at least exponential in the number of transitions.

In this section, we present a local algorithm for finding a fixed point of a boolean graph, which will only visit a subset of the graph in the search for deciding the minimum fixed point value for one particular node. This will be done in time proportional to the size of the subset being visited; hence, in the worst case it will be $|G|$. Unfortunately, in the initialisation phase the algorithm will need to visit each node in the graph once, and the running-time will then always be linear as for the global algorithm. Nevertheless, the algorithm seems interesting as it works very differently from the global algorithm and still solves the same problem. Moreover, after presenting the algorithm, we discuss a way of using the algorithm in a slightly revised version avoiding the costly initialisation – as a basis for a local model checker, which will run in time $O(|B|\log|B|)$ where B is the subgraph being traversed. Thus, the worst-case behaviour will be $O(|A||T|\log(|A||T|))$, and we have a local model checker which in the worst case is only a logarithmic factor worse than the global model checker.

The local algorithm "Avoiding 1's" is presented in Figs. 2–4, and works as follows: Initially all nodes will be marked with a zero. We start with the node of interest, x say, and try to verify whether its minimum fixed point marking is really a zero (the task of the *visit* procedure). This involves inspecting the successors each in turn, finding their minimum fixed-point markings, until, in the case of a conjunctive node, a zero is found, or in the case of a disjunctive node, a one is found, or all successors have been

---

Input: Boolean graph $G=(V, E, L)$, and a node $x \in V$.
Output: A marking $m: V \to \{0, 1\}$ and a set $B \subseteq V$ with $x \in B$ such that
m equals μg on B.

Initialisation:
$B, A := \emptyset \quad m, p := \underline{0} \quad d := \underline{\emptyset}$
Method:
visit(x)
while $A \neq \emptyset$ **do**
choose some $y \in A$; $A := A \setminus \{y\}$
for all $w \in d(y)$ **do**
if $L(w) = \vee$ & $m(w) = 0$ **then**
$m(w) := 1$ $A := A \cup \{w\}$
ff $L(w) = \wedge$ **then**
$p(w) := p(w) + 1$
fwtn(w)
fi

Fig. 2. A local algorithm: Avoiding 1's.

```
visit(x)=
if x∉B then
  B := B∪{x}
  if L(x)= ∨ then
    ok :=false
    while p(x)<|S(x)| & ¬ok do
      w := S(x)_p(x)
      visit(w)
      if m(w)=0 then d(w) := d(w)∪{x}   p(x) := p(x)+1
      ff m(w)=1 then ok := true
      fi
    if ok then m(x) := 1  A := A∪{x}
  ff L(x)= ∧ then
    fwtn(x)
  fi
```

Fig. 3. Visit.

```
fwtn(x)=
ok :=false
while p(x)<|S(x)| & ¬ok do
  w := S(x)_p(x)
  visit(w)
  if m(w)=0 then d(w) := d(w)∪{x}   ok := true
  ff m(w)=1 then p(x) := p(x)+1
  fi
if ¬ok then m(x) := 1  A := A∪{x}
```

Fig. 4. Fwtn: "find a witness".

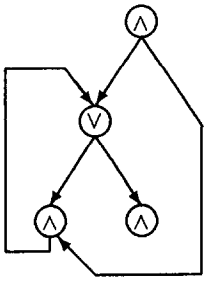
inspected. For this purpose, we assume that the successors of each node v have been numbered from 0 to $(|S(v)|-1)$, i.e. $S(v)=\{S(v)_0, \ldots, S(v)_{|S(v)|-1}\}$. The function $p: V \to \mathbb{N}$ is used in order to keep track of which successor $p(v)$ of v is being examined, or must be examined next.

Due to cycles in the graph, a node that at one point is found to be marked with zero, can later be changed into being marked with one; hence, all nodes that were assigned a marking based on this particular node being zero might have to be changed as well.

In order to be able to perform this updating efficiently, we keep for each node v a list of nodes $d(v)$ that should be informed in case the marking of v will change from zero to one. Thus, for each node v, $d: V \to \mathcal{P}(V)$ will denote a subset of its predecessors $P(v)$, and this set will grow as the algorithm proceeds.

Like in the global case, the set $A \subseteq V$ contains nodes v that have changed marking from zero to one, and for which this information has not yet been spread to the nodes in $d(v)$. The set $B \subseteq V$ contains all nodes that have been visited. The procedure *fwtn* (short for "find witness") will for a conjunctive node v search the successors starting from number $p(v)$ for one with a zero marking, that "witnesses" that v should have the marking zero. If no such exists, the node v will have to be marked with a one.

To get a feeling for the working of the algorithm, the reader is encouraged to try it on the small example below.



At any point in the execution of the algorithm, the situation will be as sketched in Fig. 5.

The complexity of the algorithm depends on how B, m, p and d are implemented. Implementing B as an V-indexed array of boolean flags, and also m, p and d as V-indexed arrays, the operations performed on these structures are all constant time, but the initialisations take time $O(|V|)$. This gives the following theorem. (The proof can be found in Appendix A.)

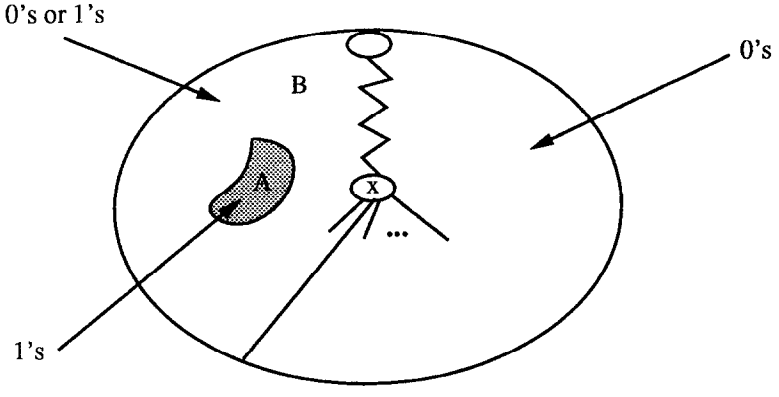


Fig. 5. A typical situation of avoiding 1's.

Theorem 4.2. *Given a boolean graph G with the induced function g. The algorithm described in Figs. 2–4 correctly computes an element m of $\mathbb{O}^V$ and a set $B \subseteq V$, such that*

$$m|_B = (\mu g)|_B,$$

and it can be implemented to run in time $O(|G|)$.

However, noting that m, p and d only need to be initialised on a node when visit is called the first time at that node, we can instead implement the set B as a balanced search tree associating nodes of the graph to their memory locations, yielding constant time initialisation and logarithmic (i.e. $\log(|B|)$) execution time when testing for membership in B and performing insertion of new nodes into B. Hence, we assume that each node x is associated with a memory address a_x on which the values of m, p and d at x will be stored. This memory address will be allocated and m, p and d initialised when *visit* is first called on x. Thus, we have the following sketch of an algorithm.

The initialisation is changed to:

$$B, A := \emptyset.$$

The procedure *visit* is changed to:

visit(x) = **if** $x \notin B$ **then**
allocate a new memory cell with address a_x
initialise d on x to $\emptyset$,
p on x to 0, m on x to 0
insert the pair (x, a_x) in B
find $S(x)$ by performing the division
if ...
... as before, where all accesses to m, d, etc. are through the addresses stored in B ...
fi

The procedure *fwtn* will not be changed, except that all access to the variables m, p, etc. will be through their addresses stored in B. The number of primitive steps performed by this algorithm will be as before, but we have to take into account the logarithmic factor coming from the searches in B. Hence, the running time will be $O(|B|\log|B|)$, which in the worst case is $O(|G|\log|G|)$.

It is important to observe that in order to fully exploit the possible benefits of only investigating a part of the boolean graph, the graph – and therefore the transition system – must be constructed in a demand-driven fashion from a given assertion and, for instance, a process algebraic term.

5. Extensions to the full modal μ-calculus

In this section, we will describe how the global algorithm Chasing 1's can be extended to yield a model checker for the full modal μ-calculus. Given an algorithm – like Chasing 1's – that can find a simultaneous unnested fixed point in time $O(|A||T|)$, we show that we can compute the set of states satisfying an arbitrary closed assertion A of alternation depth $ad(A)$ in time $O(|A|^k|S|^{k-1}|T|)$, where $k=\max\{ad(A), 1\}$. (The reader is referred to Appendix B for the definition of alternation depth.) As $|T|=|S|+|L|+|\rightarrow|$, the resulting algorithm will be linear in the number of transitions, although, when the alternation depth is unbounded, exponential in the size of the assertion and the number of states. When the alternation depth is bounded, it will yield a polynomial-time algorithm with a polynomial degree one less than the algorithm of Emerson and Lei [14].

Theorem 5.1. *Assume given an algorithm that can find a simultaneous unnested fixed point ($\mu\underline{X}.\underline{A}$ and $\nu\underline{X}.\underline{A}$) on a transition system T in time $O(|A||T|)$. There exists an algorithm that will compute the set of states denoted by an assertion A in time $O(|A|^k|S|^{k-1}|T|)$ and space $O(|A||T|)$, where $k=\max\{ad(A), 1\}$.*

The algorithm and the proof can be found in Appendix B.

This algorithm only assumes the presence of an efficient algorithm for handling the unnested case, and then by applying this at appropriate places handles the general case. Boolean graphs are not used, except perhaps in the base-case. Another attempt of extending the global algorithm to the full modal μ-calculus, could be through a generalisation of the boolean graphs. Assume that we are interested in computing the set denoted by the assertion A. First simplify all fixed points appearing in A, then perform the divison, and finally construct a boolean graph from the resulting modality-free assertion where the vertices are partitioned into disjoint sets $V_1, \ldots, V_n$ – one for each fixed point. Now, a certain marking of this partitioned graph, reflecting the minimum and maximum fixed points, would correspond to the element representing A. It is an interesting task to investigate to what extent this approach can lead to new algorithms.

However, an immediate application, which is described here, is in the generalisation of the local algorithm from one fixed point to alternation depth one.

The construction of a *partitioned boolean graph* proceeds as follows. Let an assertion A be given. If the top-most operator is not a minimum or maximum fixed point, change A into $\mu X.A$ for an arbitrary variable X (taking $\nu X.A$ would also do). Transform A into a normal form, where consecutive sequences of minimum (maximum) fixed points are replaced by one minimum (maximum) fixed point (as in the proof of Theorem 5.1 in Appendix B). Simplify all fixed points. Assume that all variables appearing in different fixed points of A are different, otherwise, rename the variables so that this is the case. Perform the division with respect to a change of variables σ and a labelled transition system $T=(S, L, \rightarrow)$.

Recall, that the top-most operator of A – and, therefore, of B – is a μ or a ν. Assume that $\{Y_1, \ldots, Y_m\}$ are the boolean variables in B and let B_i be the right-hand side corresponding to the variable Y_i. Define the boolean graph $G_B=(V, E, L)$ as follows:

$$V=\{Y_1, \ldots, Y_m\}$$

$$E=\left\{\begin{array}{l}(Y_i, Y_j) \mid (B_i=\bigvee M \text{ or } B_i=\bigwedge M) \,\&\, Y_j \in M \\ \qquad \text{or } B_i=(\mu \underline{Z}.\underline{C})_k \,\&\, Y_j \text{ is} \\ \text{the } k\text{th variable in } \underline{Z} \\ \qquad \text{or } B_i=(\nu \underline{Z}.\underline{C})_k \,\&\, Y_j \text{ is the } k\text{th variable in } \underline{Z}\end{array}\right\}$$

$$L(Y_i)=\begin{cases}\vee & \text{if } B_i=\bigvee M, \\ \wedge & \text{if } B_i=\bigwedge M \text{ or } B_i=(\mu \underline{Z}.\underline{C})_k \text{ or } B_i=(\nu \underline{Z}.\underline{C})_k.\end{cases}$$

Note that there is an edge from Y_i to Y_j if Y_j is one of the disjuncts/conjuncts in B_i, or if B_i is a projection of a fixed point and Y_j is the variable of the fixed point corresponding to that projection. The choice of assigning the label $\wedge$ to the variable Y_i in this last case is arbitrary; $\vee$ could have been chosen as well. From the definition of the quotienting, it is not difficult to see that the number of vertices of G_B is $O(|A||S|)$ and that the total size of G_B is $O(|A||T|)$.

Assume that the fixed points appearing in B are numbered from 1 to n, and let V_i be the variables in the ith fixed point. Let $\mathscr{G}_B=(G_B, \mathscr{V}, \mathscr{L})$ be defined by:

$$\mathscr{V}=\{V_1, \ldots, V_n\},$$

$$\mathscr{L}(V_i)=\begin{cases}\mu & \text{if the } i\text{th fixed point is a minimum one,} \\ \nu & \text{if the } i\text{th fixed point is a maximum one.}\end{cases}$$

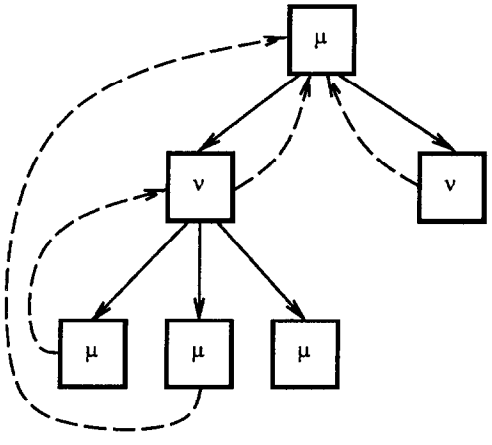


Fig. 6. A partitioned boolean graph. Boxes indicate partitionings, arrows indicate presence of one or more edges between partitionings.

We call $\mathcal{G}_B$ a *partitioned boolean graph.* Figure 6 shows an outline of a partitioned boolean graph. If A has alternation depth one, cycles in the graph will be of a special kind as expressed by the proposition below.

Proposition 5.2. *Let A be an assertion of alternation depth one. Then the partitioned boolean graph constructed from A as above will have the property that all cycles of the underlying boolean graph will consist of nodes that are all consistently labelled with only μ's or only ν's. Moreover, due to the transformation into normal form, they will all belong to the same component.*

Proof (*Sketch*). Assume that there exists a cycle with a node labelled μ and a node labelled ν. These nodes must belong to two different elements of the partitioning, V_i and V_j. Then either the ith fixed point contains the jth fixed point and the jth fixed point refers to a variable from the ith fixed point, or the other way around. In both cases, these fixed points could only come from an assertion of alternation depth at least two.

Assume that two nodes labelled μ belong to two different components of the partitioning. Then by arguments similar to the ones above, we would have a sequence of two minimum fixed points, which contradicts the fact that A has been put into normal form. □

Using this property, it is possible to give a local algorithm for alternation depth one as in the following theorem.

Theorem 5.3. *There exists a local algorithm which, given an assertion A of alternation depth one and a transition system T with a state s, determines whether s satisfies A in worst case time complexity* $O(|A||T|\log(|A||T|))$.

Proof (*Sketch*). Start with a node x in V_1. Run Avoiding 1's or Avoiding 0's (the dual of Avoiding 1's corresponding to a maximum fixed point) depending on whether $\mathcal{L}(V_1)=\mu$ or $\mathcal{L}(V_1)=\nu$, until at some point the marking of a node y in another set V_j is needed. Suspend the evaluation and run Avoiding 1's or Avoiding 0's in V_j to find the value of this y. At some point, a value in yet another set V_k might be needed, and so on. However, due to the acyclic property of Proposition 5.2 this process will stop at some point, when a node in some V_i can be determined without looking into other V_i's, and the suspended evaluations can then be resumed. Now, when the value of the node that started the search in a V_i has been determined, all the nodes visited in this search will have their correct markings, and need not be visited any more! Hence, when building the graph in a demand-driven fashion, the total execution time will be $|B|\log|B|$ where B is the subset of the graph being visited. □

6. Conclusion

The translation of a model checking problem into a problem of finding markings in boolean graphs shows the way to a rich world of algorithms. In this paper, we have presented two graph algorithms to solve the problem, but considering the wealth of graph algorithms around, there should be plenty of possibilities for finding other interesting algorithms. Moreover, the algorithms might have an interest on their own, as the graph problem – equivalent to the problem of finding fixed points in the lattice $\mathbb{O}^n$ – is a very general problem. As an example of this, the global algorithm Chasing 1's has a very close resemblance with the pebbling algorithm in [13] for solving satisfiability of propositional Horn formulas in linear time. Chasing 1's actually gives a linear-time algorithm for solving that problem and Avoiding 1's, a local algorithm which is only a logarithmic factor worse. Other applications have been investigated in Andersen [3] and the local algorithm has been extended to alternation depth two and above in Andersen [2, 4].

Another area of application is suggested by the reduction lemma from Andersen and Winskel [5]. Suppose D is a finite lattice, f a monotonic function on D, and in an ω-continuous function from $\mathbb{O}^n$ into D for an appropriate n. If it is possible to find a function g on $\mathbb{O}^n$ which is related to f as required by the lemma, i.e. $in \circ g = f \circ in$, then the minimum fixed point of f can be found by computing μg and applying in. (We claim that n and in can be chosen so that such a g can always be found.) The time to compute μg will be bounded by the size of the description of μg as a simple fixed point (in the sense of Section 2), which might be much better than using the method of computing increasing approximants, as it certainly was the case with the model checking problem.

The division idea, which is the key step in the translation from a fixed point on a powerset into a fixed point on the lattice $\mathbb{O}^n$, arose in work on trying to find compositional methods for reasoning about satisfaction. In [5], a general version of the division operator is presented. Given a process term p and an assertion A, a method is described, which computes the assertion A/p with the property:

$$x \times p : A \quad \text{iff} \quad x : A/p,$$

where $\times$ is a parallel composition operator and $p : A$ is read as "the process p satisfies the assertion A". The assertion A/p was constructed such that it belongs to the modal μ-calculus with just unary fixed-points, and for the fixed points an exponential blow-up could result from the application of Bekić's theorem. The ideas of sharing through n-ary fixed points, as used in this paper, can obviously be used in improving on these results. (A full account of these issues can be found in Andersen [4].)

Acknowledgment

I have had useful discussions with Glynn Winskel, Kim Skak Larsen, Gudmund Frandsen, and others at DAIMI. Thanks are due to Rance Cleaveland and Bernhard

Steffen for useful discussions and for pointing out some misprints and ambiguities in an earlier version; thanks are also due to the anonymous referees for helpful suggestions on improvements.

Appendix A: Proof of Theorem 4.2

In this appendix we prove correctness of the local algorithm "Avoiding 1's" (Theorem 4.2). However, we will first prove a simple useful lemma about fixed points.

Lemma A.1. *Let D be a finite lattice, g a monotonic function on D, and $m \in D$ an element with the property $m \leqslant \mu g$. Then*

$$\mu g = \bigvee_{k \in \omega} g^k(m).$$

Proof. Recall that $\mu g = \bigvee_{k \in \omega} g^k(\bot_D)$ as D is finite. From monotonicity of g, it follows easily by mathematical induction that $g^k(\bot_D) \leqslant g^k(m)$, hence, $\mu g = \bigvee g^k(\bot_D) \leqslant \bigvee g^k(m)$. For the other direction, from the assumption $m \leqslant \mu g$ and monotonicity of g, it follows by induction that $g^k(m) \leqslant \mu g$, hence, $\bigvee g^k(m) \leqslant \mu g$. □

Proof of Theorem 4.2 (Correctness part). Correctness of the local algorithm will be shown by informal use of Hoare logic. It will be informal in the sense that the formal semantics of our programming language is not given, nor will we make explicit reference to the rules of Hoare logic used in the reasoning. The intended semantics is the usual for Algol-like imperative languages.

Let Q be defined as follows:

$$\begin{aligned}
&Q(P) \Leftrightarrow_{\text{def}} \\
&\quad \forall v \in B \setminus P.\ m(v) = 0 \\
&\qquad \Leftrightarrow L(v) = \vee \ \&\ \forall w \in S(v).\ w \in B \ \&\ (m(w) = 1 \Rightarrow w \in A) \ \&\ v \in d(w) && (1) \\
&\qquad\quad \text{or } L(v) = \wedge \ \&\ \forall i (0 \leqslant i < p(v)).\ S(v)_i \in B \ \&\ m(S(v)_i) = 1 && (2) \\
&\qquad\qquad\qquad \&\ S(v)_{p(v)} \in B \\
&\qquad\qquad\qquad \&\ m(S(v)_{p(v)}) = 1 \Rightarrow S(v)_{p(v)} \in A \\
&\qquad\qquad\qquad \&\ v \in d(S(v)_{p(v)}) \\
&\quad \&\ \forall v \in B \setminus P.\ m(v) = 1 \\
&\qquad \Leftrightarrow L(v) = \vee \ \&\ \exists w \in S(v).\ w \in B \ \&\ m(w) = 1 && (3) \\
&\qquad\quad \text{or } L(v) = \wedge \ \&\ \forall w \in S(v).\ w \in B \ \&\ m(w) = 1 && (4) \\
&\& \ m \leqslant \mu g && (5)
\end{aligned}$$

$$\& \forall v \in A.\ m(v) = 1 \tag{6}$$

$$\& \forall v \in V \setminus B.\ m(v) = 0\ \&\ p(v) = 0\ \&\ d(v) = \emptyset \tag{7}$$

$$\& \forall v \in V.\ d(v) \subseteq P(v). \tag{8}$$

The assertion Q is rather complex, but it should be taken into account that it is supposed to capture the complete behaviour of a rather complicated algorithm. (Actually, it only captures formally what was described verbally in the main text.)

With small local modifications, Q is intended to be a general invariant for the complete algorithm. To see why Q is a good choice, we assume that $A = \emptyset$ and notice that $Q(\emptyset)$ now implies the following:

$$\begin{aligned}
\forall v \in B.\ m(v) = 0 \Rightarrow\ & L(v) = \vee\ \&\ \forall w \in S(v).\ w \in B\ \&\ m(w) = 0 \\
& \text{or } L(v) = \wedge\ \&\ S(v)_{p(v)} \in B\ \&\ m(S(v)_{p(v)}) = 0, \\
\forall v \in B.\ m(v) = 1 \Rightarrow\ & L(v) = \vee\ \&\ \exists w \in S(v).\ w \in B\ \&\ m(w) = 1 \\
& \text{or } L(v) = \wedge\ \&\ \forall w \in S(v).\ w \in B\ \&\ m(w) = 1.
\end{aligned}$$

Hence, by the definition of the function g induced by the graph G, we have $m|_B = g(m)|_B$, moreover, as the right-hand side only depends on the values of m on B, for *any* m' with $m'|_B = m|_B$ we have

$$g(m')|_B = m|_B. \tag{9}$$

This captures precisely an important property of B of being "self-contained", i.e. all nodes in B can be assigned correct markings based purely on other nodes in B.

Now, using (9) it is not difficult to show by mathematical induction that for all $k \in \omega$,

$$g^k(m)|_B = m|_B. \tag{10}$$

The base case is trivial. For the induction step, we assume $g^k(m)|_B = m|_B$. By (9), with $m' = g^k(m)$, we get

$$\begin{aligned}
g^{k+1}(m)|_B &= g(g^k(m))|_B \\
&= m|_B.
\end{aligned}$$

From (5) we know that $m \leqslant \mu g$, hence by Lemma A.1 we get

$$\begin{aligned}
\mu g|_B &= \left(\bigvee_{k \in \omega} g^k(m) \right)|_B \\
&= \bigvee_{k \in \omega} (g^k(m)|_B), \quad \text{as ordering is pointwise} \\
&= \bigvee_{k \in \omega} m|_B, \quad \text{by (10)} \\
&= m|_B.
\end{aligned}$$

We have shown that $Q(\emptyset)$ & $A=\emptyset$ implies $m|_B=\mu g|_B$, hence, Q is strong enough to prove the correctness of the algorithm.

Now, let c denote the body of Avoiding 1's in Fig. 2. The correctness assertion we want to show is then

$$\{x\in V\}c\{Q(\emptyset)\ \&\ A=\emptyset\ \&\ x\in B\},$$

which, by the above discussion, yields the result.

In doing so, we will be involved with a Hoare Triple capturing the effect of the procedure "*visit*":

$$\{Q(P)\}\ visit(x)\{Q(P)\ \&\ x\in B\}.$$

In words: The task of "$visit(x)$" is to extend the set of visited nodes to include x in such a way that $Q(P)$ is preserved.

Similarly, for the procedure '*fwtn*' the Hoare Triple is:

$$\{Q(P\cup\{x\})\ \&\ x\in B\ \&\ L(x)=\wedge\}fwtn(x)\{Q(P)\ \&\ x\in B\}.$$

The task of "$fwtn(x)$" is to ensure that a conjunctive node x, which currently might have a wrong marking (i.e. lines (1)–(4) of Q does not necessarily hold for x) is corrected.

Proving that these three assertions are indeed valid is a straightforward, but tedious task. (Invariants for all the loops follow easily from the pre- and post-conditions.) As an example, we sketch the proof for "*visit*". Figure 7 shows the procedure "*visit*", equipped with numbers on some of the commands.

```
if x∉B then
10: B:=B∪{x}
    if L(x)=∨ then
11:   ok:=false
12:   while p(x)<|S(x)| & ¬ok do
13:     w:=S(x)_p(x)
14:     visit(w)
15:     if m(w)=0 then d(w):=d(w)∪{x} p(x):=p(x)+1
16:     ff m(w)=1 then ok:=true
        fi
17:   if ok then m(x):=1 A:=A∪{x}
    ff L(x)=∧ then
      fwtn(x)
    fi
```

Fig. 7. *visit*(x).

We will argue that the triple

$$\{Q(P)\}\, visit(x)\, \{Q(P)\ \&\ x \in B\}$$

is valid. This will be done by cases according to the different branches of the conditionals, assuming that for any recursive call the triple is indeed valid.

To consider a nontrivial case, assume that $x \notin B$ and $L(x) = \vee$. As invariant for the while-loop (12), we take

$$I \Leftrightarrow_{\text{def}} Q(P \cup \{x\})\ \&\ J,$$

where

$$J \Leftrightarrow_{\text{def}} ok = true \Rightarrow m(S(x)_{p(x)}) = 1$$
$$\&\ \forall w \in \{S(x)_0, \ldots, S(x)_{p(x)-1}\}.$$
$$w \in B\ \&\ (m(w) = 1 \Rightarrow w \in A)\ \&\ x \in d(w).$$

Due to the precondition $Q(P)$ and the initial assignments (10) and (11), I holds initially. By assumption, the triple

$$\{Q(P \cup \{x\})\}\, visit(w)\, \{Q(P \cup \{x\})\ \&\ w \in B\}$$

is valid. Now, for $visit(w)$ to preserve I, we would have to show that J is preserved by $visit(w)$. We will, however, just assume that this is the case. (An informal argument: $visit(w)$ never decreases A or $d(w)$, nor does it affect x, hence, will not violate J.)

Branch (15) preserves $Q(P \cup \{x\})$ (increasing $d(w)$ does not affect Q), and preserves J. Branch (16) also preserves $Q(P \cup \{x\})$ and J. Hence, the body of the while-loop satisfies postcondition I under the precondition (I & $p(x) < |S(x)|$ & $\neg ok$).

When the while-loop terminates we have

$$I\ \&\ (p(x) = |S(x)| \text{ or } ok = true).$$

If $ok = true$ then from J we get $m(S(x)_{p(x)}) = 1$, hence, the assignments in (17) makes $Q(P)$ lines (1)–(4) and lines (6) and (8) valid. Let m' be the value of m after the assignment. Then $m' \leqslant \mu g$ follows from the fact that $\mu g = g(\mu g)$ and $m'(x) = g(m)(x) \leqslant g(\mu g)(x) = \mu g(x)$ Hence, line (5) of $Q(P)$ holds.

A very similar argument applies for the case $ok = false$ and $p(x) = |S(x)|$. □

Proof of Theorem 4.2 (Complexity part). We will make informal use of *amortised time analysis* (see, e.g. [11]). The crucial observation is that *no edge is visited more than twice*: once in a forward direction, and once in a backward direction. By following an edge in a forward direction, we mean following the edge from a node to a successor as it is done in "*visit*" and "*fwtn*". Whenever an edge is followed in a forward direction from a node x, the "pointer" $p(x)$ is incremented, and as the pointers $p(x)$ are never decremented, that particular edge will never again be visited. As concerns the backward direction: The while-loop in the body of the algorithm (Fig. 2) follows edges backwards by considering elements of the "dependency lists" $d(y)$. Such an edge will only be followed once – should it appear in the list $d(y)$ again, y would have changed

marking from the present marking (which is one as $y \in A$) to the marking zero. This never happens as markings always increase from zero to one when they change.

Looking at the algorithm, it is not hard to see that between two situations where an edge is followed forwards or backwards, only a constant amount of computation is performed (i.e. the cost is bounded by some constant c), hence – including the cost for initialisation – the total cost will be $O(|G|)$. Actually, ignoring the initialisation and letting B be the nodes visited when the algorithm terminates, the above analysis yields $O(|G_B|)$, where G_B is now the subgraph of G being spanned by the nodes of B (formally, $G_B = (B, E \cap B \times B, L|_B)$). □

Appendix B: Proof of Theorem 5.1

The global algorithm for the full modal μ-calculus is presented in Fig. 9. To understand the algorithm, the following notions are needed (we state them for μ, there are dual definitions for ν):

Definition B.1. A *μ-subexpression of A* is a subexpression of A with main connective μ. A *top μ-subexpression of A* is a μ-subexpression of A which is not contained in a ν-subexpression of A. A *top-level μ-subexpression of A* is a maximal μ-subexpression of A, i.e. it is not contained in other μ-subexpressions of A. Define $tl_\mu(A)$ (resp., $tl_\nu(A)$) to be the top-level μ-subexpressions (resp., ν-subexpressions) of A.

Figure 8 gives a sketch of the locations of top and top-level μ- and ν-subexpressions within an assertion. We shall extend the syntax to include constants Q in the sequel. These constants can be thought of as "a special kind of variables", which we will assume to have size zero (this does not affect the original measure of size, as these constants are not present in the assertion a priori, but only introduced by the algorithm). A constant is, of course, considered to be a closed expression.

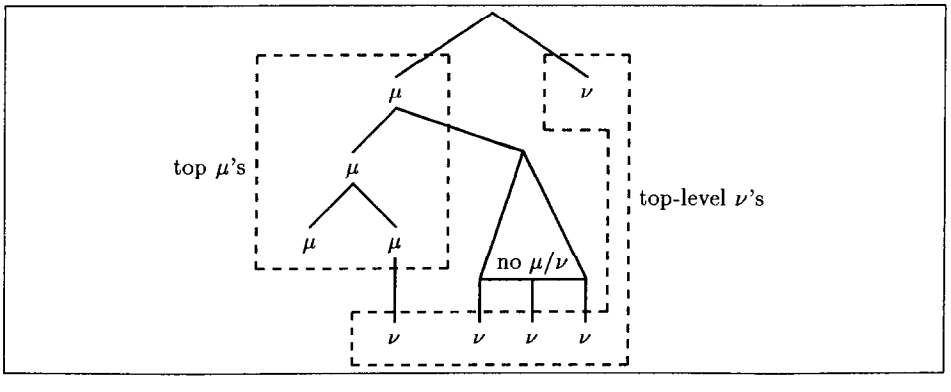


Fig. 8. Top and top-level μ- and ν-subexpressions.

Compute(A, ρ):

Let $\{B_1, \ldots, B_m\} = mcps(A)$ be the maximal, closed, proper μ/ν-subexpressions of A.
if $m > 0$ **then**
  Replace $B_1, \ldots, B_m$ in A with new constants $Q_1, \ldots, Q_m$ yielding A'.
  $\rho' := \rho[\text{Compute}(B_1, \rho)/Q_1, \ldots, \text{Compute}(B_m, \rho)/Q_m]$.
  return Compute(A', ρ')
ff $A \equiv Q$ **then**
  return $\rho(Q)$
ff $A \equiv \langle \alpha \rangle B$ **then**
  return $\{s \in S \mid \exists s' \in S.\ s \xrightarrow{\alpha} s' \ \&\ s' \in \text{Compute}(B, \rho)\}$
ff $A \equiv B_0 \wedge B_1$ **then**
  return Compute(B_0, ρ) $\cap$ Compute(B_1, ρ)
ff $A \equiv (\mu \underline{X}.\underline{B})_i$ **then**
  Let $X_1, \ldots, X_m$ be the variables bound by top μ-subexpressions of A (the first n are assumed to be $X_1, \ldots, X_n$ from $\mu \underline{X}.\underline{B}$), and let $B_1, \ldots, B_m$ be the corresponding right-hand sides.[a]
  Define

$$B' = \mu \begin{pmatrix} X_1 \\ \vdots \\ X_m \end{pmatrix}. \begin{pmatrix} B'_1 \\ \vdots \\ B'_m \end{pmatrix},$$

  where B'_i is constructed from B_i by replacing all occurrences of top μ-subexpressions $(\mu \underline{Y}.\underline{C})_j$ of A by Y_j.
  if B' is an unnested fixed point **then**
    Compute B' using the efficient algorithm for unnested fixed points, returning the ith component.
  else
    $\underline{U}^0 := (\emptyset, \ldots, \emptyset)$ $p := 0$.
    Convert the variables X_i into constants.
    repeat
      $p := p + 1$
      $\underline{U}^p := \text{Compute}((B'_1, \ldots, B'_m), \rho[\underline{U}^{p-1}/(X_1, \ldots, X_m)])$
    until $\underline{U}^p = \underline{U}^{p-1}$
    return U_i^p
ff *All remaining cases are analogous*
fi

[a] Here variables should be thought of as identifying occurrences instead of just names.

Fig. 9. Global algorithm for the full calculus.

Correctness of the algorithm is immediate from the semantics, the only non trivial case being the transformation for $A \equiv (\mu \underline{X}.\underline{B})_i$, the correctness of which follows from Bekić's theorem [7].

The notion of alternation depth will be used in the complexity analysis. It is slightly tricky to define. Emerson and Lei [14] gives an inductive definition, which is close to the definition we will use. (We have, however, changed it slightly to remedy what we consider mistakes in their definition.)

Definition B.2 (*Alternation depth*). Assume A is a closed assertion. Let $mcps(A)$ be the maximal, closed, proper μ/ν-subexpressions of A. Then define the *alternation depth* $ad(A)$ of A as follows (taking $\max(\emptyset)=0$): Let $Q_1, \ldots, Q_k$ be arbitrary constants.

if $mcps(A)=\{B_1, \ldots, B_k\} \neq \emptyset$ then

$$ad(A)=\max\{ad(A[Q_1/B_1, \ldots, Q_k/B_k]), ad(B_1), \ldots, ad(B_k)\}$$

if $mcps(A)=\emptyset$ then

$$ad(A)=\begin{cases} 0 & \text{if } A \equiv Q \\ \max\{ad(A_0), ad(A_1)\} & \text{if } A \equiv A_0 \vee A_1,\ A \equiv A_0 \wedge A_1 \\ ad(A') & \text{if } A \equiv \langle a \rangle A',\ A \equiv [a]A' \\ 1+\max\{ad(C^{cl}) \mid C \in tl_\nu(B)\} & \text{if } A \equiv (\mu \underline{X}.\underline{B})_i \\ 1+\max\{ad(C^{cl}) \mid C \in tl_\mu(B)\} & \text{if } A \equiv (\nu \underline{X}.\underline{B})_i, \end{cases}$$

where C^{cl} is constructed from C by replacing all free variables with arbitrary constants.

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

The global algorithm follows very closely the definition of alternation depth, which simplifies the analysis considerably. Formally, as in the theorem we will assume the presence of an algorithm for computing unnested fixed points $A \equiv (\mu \underline{X}.B)_i$ (and $(\nu \underline{X}.\underline{B})_i$) which runs in time $O(|A||T|)$, i.e. there exists a constant c such that the running time of the algorithm is asymptotically bounded by $c|A||T|$. One such algorithm is of course Chasing 1's. Under this assumption, we show by induction that the algorithm of Fig. 9 runs in time asymptotically bounded by $c|A|^k|S|^{k-1}|T|$, where $k=\max\{ad(A), 1\}$.

Proof of Theorem 5.1. Define the predicate P on closed assertions by

$$P(A) \Leftrightarrow_{\text{def}} \textit{for all } \rho.\ \text{Compute}(A, \rho) \textit{ executes in time asymptotically bounded by } c|A|^k|S|^{k-1}|T|,$$

where $k = \max\{ad(A), 1\}$. Assume inductively that for all A', $|A'| < |A| \Rightarrow P(A')$. We show by cases that $P(A)$ holds. In the sequel, k will always be $\max\{ad(A), 1\}$.
Case $m > 0$. By the induction hypothesis, $\text{Compute}(B_i, \rho)$ takes time $c|B_i|^{k_i}|S|^{k_i-1}|T|$, where $k_i = \max\{ad(B_i), 1\}$. Hence, letting $k' = \max\{ad(A'), 1\}$ the total time for computing A is

$$c|A'|^{k'}|S|^{k'-1}|T| + \sum_{i=1}^{m} c|B_i|^{k_i}|S|^{k_i-1}|T| \leqslant c|A'|^k|S|^{k-1}|T| + \sum_{i=1}^{m} c|B_i|^k|S|^{k-1}|T|$$

by definition of alternation depth

$$\leqslant c(|A'| + \sum_{i=1}^{m} |B_i|)^k|S|^{k-1}|T|$$

$$\leqslant c|A|^k|S|^{k-1}|T|.$$

Case $A \equiv Q$. Trivial.
Case $A \equiv \langle\alpha\rangle B$. The time to compute the diamond-modality is bounded by $c|T|$, hence, the total cost is $c|T| + c|B|^k|S|^{k-1}|T| \leqslant c(|B|+1)^k|S|^{k-1}|T| = c|A|^k|S|^{k-1}|T|$.
Case $A \equiv B_0 \wedge B_1$. As for $\langle\alpha\rangle B$.
Case $A \equiv (\mu \underline{X}.\underline{B})_i$. Observe that $|B'| \leqslant |A|$.
Subcase B' unnested. As $ad(A) = ad(B') = 1$, the claim follows immediately from the assumption about the efficient algorithm for unnested fixed points.
Subcase B' nested. Let $k_i' = \max\{ad(B_i'), 1\}$. Observe, that $k = 1 + \max\{k_i' | 1 \leqslant i \leqslant m\}$. By the induction hypothesis, each iterate U^p is computed in time

$$\sum_{i=1}^{m} c|B_i'|^{k_i'}|S|^{k_i'-1}|T| \leqslant c|B'|^{\max\{k_i' | 1 \leqslant i \leqslant m\}}|S|^{\max\{k_i' | 1 \leqslant i \leqslant m\}-1}|T|$$

$$= c|B'|^{k-1}|S|^{k-2}|T| \quad \text{as } k = 1 + \max\{k_i' | 1 \leqslant i \leqslant m\}.$$

The number of required iterations are bounded by the height of the lattice $\mathscr{P}(S)^m$, which is $m|S| \leqslant |B'||S|$. Hence, the total cost is

$$|B'||S|c|B'|^{k-1}|S|^{k-2}|T| = c|B'|^k|S|^{k-1}|T| \leqslant c|A|^k|S|^{k-1}|T|.$$

Throughout the algorithm, we have assumed that maximal, closed and top μ/ν-subexpressions of A can be detected in time $O(|A|)$ and, therefore, does not increase the overall complexity. This is justified by the assumption that variables are represented by elements of an interval of integers so that sets of variables can be represented effectively (i.e. by arrays of boolean flags). □

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] H.R. Andersen, Local computation of alternating fixed-points, Tech. Report No. 260, Computer Laboratory, Univ. of Cambridge, June 1992.

[3] H.R. Andersen. Local computation of simultaneous fixed-points, Tech. Report PB-420, Computer Science Department, Aarhus Univ., October 1992.

[4] H.R. Andersen. *Verification of Temporal Properties in Concurrent Systems.* Ph.D. Thesis, PB-445, Department of Computer Science, Aarhus Univ., Denmark, 1993.

[5] H.R. Andersen and G. Winskel, Compositional checking of satisfaction, *Formal Methods In System Design* **1** (4) (1992); extended abstract in [19].

[6] A. Arnold and P. Crubille, A linear algorithm to solve fixed-point equations on transitions systems, *Inform. Process. Lett.* **29** (1988) 57–66.

[7] H. Bekić, Definable operations in general algebras, and the theory of automata and flow charts, Lecture Notes in Computer Science, Vol. 177 (Springer, Berlin, 1984) 30–55.

[8] R. Cleaveland, Tableau-based model checking in the propositional mu-calculus, *Acta Inform.* **27** (1990) 725–747.

[9] R. Cleaveland, J. Parrow and B. Steffen, The Concurrency Workbench: A semantics based tool for the verification of concurrent systems. Tech. Report ECS-LFCS-89-83, Laboratory for Foundations of Computer Science, Univ. of Edinburgh, August 1989.

[10] R. Cleaveland and B. Steffen, A linear-time model-checking algorithm for the alternation-free modal mu-calculus; in [19].

[11] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (McGraw-Hill, New York, 1990).

[12] M. Dam, Translating CTL* into the modal μ-calculus, Tech. Report ECS-LFCS-90-123, Laboratory for Foundations of Computer Science, Univ. of Edinburgh, November 1990.

[13] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *J. Logic Programming* **1** (1984) 267–284.

[14] E.A. Emerson and C.L. Lei, Efficient model checking in fragments of the propositional mu-calculus, in: *Proc. Symp. on Logic in Computer Science* (IEEE Press, New York, 1986) 267–278.

[15] D. Kozen, Results on the propositional mu-calculus, *Theoret. Comput. Sci.* **27** (1983).

[16] K.G. Larsen, Proof systems for Hennessy–Milner logic with recursion, in: M. Dauchet and M. Nivat, eds., *Proc. CAAP, Nancy, France*, Lecture Notes in Computer Science, Vol. 299 (Springer, Berlin, 1988) 215–230.

[17] K.G. Larsen, Efficient local correctness checking, in: *Proc. of the 4th Workshop on Computer Aided Verification, June 29–July 1, 1992, Montreal, Que., Canada*, Lecture Notes in Computer Science, Vol. 663 (Springer, Berlin, 1993) 30–43.

[18] K.G. Larsen, J.C. Godskesen and M. Zeeberg, TAV – Tools for Automatic Verification, Tech. Report R 89–19, Aalborg Universitetscenter, 1989.

[19] K.G. Larsen and A. Skou, eds., *Proc. of the 3rd Workshop on Computer Aided Verification, July 1991, Aalborg*, Lecture Notes in Computer Science, Vol. 575 (Springer, Berlin, 1992).

[20] K.G. Larsen and L. Xinxin, Compositionality through an operational semantics of contexts, in: M.S. Paterson, ed., *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 443 (Springer, Berlin, 1990) 526–539.

[21] C. Stirling and D. Walker, Local model checking in the modal mu-calculus, in: J. Diaz and F. Orejas, eds., *Proc. of TAPSOFT, Barcelona, Spain*, Lecture Notes in Computer Science, Vol. 351 (Springer, Berlin, 1989) 369–383.

[22] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, *Pacific J. Math.* **5** (1955) 285–309.

[23] B. Vergauwen and J. Lewi, A linear algorithm for solving fixed-point equations on transition systems, in: J.-C. Raoult, ed., *Proc. of 17th Colloquium on Trees in Algebra and Programming, CAAP'92, Rennes, France*, Lecture Notes in Computer, Science, Vol. 581 (Springer, Berlin, 1992) 322–341.

[24] G. Winskel, A note on model checking the modal ν-calculus, in: G. Ausiello, M. Dezani-Ciancaglini and S. Ronchi Della Rocca, eds., *Proc. of ICALP*, Lecture Notes in Computer Science, Vol. 372 (Springer, Berlin, 1989) 761–772.