Model Checking and Boolean Graphs*

Henrik Reif Andersen

Department of Computer Science, Aarhus University
Ny Munkegade 116, DK-8000 Aarhus C, Denmark
E-mail: henrikan@daimi.aau.dk

Abstract

This paper describes 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 the modal μ-calculus of alternation depth one, which has time-complexity $|A||T|$, where $|A|$ is the size of the assertion and $|T|$ is the size of the model (a labelled transition system). This algorithm extends to an algorithm for the full modal μ-calculus which runs in time $(|A||T|)^{ad}$, where ad is the alternation depth, improving on earlier presented algorithms. Moreover, a local algorithm is presented for alternation depth one, which runs in time $|A||T|\log(|A||T|)$, improving on the earlier published algorithms that are all at least exponential.

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 generalisation of the modal μ-calculus presented by Kozen [Koz83] as the assertion language and as models we take labelled transition systems (essentially equivalent to labelled Kripke models). The modal μ-calculus is a very expressive modal logic (see e.g. [Koz83], [EL86], and [Dam90]) 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. Firstly, it shows that the problem of finding the sets 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

*This work is supported by the ESPRIT Basic Research Action CEDISYS and by the Danish Natural Science Research Council.

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 $|A||T|$ model checking algorithm. Thirdly, this algorithm will be extended to the full calculus, giving an algorithm running in time $(|A||T|)^{ad}$, ad being the alternation depth – a measure of how intertwined minimal and maximal 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 $|A||T|\log(|A||T|)$.

Related work can be found in Emerson and Lei [EL86] which describes an $(|A||T|)^{ad+1}$ algorithm and defines the notion of alternation depth, in Arnold and Crubille [AC88] which describes an $|A|^2|T|$ algorithm for the case of one simultaneous fixed-point, in Cleaveland and Stirling [CS91] which describes an $|A||T|$ algorithm for alternation depth one, and finally in Larsen [Lar88], Stirling and Walker [SW89], Cleaveland [Cle90], and Winskel [Win89] which all describe local model checkers that are at least exponential – even for alternation depth one.

2 Logic and models

We will consider a version of the modal ν-calculus with simultaneous fixed-points. The expressive power will be equivalent to the modal ν-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 to express *sharing* of subexpressions.

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

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

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 $(\nu\underline{X}.\underline{A})_i$ will denote the i'th component of the simultaneous maximal fixed-point $\nu\underline{X}.\underline{A}$. Dually $(\mu\underline{X}.\underline{A})_i$ denotes the i'th component of the minimal fixed-point $\mu\underline{X}.\underline{A}$. The usual unary fixed-point $\nu X.A$ corresponds to the case where $n = 1$, and for notational convenience we simply write $\nu X.A$ instead of $(\nu X.A)_1$.

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. 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 $(\mathcal{P}(S), \subseteq)$ forms a complete lattice which by taking pointwise ordering extends to a complete lattice $(\mathcal{P}(S)^n, \subseteq^n)$ on the n-ary product of $\mathcal{P}(S)$. Let $\pi_i : \mathcal{P}(S)^n \rightarrow \mathcal{P}(S)$ denote the projection onto the i'th component.

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 $\rho[U/X]$ to denote the environment which is like ρ except that X is mapped to U. The interpretation

of A denoted $[\![A]\!]_T\,\rho$ is defined inductively on the structure of A as follows:

$$\begin{array}{lcl}
[\![T]\!]_T\,\rho & = & S \\
[\![F]\!]_T\,\rho & = & \emptyset \\
[\![A_0 \wedge A_1]\!]_T\,\rho & = & [\![A_0]\!]_T\,\rho \cap [\![A_1]\!]_T\,\rho \\
[\![A_0 \vee A_1]\!]_T\,\rho & = & [\![A_0]\!]_T\,\rho \cup [\![A_1]\!]_T\,\rho \\
[\![[\alpha]A]\!]_T & = & \{s \in S \mid \forall s' \in S.\ s \xrightarrow{\alpha} s' \ \Rightarrow\ s' \in [\![A]\!]_T\,\rho\} \\
[\![\langle\alpha\rangle A]\!]_T & = & \{s \in S \mid \exists s' \in S.\ s \xrightarrow{\alpha} s' \ \&\ s' \in [\![A]\!]_T\,\rho\} \\
[\![X]\!]_T\,\rho & = & \rho(X) \\
[\![(\nu\underline{X}.\underline{A})_i]\!]_T\,\rho & = & \pi_i(\nu\psi) \\
 & & \quad \text{where } \psi : (U_1,\ldots,U_n) \mapsto ([\![A_1]\!]_T\,\rho',\ldots,[\![A_n]\!]_T\,\rho'), \\
 & & \quad \text{and } \rho' = \rho[U_1/X_1,\ldots,U_n/X_n] \\
[\![(\mu\underline{X}.\underline{A})_i]\!]_T\,\rho & = & \pi_i(\mu\psi) \\
 & & \quad \text{where } \psi \text{ is as above}
\end{array}$$

For the fixed-points we notice that the map ψ on $\mathcal{P}(S)^n$ is monotonic in all variables. According to Tarski's theorem [Tar55] then ψ will have a maximal postfixed point given by

$$\bigcup\{U \in \mathcal{P}(S)^n \mid U \subseteq^n \psi(U)\}, \tag{1}$$

which we denote $\nu\psi$. Similarly ψ will have a minimal prefixed point $\mu\psi$ given by

$$\bigcap\{U \in \mathcal{P}(S)^n \mid \psi(U) \subseteq^n U\}. \tag{2}$$

Given a transition system $T = (S, L, \rightarrow)$ we will say that a state $s \in S$ satisfies the closed 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 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 three 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 $\nu\underline{X}.\underline{A}$ is an *unnested fixed-point* if no fixed-points appear in the body $\underline{A}$. Furthermore we will say that an unnested fixed-point $\nu\underline{X}.\underline{A}$ is *simple* if each of the components A_j of $\underline{A}$ contains at most one operator, i.e. A_j is on one of the forms

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

Any unary, unnested fixed-point assertion $\nu X.A$ can be translated into an equivalent simultaneous simple fixed-point assertion, where $n = |A|$ is the size of A, measured as the number of operators. 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 $\nu\underline{X}.\underline{A}$ by

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

assuming that X_1 is associated with A. Using Bekić's theorem [Bek84] one can show the following proposition.

Proposition 1 *Let $\nu X.A$ be a closed unnested fixed-point and let $\nu \underline{X}.\underline{A}$ be the translated simple fixed-point. Then*

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

for all environments ρ.

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

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

The translation and proposition 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.

Given a transition system T and an assertion A, we will for each state s describe a method for finding an assertion B without modalities, which intuitively (when ignoring variables) has the property that $s \models_T A$ if and only if B 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 $\mathcal{P}(\{\bullet\})$. The lattice $\mathcal{P}(\{\bullet\})$ is nothing else than a distinct copy of the well-known Sierpinski space $\mathbf{O} = \{0, 1\}$ with the partial ordering $0 \leq 0, 0 \leq 1, 1 \leq 1$, so we will 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 $\mathbf{O}$ extends to a complete lattice $\mathbf{O}^n$ by extending the ordering pointwise, and note that there is an obvious isomorphism on lattices $in : \mathbf{O}^n \cong \mathcal{P}(S)$ defined by $in(x_1, \ldots, x_n) = \{s_i \in S \mid x_i = 1\}$.

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 modality free assertions with the wanted property.

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 assoicates a new set of variables $\sigma(X^i) = (X^i_1, \ldots, X^i_n)$ 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 $\rho : Var \to \mathcal{P}(S)$ and $\rho' : Var \to \mathbf{O}$, and

$$\rho(X) = in(\rho'(X_1), \ldots, \rho'(X_n))$$

[1] Larsen and Xinxin [LX90] describes a similar translation.

for all variables X with $\sigma(X) = (X_1, \ldots, X_n)$. For two such appropriate environments assume inductively that we have found A/s_i's such that

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

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

$$\theta(U_1, \ldots, U_n) = ([\![A/s_1]\!]_\bullet \rho'[U_1/X_1, \ldots, U_n/X_n], \ldots, [\![A/s_n]\!]_\bullet \rho'[U_1/X_1, \ldots, U_n/X_n]).$$

We can then by the reduction lemma below conclude that $\mu\psi = in(\mu\theta)$, hence

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

and we have found the modality free assertions corresponding to $\mu X.A$.

Lemma 2 *(Reduction lemma.)*
Suppose D and E are complete lattices of countable height, and $in : D \to E$ an ω-continuous function with $in(\bot_D) = \bot_E$. Suppose $\psi : E \to E$ and $\theta : D \to D$ are both monotonic and have the property

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

We can then conclude that

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

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

$$\begin{array}{lcl}
F/s_i & = & F \\
T/s_i & = & T \\
(A_0 \vee A_1)/s_i & = & (A_0/s_i) \vee (A_1/s_i) \\
(A_0 \wedge A_1)/s_i & = & (A_0/s_i) \wedge (A_1/s_i) \\
([\alpha]A)/s_i & = & \bigwedge_{\{j \mid s_i \xrightarrow{\alpha} s_j\}} (A/s_j) \\
(\langle\alpha\rangle A)/s_i & = & \bigvee_{\{j \mid s_i \xrightarrow{\alpha} s_j\}} (A/s_j) \\
X_j/s_i & = & X_{ij} \\
 & & \textit{where } \sigma(X_j) = (X_{1j}, \ldots, X_{nj})
\end{array}$$

For the k-ary fixed-point $\nu\underline{X}.\underline{A}$, assume that we have a change of variables σ with $\sigma(\underline{X}_j) = (\underline{\underline{X}}_{1j}, \ldots, \underline{\underline{X}}_{nj})$, and let the nk-ary fixed-point $\nu\underline{\underline{X}}.\underline{\underline{A}}$ be defined by

$$\underline{\underline{A}}_{ij} = \underline{A}_j/s_i,$$

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

$$(\mu\underline{X}.\underline{A})_j/s_i = (\mu\underline{\underline{X}}.\underline{\underline{A}})_{ji},$$

and similarly for the maximal fixed-point.[2] We have:

[2]We use double underlining as in $\underline{\underline{A}}$ to indicate matrices of assertions, which of course in this context is just a convenient way of writing large vectors.

Theorem 3 *(Quotienting theorem)*
For an arbitrary assertion A, association 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_j satisfies the closed assertion A can now be recast by applying the quotienting theorem:

$$\begin{array}{rl} s_j \models_T A & \textit{iff}\ \ s_j \in [\![A]\!]_T\,\rho\ \ \textit{for all}\ \rho \\ & \textit{iff}\ \ s_j \in in([\![A/s_1]\!]_\bullet\,\rho', \ldots, [\![A/s_n]\!]_\bullet\,\rho') \\ & \textit{iff}\ \ [\![A/s_j]\!]_\bullet\,\rho' = 1, \end{array}$$

where (ρ, ρ') is appropriate for σ and *in*. In other words, model checking can be reduced to deciding whether the assertion A/s_j denotes the top element of $\mathbf{O}$. 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 $\mu\underline{\underline{X}}.\underline{\underline{A}}$ in the lattice $\mathbf{O}^{k|S|}$. Moreover, if $\mu\underline{X}.\underline{A}$ is simple, the total size of $\mu\underline{\underline{X}}.\underline{\underline{A}}$ will be bounded by $|A||T|$, where $|T| = |S| + |L| + |\rightarrow|$, as simple calculations show:

$$\begin{array}{rcl} |\mu\underline{\underline{X}}.\underline{\underline{A}}| & = & \sum_{j=1}^{|S|}\sum_{i=1}^{k}|\underline{\underline{A}}_{ji}| \\ & = & \sum_{i=1}^{k}\sum_{j=1}^{|S|}|\underline{A}_i/s_j| \\ & \leq & \sum_{i=1}^{k}\sum_{j=1}^{|S|} max(1, |\{j' \mid \exists\alpha.\ s_j \xrightarrow{\alpha} s_{j'}\}|) \\ & \leq & \sum_{i=1}^{k}|T| \ = \ k|T| \ = \ |A||T| \end{array}$$

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 diamond- and box-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 righthand-side of the resulting assertion will be:

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

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 $2ln^2$ – and not n^l – which is less than $|A||T|$.

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

[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 timed, although it rather should be in time the logarithm of the size of the integer or memory address.

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 $K(n)$'. All these assumptions and slight abuses of language are standard in complexity analysis.

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 $|A||T|$.

3 Boolean graphs

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

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 $S(v) = \{w|(v,w) \in E\}$ and $P(v) = \{w|(w,v) \in E\}$. Given a simple boolean k-ary fixed-point $\mu\underline{X}.\underline{A}$ we can define a graph $G_{\underline{A}} = (V, E, L)$ where

$$\begin{aligned} V &= \{i \mid 1 \leq i \leq k\} \\ E &= \{(i,j) \mid (A_i = \bigvee_{l \in I} X_l \text{ or } A_i = \bigwedge_{l \in I} X_l) \;\&\; j \in I\} \\ L(i) &= \begin{cases} \vee & \text{if } A_i = \bigvee_{l \in I} X_l \\ \wedge & \text{if } A_i = \bigwedge_{l \in I} X_l \end{cases} \end{aligned}$$

Note that there is an edge from i to j iff X_j is one of the disjuncts/conjuncts in A_i, expressing the fact that the value of X_i 'depends' on the value of X_j.

A *marking* of a boolean graph G is a function $m : V \to \{0, 1\}$ 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 $\mu\underline{X}.\underline{A}$ the function g is exactly the function defined by the body of the fixed-point $\mu\underline{X}.\underline{A}$, and m is nothing else than an element of $\mathbf{O}^n$, but thinking of m as a marking will be helpful in the development of the algorithms. As $\mathbf{O}^V$ is just an isomorphic copy of $\mathbf{O}^n$, $\mathbf{O}^V$ will be a complete lattice with the same ordering as $\mathbf{O}^n$, i.e. the pointwise extension of the Sierpinski ordering. The problem we

have to solve is now: Given a boolean graph G defining the monotonic map $g : \mathbf{O}^V \to \mathbf{O}^V$, what is the minimal prefixed point $\mu g \in \mathbf{O}^V$?

4 Algorithms

In this section we will describe two algorithms for computing the minimal 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 it will on a graph G have time and space complexity $|G|$. 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 $|A||T|$, 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 only traverse a necessary subset of the graph. The approximation will be correct on x and on all nodes visited. This algorithm will on a graph G have worst-case space complexity $|G|$ and time complexity $|G| \log |G|$.

Both algorithms will be presented in the version for finding minimal fixed-points, the case of maximal fixed-points being completely dual.

4.1 A global algorithm

The global algorithm will start with the bottom element of the lattice $\mathbf{O}^V$ and gradually increase it until eventually the minimal fixed-point will be 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 minimal fixed-point.

Figure 1 describes the algorithm. The function $st : V \to Z$, where 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 $v \in V$:

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

where $st_{>0} = \{v | st(v) > 0\}$, i.e. the set of nodes which still needs some successors to become one, and $st_{\leq 0} = \{v | st(v) \leq 0\}$, 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 $\widehat{st}$ by

$$\widehat{st}(v) = \begin{cases} 1 & \text{if } st(v) \leq 0 \\ 0 & \text{if } st(v) > 0. \end{cases}$$

It is now easy to see that if $g(st) = st$ then $g(\widehat{st}) = \widehat{st}$, implying that $\widehat{st}$ 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 from the invariant I:

$$\begin{aligned} I \;\equiv\;& A \subseteq st_{\leq 0} \;\&\\ & \widehat{st} \leq \mu g \;\&\\ & \forall v \in V.\; st(v) = \begin{cases} |S(v) \cap (st_{>0} \cup A)| & \text{if } L(v) = \wedge \\ 1 - |S(v) \cap (st_{\leq 0} \setminus A)| & \text{if } L(v) = \vee \end{cases} \end{aligned}$$

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** $st(v) := \begin{cases} |S(v)| & \textbf{if } L(v) = \wedge \\ 1 & \textbf{if } L(v) = \vee \end{cases}$
$A := st_{\leq 0}$
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** $A := A \cup \{w\}$
$m := \widehat{st}$

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

Theorem 4 *The algorithm of figure 1 correctly computes the minimal 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 $st = g(st)$ and $\widehat{st}$ is a fixed-point, which by the second conjunct of the invariant is less than or equal to the minimal fixed-point, hence $\widehat{st} = \mu g$.

For the time complexity, first notice 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 outermost while-loop, will at most be executed once for each node v of the graph. Each execution of the innermost while-loop takes time proportional to the size of $P(v)$, i.e. the number of predecessors for the node v. In total the outermost 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 $|G|$. □

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 [Lar88] describes such an algorithm for the case of one fixed-point, which in an improved version is used in the TAV system [LGZ89]. Stirling and Walker [SW89] and Cleaveland [Cle90] describes a similar method for the full modal μ-calculus based on tableaux, which has been used in the implementation of the Concurrency Workbench [CPS89]. Using a single key-property of maximal fixed-points Winskel, in [Win89] develops a very similar and quite simple model checker. Unfortunately, they all have very bad worst-case behaviours. Even for the fixed-point free subset of the modal μ-calculus they have worst-case time complexity which is at least exponential in the size of assertions, and for formulas with one fixed-point, worse than 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 minimal 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 $|B|\log|B|$ where B is the subset of the graph being traversed. Thus the worst-case behaviour will be $|A||T|\log(|A||T|)$, and we have a local model checker which in the worst-case is only a logarithmic factor worse than the global model checker.

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:
$B, A := \emptyset \;\; m, p := \underline{0} \;\; d := \underline{\emptyset}$
Method:
visit$(x, \emptyset)$
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$(w, \emptyset)$
fi

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

```
visit(x, P) =

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, P ∪ {x})
                        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, P)
        fi
```

Figure 3: Visit.

```
fwtn(x, P) =

ok := false
while p(x) < |S(x)| & ¬ok do
        w := S(x)_{p(x)}
        visit(w, P ∪ {x})
        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}
```

Figure 4: Fwtn: 'find a witness'.

The local algorithm 'Avoiding 1's' is presented in figures 2, 3, and 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 minimal fixed-point marking is really a zero (the task of the *visit* procedure). This involves inspecting the successors each in turn, finding their minimal 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 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 $p : V \to I\!N$ 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 $d : V \to \mathcal{P}(V)$ will for each node v denote a subset of its predecessors $P(v)$, and this set will grow as the algorithm proceeds.

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.

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

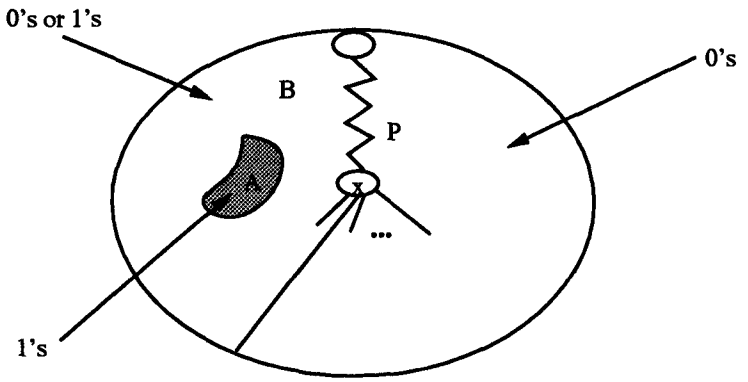

Figure 5: A typical situation of Avoiding 1's

Using a more involved invariant expressing essentially just the relationship between the variables explained above, and *amortised time analysis* (see f.ex. [CLR90]) it is possible to show the following theorem.

Theorem 5 *Given a boolean graph G with the induced function g. The algorithm described in figure 2 correctly computes an element m of $\mathbf{O}^V$ and a set $B \subseteq V$, such that*

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

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

As an alternative to the linear time initialisation, we could just omit it. Instead we would have to somehow remember the nodes that had been visited and the check '$x \in B'$ should be implemented by a search for x in the data structure representing B. Representing B as a balanced binary search tree this search could be done in logarithmic time. Moreover one could imagine the graph being constructed from the assertion and the transition system in a demand-driven fashion. Even the transition system could be build in a demand-driven manner from for instance a process algebraic term.

Assuming that each node x will be associated with a memory address a_x on which the values of d, p, d, and m at x will be stored, we have the following sketch of an algorithm.

The initialisation is changed to:

$B, A := \emptyset$

The procedure *visit* is changed to:

$visit(x, P) =$

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 $|B| \log |B|$, which in the worst-case is $|G| \log |G|$.

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 μ-calculus which has the running time $(|A||T|)^{ad}$, where ad is the alternation depth of the assertion A. For the precise definition of alternation depth, the reader is referred to [EL86].

Actually the algorithm will be slightly better than stated. Given an algorithm – like Chasing 1's – that can find a simultaneous unnested fixed-point in time $|A||T|$, we show that we can compute an arbitrary assertion of alternation depth k in time $|A|^k|S|^{k-1}|T|$. 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 by Emerson and Lei [EL86].

Theorem 6 *Given an algorithm that can find a simultaneous unnested fixed-point ($\mu X.A$ and $\nu X.A$) on a transition system T in time $|A||T|$. There exists an algorithm that will compute an arbitrary assertion A of alternation depth k in time $|A|^k|S|^{k-1}|T|$ and space $|A||T|$.*

The proof can be found in appendix A.

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 in the base-case. Another attempt of extending the

global algorithm to the full 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 division, 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 minimal and maximal fixed-points, would correspond to the element representing A. It is currently being investigated to what extent this approach can lead to new algorithms.

However, one application, which is described here, is in the generalisation of the local algorithm to alternation depth one.

The constructing of a *partitioned boolean graph* proceeds as follows: Given an assertion A. If the top-level operator is not a minimal or maximal 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 minimal (maximal) fixed-points are replaced by one minimal (maximal) fixed-point (as in the proof of theorem 6 in appendix A). Simplify all fixed-points. Assume that all variables appearing in different fixed-points of A are different, otherwise rename the variables so this is the case. Perform the division with respect to a change of variables σ and call the resulting assertion A'. Let A'_i be the right-handside corresponding to the variable X_i. Define the boolean graph $G_{A'} = (V, E, L)$ as follows:

$$
\begin{aligned}
V &= \textit{the set of all variables appearing in } A' \\
E &= \left\{ (X_i, X_j) \mid \begin{array}{rl} & (A'_i = \bigvee M \textit{ or } A'_i = \bigwedge M) \;\&\; X_j \in M \\ \textit{or} & A'_i = (\mu \underline{X}.\underline{A})_k \;\&\; X_j \textit{ is the } k\textit{'th variable in } \underline{X} \\ \textit{or} & A'_i = (\nu \underline{X}.\underline{A})_k \;\&\; X_j \textit{ is the } k\textit{'th variable in } \underline{X} \end{array} \right\} \\
L(i) &= \begin{cases} \vee & \textit{if } A'_i = \bigvee M \\ \wedge & \textit{if } A'_i = \bigwedge M \textit{ or } A'_i = (\mu \underline{X}.\underline{B})_k \textit{ or } A'_i = (\nu \underline{X}.\underline{B})_k \end{cases}
\end{aligned}
$$

Note that there is an edge from X_i to X_j if X_j is one of the disjuncts/conjuncts in A'_i, or if A'_i is a projection of a fixed-point, X_j is the variable of the fixed-point corresponding to that projection.

Assume that the fixed-points of A' are numbered from 1 to n and let V_i be the variables in the i'th fixed-point. Let $\mathcal{G}_{A'} = (G_{A'}, \mathcal{V}, \mathcal{L})$ be defined by:

$$
\begin{aligned}
\mathcal{V} &= \{V_1, \ldots, V_n\} \\
\mathcal{L}(V_i) &= \begin{cases} \mu & \textit{if the } i\textit{'th fixed-point is a minimal one} \\ \nu & \textit{if the } i\textit{'th fixed-point is a maximal one} \end{cases}
\end{aligned}
$$

We will call such a $\mathcal{G}_{A'}$ a *partitioned boolean graph.*

Cycles in this partitioned boolean graph will be of a special kind if A has alternation depth one:

Proposition 7 *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 i'th fixed-point contains the j'th fixed-point and the j'th fixed-point refers to a variable from the i'th 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 we would have a sequence of two minimal 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:

Theorem 8 *There exists a local algorithm that for 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 $|A||T|\log(|A||T|)$.*

The proof can be found in appendix B.

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 its own, as the graph problem – equivalent to the problem of finding fixed-points in the lattice $\mathbf{O}^n$ – 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 [DG84] for solving satisfiability of propositional Horn formulas in linear time, and Chasing 1's actually gives a linear time algorithm for solving that problem.

Another area of application is suggested by the Reduction lemma. Suppose you have a finite lattice D, a monotonic function f on D, and an ω-continuous function in from D into $\mathbf{O}^n$ for an appropriate n (we claim that such an n and function can always be found). If it is possible to find a function g on $\mathbf{O}^n$ which is related to f as required by the lemma, i.e. $in \circ f = g \circ in$, then the minimal fixed-point of f can be found efficiently, by computing μg and applying in. 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 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 $\mathbf{O}^n$, arose in work on trying to find compositional methods for reasoning about satisfaction. In [AW91] a general version of the division operator was presented ([AW91] also introduced the reduction lemma). Given a process term p and an assertion A, a method is described, which computes the assertion A/p with the property:

$$x \times p : A \;\; \textit{iff} \;\; x : A/p,$$

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. Currently we are investigating to what extent the ideas of sharing through n-ary fixed-points, as used in this paper, can be used in improving on these results.

Acknowledgements

I have had useful discussions with Glynn Winskel, Kim Skak Larsen, Gudmund Frandsen, and others at DAIMI.

References

[AC88] André Arnold and Paul Crubille. A linear algorithm to solve fixed-point equations on transitions systems. *Information Processing Letters*, 29:57–66, 1988.

[AW91] Henrik Reif Andersen and Glynn Winskel. Compositional checking of satisfaction. In Larsen and Skou [LS91]. To appear.

[Bek84] H. Bekić. Definable operations in general algebras, and the theory of automata and flow charts. *Lecture Notes in Computer Science*, 177, 1984.

[Cle90] Rance Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27:725–747, 1990.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[CPS89] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The Concurrency Workbench: A semantics based tool for the verification of concurrent systems. Technical Report ECS-LFCS-89-83, Laboratory for Foundations of Computer Science, Uni. of Edinburgh, August 1989.

[CS91] Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In Larsen and Skou [LS91]. To appear.

[Dam90] Mads Dam. Translating CTL* into the modal μ-calculus. Technical Report ECS-LFCS-90-123, Laboratory for Foundations of Computer Science, Uni. of Edinburgh, November 1990.

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[EL86] E. Allen Emerson and Chin-Luang Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science, Proceedings*, pages 267–278. IEEE, 1986.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27, 1983.

[Lar88] Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In *Proceedings of CAAP*, 1988.

[LGZ89] Kim G. Larsen, J.C. Godskesen, and M. Zeeberg. TAV–Tools for Automatic Verification. Technical Report R 89-19, Aalborg Universitetscenter, 1989.

[LS91] Kim G. Larsen and Arne Skou, editors. *Proceedings of the 3rd Workshop on Computer Aided Verification, Aalborg*, LNCS. Springer-Verlag, July 1991. To appear.

[LX90] Kim G. Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. In M.S. Paterson, editor, *Proceedings of ICALP*, volume 443 of *LNCS*, pages 526–539. Springer-Verlag, 1990.

[SW89] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. In *Proceedings of TAPSOFT*, 1989.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 1955.

[Win89] Glynn Winskel. A note on model checking the modal ν-calculus. In Ausiello, Dezani-Ciancaglini, and Rocca, editors, *Proceedings of ICALP*, volume 372 of *LNCS*, 1989.

Appendices

A Proof of theorem 6

The algorithm will be given along with the analysis of its time-complexity and proof of correctness which is by induction on the size of the assertion A.
Proof: (Sketch)
Define the predicate P by

$$P(A) \quad \Leftrightarrow_{\text{def}} \quad \forall \rho.\ [\![A]\!]_T\,\rho \textit{ can be computed in time } |A|^k|S|^{k-1}|T|,$$

where $k = ad(A)$. Assume inductively that for all $A', |A'| < |A| \Rightarrow P(A')$. We show by cases that $P(A)$ holds.

Case $\langle\alpha\rangle A$. Given a set $U \subseteq \mathcal{P}(S)$, the set $\{s \in S | \exists s' \in S.\ s \xrightarrow{\alpha} s' \ \&\ s' \in U\}$ can obviously be computed in time $|T|$. By the induction hypothesis, $[\![A]\!]_T\,\rho$ can be computed in time $|A|^k|S|^{k-1}|T|$, where $k = ad(A)$. Then $[\![\langle\alpha\rangle A]\!]_T\,\rho$ can be computed in time

$$\begin{aligned} |A|^k|S|^{k-1}|T| + |T| &\leq (|A|+1)^k|S|^{k-1}|T| \\ &= |\langle\alpha\rangle A|^k|S|^{k-1}|T| \end{aligned}$$

where $k = ad(A) = ad(\langle\alpha\rangle A)$.

Case X. Computing $[\![X]\!]_T\,\rho = \rho(X)$ takes time $|S|$ by the assumptions on representations of sets. Trivially we have $|S| \leq |X|^1|S|^0|T|$.

Case $\mu X.A$. We just consider the case of a unary fixed-point, arbitrary arities being similar but more cluttered up with indices. If $\mu X.A$ contains closed proper subexpressions $A_1, \ldots, A_m$ then, by the induction hypothesis, compute $[\![A_1]\!]_T\,\rho, \ldots, [\![A_m]\!]_T\,\rho$ in time

$$\sum_{i=1}^{m} |A_i|^{ad(A_i)}|S|^{ad(A_i)-1}|T| \ \leq\ (\sum_{i=1}^{m} |A_i|)^k|S|^{k-1}|T|,$$

where $k = ad(A)$. Insert new constants $Q_1, \ldots, Q_m$ for $A_1, \ldots, A_m$ in A denoting $[\![A_1]\!]_T\,\rho, \ldots, [\![A_m]\!]_T\,\rho$ and call the new assertion B. Let $\{X_1, \ldots, X_n\}$ be the variables bound by the top-level μ-operators in B, and let $\{B_1, \ldots, B_n\}$ be the corresponding bodies.[4] Define

$$B' = \mu \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} . \begin{pmatrix} B_1[X_2/\mu X_2.A_2, \ldots, X_n/\mu X_n.B_n] \\ \vdots \\ B_n[X_1/\mu X_1.B_1, \ldots, X_{n-1}/\mu X_{n-1}.B_{n-1}] \end{pmatrix} .$$

Using Bekić's theorem one can show that the first component of B' is equal to $\mu X.A$. Notice that $|B'| = |B| \leq |A|$.

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

If B' is now an unnested fixed-point, compute it in time $|B'||T|$ by the efficient algorithm for unnested fixed-points, hence $[\![B']\!]_T\,\rho$ has been computed in time

$$\begin{aligned} & (\sum_{i=1}^{m}|A_i|)^k|S|^{k-1}|T| + |B'||T| \\ \leq\ & |A|^k|S|^{k-1}|T|. \end{aligned}$$

If B' is a nested fixed-point, let B'' denote the body of B'. Observe that B'' defines a monotonic function on $\mathcal{P}(S)^n$. We will use the method of increasing approximants to calculate B'. Define

$$\begin{aligned} U^0 &= (\emptyset,\ldots,\emptyset) \\ U^{p+1} &= [\![B'']\!]_T\,\rho[U_1^p/X_1,\ldots,U_n^p/X_n]. \end{aligned}$$

Compute by iteration U^p until $U^p = U^{p-1}$. The number of iterations is bounded by the height of the lattice $\mathcal{P}(S)^n$, which is $n|S|$. Each iterate can, by the induction hypothesis, be computed in time $|B''|^{k''}|S|^{k''-1}|T|$, where $k'' = ad(B'')$. Hence the total cost will be bounded by

$$\begin{aligned} & (\sum_{i=1}^{m}|B_i|)^k|S|^{k-1}|T| + n|S||B''|^{k''}|S|^{k''-1}|T| \\ \leq\ & (\sum_{i=1}^{m}|B_i|)^k|S|^{k-1}|T| + |B''|^{k''+1}|S|^{k''}|T| \\ \leq\ & (|B''| + \sum_{i=1}^{m}|B_i|)^k|S|^{k-1}|T|, \\ & \qquad \text{as } k''+1 \leq k \text{ by the definition of alternation depth} \\ =\ & |B|^k|S|^{k-1}|T|. \end{aligned}$$

The missing cases are very similar to the ones above. □

B Proof of theorem 8

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 maximal 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. But due to the acyclic property of proposition 7 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. □