Technical Report

UCAM-CL-TR-260
ISSN 1476-2986

Number 260

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Local computation of alternating fixed-points

Henrik Reif Anderson

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Local Computation of Alternating Fixed-Points*

Henrik Reif Andersen

Computer Laboratory†
University of Cambridge
New Museums Site
Pembroke Street
Cambridge CB2 3QG
England

Abstract

In this paper we consider the problem of computing alternating fixed-points of monotone functions on finite boolean lattices. We describe a *local* (demand-driven, lazy) algorithm for computing a boolean expression with two alternating fixed-points, i.e. with a minimal and a maximal fixed-point intertwined. Such expressions arise naturally in the modal μ-calculus and is the main source to its expressive power – and its difficult model checking problem. By a translation of the model checking problem of the modal μ-calculus into a problem of finding fixed-points on boolean lattices, we get a local model checker for two alternating fixed-points which runs in time $O(|A|(|T|^2)\log(|A||T|))$, where $|A|$ is the size of the assertion and $|T|$ the size of the model, a labelled transition system. This extends earlier results by the author and improves on earlier published local algorithms. We also sketch how the algorithm can be extended to arbitrary alternations.

Due to the generality of the algorithm it can be applied to other (alternating or non-alternating) fixed-point problems.

*This work is supported by the Danish Natural Science Research Council and the Danish Research Academy.

†Internet: Henrik.Andersen@cl.cam.ac.uk, hrandersen@daimi.aau.dk

1 Introduction

The work reported in this paper arose from trying to find efficient methods for doing model checking in the modal μ-calculus, but it might turn out to have applications in other fixed-point finding problems. 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 interacting processes and the logical assertion as a specification, model checking can be viewed as the process of verifying that a system meets a specification. Taking as models labelled transition systems (essentially equivalent to labelled Kripke models) the modal μ-calculus as presented by Kozen [10] is an example of such an assertional language. It is very expressive (see e.g. Kozen [10], Emerson and Lei [9], and Dam [8]) 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'.

The expressive power of the modal μ-calculus is essentially due to the presence of minimal and maximal fixed-point operators, which can be viewed as introducing restricted forms of infinite disjunctions and conjunctions. Unfortunately, the presence of these operators, especially when minimal and maximal fixed-points are nested, complicates the task of performing model checking. In [2] it was described how a modular approach might be useful in finding efficient algorithms. The model checking problem for one fixed-point was viewed, through a translation, as a special case of a more general problem of computing fixed-points for monotone functions on boolean lattices. Two algorithms were presented for the more general problem, a global algorithm (called 'Chasing 1's') computing the 'complete fixed-point', and a local algorithm (called 'Avoiding 1's') computing only a 'part of the fixed-point'. The term 'local' has been introduced by Stirling and Walker [14] with the interpretation that fixed-points are computed in a demand-driven or lazy fashion, such that only certain 'necessary parts' of the underlying transition system will be investigated.

Chasing 1's and Avoiding 1's provided efficient model checking algorithms for the modal μ-calculus when restricted to one type of fixed-points, and for the global case an extension to the full modal μ-calculus was described running in time $O((|A||T|)^{ad})$, improving on the algorithm of Emerson and Lei [9], which runs in time $O((|A||T|)^{ad+1})$ (see their paper for a definition of ad, the alternation depth).[1] The local algorithm runs in time $O(|A||T|\log(|A||T|))$ for one fixed-point and so does the extension to full alternation depth one. For the local algorithm, however, no extension to alternating fixed-points was given. In this paper we will describe the local algorithm in a slightly different way, allowing an extension to the case of two nested fixed-points, thereby giving an $O(|A|(|T|^2)\log(|A||T|))$ algorithm for a non-trivial part of the modal μ-calculus of alternation depth two, improving on the local algorithms described by Larsen [11], Stirling and Walker [14], Cleaveland [4], and Winskel [16].

[1] The O notation means 'asymptotically bounded by'.

2 The modal μ-calculus

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 for *sharing* of subexpressions.

The syntax of the calculus is given by the following grammar:

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

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 $\vec{X}$ is shorthand for $(X_1, \ldots, X_n)$ and $\vec{A}$ for $(A_1, \ldots, A_n)$, where n should be clear from context. The assertion $(\mu\vec{X}.\vec{A})_i$ will denote the i'th component of the simultaneous minimal fixed-point $\mu\vec{X}.\vec{A}$. Dually $(\nu\vec{X}.\vec{A})_i$ denotes the i'th component of the maximal fixed-point $\nu\vec{X}.\vec{A}$. 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. 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.

Now, due to the possibility of free variables the interpretation of assertions will be given on *assertions in context*, $A\ (\vec{X})$ where $\vec{X} = (X_1, \ldots, X_k)$ is a tuple of different variables including all free variables of A. To each assertion in context we will associate a mapping

$$[\![A\ (\vec{X})]\!] : \mathcal{P}(S)^k \rightarrow \mathcal{P}(S).$$

Hence for a closed assertion we can use the empty context and get a mapping $\mathcal{P}(S)^0 \rightarrow \mathcal{P}(S)$, i.e. essentially an element of $\mathcal{P}(S)$. We extend $[\![\ ]\!]$ to tuples

$$[\![(A_1, \ldots, A_l)\ (\vec{X})]\!] : \mathcal{P}(S)^k \rightarrow \mathcal{P}(S)^l$$

by

$$[\![(A_1, \ldots, A_l)\ (\vec{X})]\!]\vec{U} = ([\![A_1\ (\vec{X})]\!]\vec{U}, \ldots, [\![A_l\ (\vec{X})]\!]\vec{U}).$$

Joining of tuples will be denoted by juxtapositioning. If $\vec{X}$ is an n-tuple of different variables and $\vec{Y}$ is a k-tuple of different variables, then $\vec{X}\vec{Y}$ will be an $(n+k)$-tuple, where any name clashes are resolved by replacing variables in $\vec{X}$ with new variables different from all other variables in $\vec{X}\vec{Y}$.

The definition of $[\![A\ (\vec{X})]\!]$ proceeds by structural induction on A as follows:

$$\begin{array}{rcl}
[\![F\ (\vec{X})]\!]\vec{U} & = & \emptyset \\
[\![T\ (\vec{X})]\!]\vec{U} & = & S \\
[\![A_0 \vee A_1\ (\vec{X})]\!]\vec{U} & = & [\![A_0\ (\vec{X})]\!]\vec{U} \cup [\![A_1\ (\vec{X})]\!]\vec{U} \\
[\![A_0 \wedge A_1\ (\vec{X})]\!]\vec{U} & = & [\![A_0\ (\vec{X})]\!]\vec{U} \cap [\![A_1\ (\vec{X})]\!]\vec{U} \\
[\![\langle\alpha\rangle A\ (\vec{X})]\!]\vec{U} & = & \{s \in S \mid \exists s' \in S.\ s \xrightarrow{\alpha} s'\ \&\ s' \in [\![A\ (\vec{X})]\!]\vec{U}\} \\
[\![[\alpha] A\ (\vec{X})]\!]\vec{U} & = & \{s \in S \mid \forall s' \in S.\ s \xrightarrow{\alpha} s' \ \Rightarrow\ s' \in [\![A\ (\vec{X})]\!]\vec{U}\} \\
[\![X_i\ (\vec{X})]\!]\vec{U} & = & U_i \\
[\![(\mu\vec{Y}.\vec{A})_i\ (\vec{X})]\!]\vec{U} & = & \pi_i(\mu\vec{V} \in \mathcal{P}(S)^l.[\![\vec{A}\ (\vec{X}\vec{Y})]\!](\vec{U}\vec{V})) \\
 & & \quad \text{where } \vec{Y} = (Y_1, \ldots, Y_l) \\
[\![(\nu\vec{Y}.\vec{A})_i\ (\vec{X})]\!]\vec{U} & = & \pi_i(\nu\vec{V} \in \mathcal{P}(S)^l.[\![\vec{A}\ (\vec{X}\vec{Y})]\!](\vec{U}\vec{V})) \\
 & & \quad \text{where } \vec{Y} = (Y_1, \ldots, Y_l)
\end{array}$$

For the fixed-points we notice that the map $\psi : \vec{V} \mapsto [\![\vec{A}\ (\vec{X}\vec{Y})]\!](\vec{U}\vec{V})$ on $\mathcal{P}(S)^l$ is monotonic. According to Tarski's theorem [15] then ψ will have a minimal prefixed point given by

$$\bigcap\{\vec{V} \in \mathcal{P}(S)^l \mid \psi(\vec{V}) \subseteq^n \vec{V}\},$$

which we denote $\mu\vec{V}.\psi(\vec{V})$. Similarly ψ will have a maximal postfixed point $\nu\vec{V}.\psi(\vec{V})$ given by

$$\bigcup\{\vec{V} \in \mathcal{P}(S)^l \mid \vec{V} \subseteq^n \psi(\vec{V})\}.$$

For a closed assertion A, we let $[\![A]\!] = [\![A\ ()]\!]*$, where $()$ is the empty context, and $*$ is the single element of $\mathcal{P}(S)^0$. Given a transition system $T = (S, L, \rightarrow)$ we will say that a state $s \in S$ satisfies the closed assertion A, if $s \in [\![A]\!]$, and use the notation $s \models A$. Deciding $s \models A$ is what is known as *model checking.*

Model checking of the modal μ-calculus depends on the alternation of minimal and maximal fixed-points. The definition of the depth of alternation is a bit tricky (see Emerson and Lei [9]). For our purpose it suffices to notice that for two fixed-points $\mu X.A[\nu Y.B]$ to be alternating (hence have alternation depth 2) it is required that X appears free in B. If this is not the case, the fixed-points are not alternating; $\nu Y.B$ appears as a constant in $\mu X.A[\nu Y.B]$ and the model checking problem is no harder than for one fixed-point (see [2] for elaboration).

Given a finite transition system, we will now describe how to translate any assertion A with free variables among $\vec{X} = (X_1, \ldots, X_k)$ into a boolean function $(\mathbb{O}^n)^k \rightarrow \mathbb{O}^n$, where $n = |S|$ and $\mathbb{O} = \{0, 1\}$ is the lattice with ordering $0 < 1$ known as Sierpinski space. In order to state this formally, we will use a *change of variables* σ which take each $X \in Var$ to an n-tuple $\sigma(X) = \vec{x} = (x_1, \ldots, x_n)$ of boolean variables, such that all variables in the image of σ are different. Given such a change of variables, σ, we associate to each assertion in context $A\ (\vec{X})$ a function $\|A\ (\vec{X}); \sigma\| : (\mathbb{O}^n)^k \rightarrow \mathbb{O}^n$ as follows:

$$\|A\ (\vec{X}); \sigma\| = \lambda\vec{x}.(A/\vec{s})$$

where $\vec{x} = (\sigma(X_1), \ldots, \sigma(X_k))$, $\vec{s} = (s_1, \ldots, s_n)$,

$$A/\vec{s} = (A/s_1, \ldots, A/s_n),$$

and

$$
\begin{aligned}
F/s_j &= F \\
T/s_j &= T \\
(A_0 \vee A_1)/s_j &= (A_0/s_j) \vee (A_1/s_j) \\
(A_0 \wedge A_1)/s_j &= (A_0/s_j) \wedge (A_1/s_j) \\
X_i/s_j &= \sigma(X_i)_j \\
\langle\alpha\rangle A/s_j &= \bigvee_{\{s \mid s_j \xrightarrow{\alpha} s'\}} A/s \\
[\alpha]A/s_j &= \bigwedge_{\{s \mid s_j \xrightarrow{\alpha} s'\}} A/s \\
(\mu(Y_1,\ldots,Y_l).(B_1,\ldots,B_l))_i/s_j &= \pi_j(\pi_i(\mu(\vec{y}_1,\ldots,\vec{y}_l).(B_1/\vec{s},\ldots,B_l/\vec{s}))) \\
&\quad \text{where } \vec{y}_h = \sigma(Y_h),\ 1 \le h \le l \\
(\nu(Y_1,\ldots,Y_l).(B_1,\ldots,B_l))_i/s_j &= \pi_j(\pi_i(\nu(\vec{y}_1,\ldots,\vec{y}_l).(B_1/\vec{s},\ldots,B_l/\vec{s}))) \\
&\quad \text{where } \vec{y}_h = \sigma(Y_h),\ 1 \le h \le l
\end{aligned}
$$

Thus the function $\| A\ (\vec{X}); \sigma \|$ is constructed from finite conjunctions, finite disjunctions, boolean variables, and fixed-points of such. We can now prove the following theorem.

Theorem 1 ([2]) *Assume $T = (S, L, \rightarrow)$ is a finite transition system with $|S| = n$ and with a numbering of the states such that $S = \{s_1, \ldots, s_n\}$. Let $\iota : \mathcal{P}(S) \xrightarrow{\simeq} \mathbb{O}^n$ be the isomorphism $\iota(U) = (x_1, \ldots, x_n)$, where $x_i = 1$ iff $s_i \in U$. Then for any assertion in context $A\ (\vec{X})$, $\vec{X} = (X_1, \ldots, X_k)$, and change of variables σ,*

$$\iota \circ [\![A\ (\vec{X})]\!] = \| A\ (\vec{X}); \sigma \| \circ \iota^{\times k}$$

where $\iota^{\times k} : \mathcal{P}(S)^k \rightarrow (\mathbb{O}^n)^k$ is the obvious extension $\iota \times \ldots \times \iota$ of ι to $\mathcal{P}(S)^k \rightarrow (\mathbb{O}^n)^k$.

For a closed assertion we can use the empty context, and define $\| A; \sigma \| = \| A\ (); \sigma \| *$. Moreover, for a closed A it follows from the theorem that

$$
\begin{aligned}
s_j \models A &\Leftrightarrow s_j \in [\![A]\!] \\
&\Leftrightarrow \pi_j(\iota([\![A]\!])) = 1 \\
&\Leftrightarrow \pi_j(\| A; \sigma \|) = 1.
\end{aligned}
$$

So to determine whether the j'th state satisfies an assertion, it suffices to check whether the j'th coordinate of $\| A; \sigma \|$ is 1. A fact which will be exploited when we later give an algorithm for solving the model checking problem.

With the translation given above, the boolean fixed-point expression given by $\| \ \|$ can actually have exponential size in the size of the original assertion, because of problems with the modalities and the fixed-points: subexpressions can be duplicated. By first transforming the assertion A into a *simple* form and then computing $\| A\ (\vec{X}); \sigma \|$ we can avoid the part of the exponential explosion, which comes from the modalities.

We will say that $\mu\vec{X}.\vec{A}$ is *simple* if each of the components A_j of $\vec{A}$ contains at most one operator, i.e. A_j is on one of the forms

$$F, T, X_{j_0} \vee X_{j_1}, X_{j_0} \wedge X_{j_1}, \langle\alpha\rangle X_{j'}, [\alpha] X_{j'}, X_{j'}, (\mu\vec{Y}.\vec{B})_i, (\nu\vec{Y}.\vec{B})_i$$

where $\mu\vec{Y}.\vec{B}$ and $\nu\vec{Y}.\vec{B}$ are simple. It is not hard to show that any fixed-point can be transformed into an equivalent simple fixed-point by adding new variables, but without increasing the size (except from the added variables, the number of which is bounded by the size of the original assertion). For one fixed-point it is now easy to see that in the translation the number of boolean variables needed is $O(|A||S|)$ and the overall size of the boolean expression is $O(|A||T|)$.

The remaining problem with exponential blow-up is easily solved. It occurs when various coordinates of the same fixed-point occur in the translated expression, for example when $(\mu\vec{Y}.\vec{B})_i/s_{j_1}$ and $(\mu\vec{Y}.\vec{B})_i/s_{j_2}$ give raise to the two expressions

$$\pi_{j_1}(\pi_i(\mu(\vec{y}_1,\ldots,\vec{y}_l).(B_1/\vec{s},\ldots,B_l/\vec{s})))$$

and

$$\pi_{j_2}(\pi_i(\mu(\vec{y}_1,\ldots,\vec{y}_l).(B_1/\vec{s},\ldots,B_l/\vec{s}))).$$

Hence, the *one* fixed-point $(\mu\vec{Y}.\vec{B})_i$ gives raise to *two identical* fixed-points in the translation. The way out of this is of course to avoid the duplication by some kind of sharing construction, like '`let_in_`' from Standard ML or by 'abstracting out the fixed-point.' However, we will not introduce a notation for this sharing because later when we consider boolean graphs the problem will vanish, but we note that by taking proper care of the sharing the resulting boolean expression will have size $O(|A||T|)$. (Again [2] should be consulted for details.)

3 Boolean Graphs

The problem we are going to solve can be formulated as follows. Suppose D and E are two finite lattices, and $g : D \times E \to D, h : D \times E \to E$ two monotonic functions. What is the value of the expression

$$\mu x.g(x, \nu y.h(x, y)), \tag{1}$$

where μ and ν denotes minimal and maximal fixed-points? (Well-definedness of this expression follows from Tarski's fixed-point theorem and some simple facts about monotonicity as in the previous section.) Expressions like this arise naturally in the modal μ-calculus. An example is $A = \mu X.[\alpha]X \vee (\nu Y.X \wedge \langle\beta\rangle Y)$.

We will consider the special case where D and E are the boolean lattices $\mathbb{O}^k$ and $\mathbb{O}^n$. Let $\vec{0} = (0,\ldots,0)$, and $\vec{1} = (1,\ldots,1)$. Now, fixed-points of a monotonic function f can be computed by a well-known *iterative method* as follows (see e.g. Aczel [1]). Let

$$\begin{aligned} f^0(x) &= x \\ f^{i+1}(x) &= f(f^i(x)). \end{aligned}$$

Then the minimal fixed-point μf of f is given by

$$\mu f = \bigvee_{i\in\mathbb{N}} f^i(\vec{0})$$

and the maximal fixed-point νf of f is given by

$$\nu f = \bigwedge_{i\in\mathbb{N}} f^i(\vec{1}).$$

Hence (1) can be found as

$$\mu x.g(x, \nu y.h(x,y)) = \bigvee_i (\lambda x.g(x, \nu y.h(x,y)))^i(\vec{0})$$

where

$$\nu y.h(x,y) = \bigwedge_i (\lambda y.h(x,y))^i(\vec{1}).$$

In order to compute the minimal fixed-point we compute the increasing approximants $\vec{0}, f(\vec{0}), f^2(\vec{0}), \ldots$ iteratively until $f^i(\vec{0}) = f^{i-1}(\vec{0})$ which is then the minimal fixed-point. In the worst-case this will not happen until i equals the height of the lattice, hence f might have to be recomputed that many times. For (1) this method can result in k computations of g and nk computations of h. The global method, Chasing 1's, of [2] essentially removes all the iterations in the case of one fixed-point, moreover in the case of two fixed-points, it can be used to improve on the iterative method, using the iterative method for the outermost fixed-point and the efficient global algorithm for the innermost fixed-point, requiring at the most k computations of g and k computations of h. The local method we describe here will be even better. Not only will it potentially only compute a part of the needed fixed-point, but it will also behave better in the worst-case: one computation of g and at most k computations of h will be needed. Considering that in the case of the modal μ-calculus with k depending on the size of the transition system, any decrease in the number of needed recomputations is important.

These efficient methods require, however, the functions g and h to be represented as what we call *monotone, boolean graphs*. To motivate the monotone boolean graphs, suppose that $f : \mathbb{O}^n \times \mathbb{O}^k \to \mathbb{O}^n$ is a monotonic function and consider the expression

$$\mu x.f(x,y), \tag{2}$$

with the free variable(s) $y = (y_1, \ldots, y_k)$, the bound variable(s) $x = (x_1, \ldots, x_n)$ and $f = (f_1, \ldots, f_n)$. Now, another way to think of (2) is as the minimal solution to the equational system

$$\begin{aligned} x_1 &= f_1(x,y) \\ &\vdots \\ x_n &= f_n(x,y) \end{aligned}$$

We will assume that all the f_i's are *simple*, i.e. they are either finite conjunctions or disjunctions of variables. This will be the case if f has arisen from the translation from the modal μ-calculus.

Now, a simple equational system can be viewed as a directed graph in the following way: The variables are nodes labelled with the operation corresponding to the right hand side of that variable, and there will be an edge from x_i to x_j if x_i appears as an argument in the right hand side of x_j. The free variables will be represented as input nodes with no operations attached. This is what we call a *monotone, boolean graph*. For an $n \in \mathbb{N}$ we let $\underline{n} = \{1, 2, \ldots, n\}$.

Definition 2 A *monotone, boolean graph* (*m.b.g.*) is a tuple

$$G = (V, V_{in}, E, in, out, L),$$

where

$$V_{in} \cap V = \emptyset,$$
$$E \subseteq (V_{in} \cup V) \times V,$$
$$in : \underline{k} \xrightarrow{\simeq} V_{in},$$
$$out : \underline{n} \to V,$$
$$L : V \to \{\vee, \wedge\}.$$

The set V of *nodes* (or *gates*), the set V_{in} of *input nodes* (or *input gates*), and the set E of directed edges are all assumed to be finite. The bijection in is a numbering of the k input nodes, and out picks out some of the internal nodes as *output nodes*. Often we will refer to G as a (k, n)-m.b.g. to stress the input-output behaviour.□

Figure 1 shows an example of a monotone boolean graph. Edges are intended to indicate

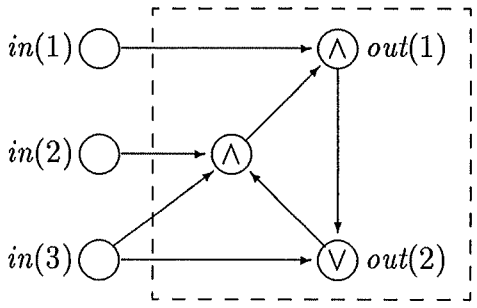

Figure 1: A $(3, 2)$-monotone boolean graph.

the direction of flow of values.[2] We will use $S(v)$ to denote the set of 'sons' of v, i.e. the set $\{w|(w, v) \in E\}$, and $P(v)$ to denote the set of 'parents' of v, i.e. the set $\{w|(v, w) \in E\}$. Notice, that G in general contains cycles.

With a m.b.g. G we can associate a *marking* $M \in \mathbb{O}^{V_{in} \cup V}$ which assigns a value to each node of the graph. Often we will consider M as consisting of an *internal marking* $m \in \mathbb{O}^V$ assigning values to the internal nodes, and an *input marking* $m_{in} \in \mathbb{O}^{V_{in}}$ assigning values to the input nodes, i.e. $M = m_{in} \cdot m$ where

$$m_{in} \cdot m(v) = \begin{cases} m_{in}(v) & \text{if } v \in dom(m_{in}) = V_{in} \\ m(v) & \text{if } v \in dom(m) = V \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Notation is summarised in appendix A. In terms of the equation system, M is an assignment of values to all variables, m_{in} to the free variables of the system, and m to the bounded variables. Now, the set of markings is, with pointwise ordering and operations, a complete lattice and G essentially defines a monotonic function $G : \mathbb{O}^{V_{in}} \times \mathbb{O}^V \to \mathbb{O}^V$ as follows:

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

[2]Observe that this is the opposite of what was used in [2]. This interpretation, however, seems more appropriate for the present discussion.

The numbering of output nodes *out* induces a map $\pi_{out} : \mathbb{O}^V \to \mathbb{O}^n$ by $\pi_{out}(m)_i = m(out(i))$, and *in* induces an isomorphism $\varphi_{in} : \mathbb{O}^k \to \mathbb{O}^{V_{in}}$, so through π_{out} and φ_{in} we can view the marking of input and output nodes as vectors of boolean values.

4 A Mu-Component

Suppose that we are really interested in computing only one component of the fixed-point, i.e. in the value $\pi_j(\pi_{out}(\mu x.G(y,x)))$ for some j and given y. Then it seems wasteful to compute the complete fixed-point if less will do. This is the task of a *local* method – it will compute only what is 'necessary' to find the value. For this purpose we will describe a data-structure K called a *Mu-Component* which will compute the fixed-point in a local fashion and it will even suppose that y is only being supplied on demand. The data-structure K will represent a m.b.g. with a marking M and have the property that performing initialisation and *any sequence of operations apart from initialisation*, will imply that *the total (amortised) cost will be proportional to the subset of the graph that has been visited plus the number of operations performed* (multiplied with a logarithmic factor due to searches). Moreover, whenever the component is 'stable', all the defined output values will be correct. Stability can be reached by repeatingly, and at most $|G|$ times, performing an operation called 'update'. It immediately follows that reaching stability will in the worst-case have a cost of $|G| \log |G|$, where $|G|$ is the size of the graph.

4.1 Description

A Mu-Component K will besides initialisation (*init*) have the following operations:

find(j) Searches for the value of output number j; returns $\bullet$ if successful (K is stable), otherwise returns $r \in \underline{k}$, if input r is needed.

update() Continues the search; returns $\bullet$ if successful (K is stable), otherwise returns $r \in \underline{k}$, if input r is needed.

$val_{in}(j)$ Returns the value of input j.

$val_{out}(j)$ Returns the value of output j.

set(j,b) Set input j to value $b \in \{0,1\}$. It is not allowed to decrease an input from 1 to 0, hence $val_{in}(j) \leq b$ prior to applying *set(j,b)*.

Tentatively the behaviour of a Mu-Component will be as follows: The operation 'find' start with the node of interest, $x = in(j)$, assume that its marking is zero, and try to verify whether this is correct. This involves inspecting the sons each in turn, finding their minimal fixed-points 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 sons have been inspected. For this purpose we assume that the sons of each node v have been numbered from 0 to $|S(v)|-1$, i.e. $S(v) = \{S(v)_0, \ldots, S(v)_{|S(v)|-1}\}$. A partial function $p : V \to I\!N$ is used in order to keep track of which son $p(v)$ of v is currently being examined, or must be examined next.

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 parents $P(v)$, and this set will grow as the algorithm proceeds.

A 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 $d(v)$. Another set $susp \subseteq V$ contains disjunctive nodes, the evaluation of which is suspended because a value of one of the sons had to be computed first.

The input nodes will not be assumed to be present a priori, hence the searching process is complicated by the fact that the value of an undefined input node might be needed. In this situation the search will stop with a number, telling which input node is needed; the input can be supplied and the evaluation proceed with 'update'. At any point of the computation, only a subset of the nodes will have assigned a marking, hence we will represent the current marking of the graph as a partial map $M : V_{in} \cup V \rightharpoonup \mathbf{0}$.

4.2 Correctness

The full description of the Mu-Component can be found in appendix B. The data-structure is constructed around an invariant I, which expresses relationships between the variables of the component and holds *invariantly*. That is, I holds after a call to 'init', and if it holds before any call to the other operations it will also hold afterwards. Let $Q(w, A, M)$ be the assertion that states that M is defined on w and only equals one if w belongs to A:

$$Q(w, A, M) \Leftrightarrow_{\text{def}} M(w) = 0 \text{ or } (M(w) = 1 \;\&\; w \in A)$$

Take I_1 to be the assertion:

$$\begin{array}{lll}
\forall x \in A. & M(x) = 1 \;\& & \\
\forall v \in V. & L(v) = \vee \;\&\; M(v) = 0 & \Rightarrow \\
& & \forall w \in \{S(v)_0, \ldots, S(v)_{p(v)-1}\}.Q(w, A, M) \;\&\; v \in d(w) \\
& & \;\&\; (p(v) = |S(v)| \text{ or } (\; v \in susp \;\&\; v \in d(S(v)_{p(v)})\;)) \\
\& & L(v) = \vee \;\&\; M(v) = 1 \quad \Rightarrow \quad \exists w \in S(v).M(w) = 1 \\
\& & L(v) = \wedge \;\&\; M(v) = 0 \quad \Rightarrow \\
& & \forall w \in \{S(v)_0, \ldots, S(v)_{p(v)-1}\}.M(w) = 1 \\
& & \;\&\; p(v) < |S(v)| \\
& & \;\&\; Q(S(v)_{p(v)}, A, M) \;\&\; v \in d(S(v)_{p(v)}) \\
\& & L(v) = \wedge \;\&\; M(v) = 1 \quad \Rightarrow \quad \forall w \in S(v).M(w) = 1 \\
\& & L(v) = \wedge \;\Rightarrow\; (v \in d(x) \;\Leftrightarrow\; x = S(v)_{p(v)})
\end{array}$$

Although I_1 appears complicated, it essentially only expresses that p, d, A, and $susp$ are fulfilling the role explained above. Write the partial function denoting the marking M as the union of an input marking m_{in} and an internal marking m, i.e. $M = m_{in} \cdot m$ and take I_2 to be defined by

$$I_2 \Leftrightarrow_{\text{def}} \forall y^0 \sqsupseteq m_{in}.m \leq \mu x.G(y^0, x)|_{dom(m)}, \tag{3}$$

where $y^0 \sqsupseteq m_{in}$ means that $dom(m_{in}) \subseteq dom(y^0) = V_{in}$ and $y^0|_{dom(m)} = m_{in}$, in other words: y^0 *extends* m_{in}. We use superscript 0 to indicate total functions, like $y^0 \in \mathbb{O}^{V_{in}}$. (See appendix A for a summary of notation.) Hence, I_2 states that m is smaller than the fixed-point wherever defined, *independent* of what is supplied for the remaining undefined inputs.

Finally, the overall invariant I is: $I \Leftrightarrow_{def} I_1 \;\&\; I_2$.

We say that K is *stable* if $A = \emptyset$ and $susp = \emptyset$. Now, if K is stable then it follows from I_1, by substituting $\emptyset$ for A and *susp*, that for all $v \in V$:

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

This implies that, whenever K is stable, *no matter what values are supplied for the undefined elements of M, the defined elements of M will stay the same under evaluation of G,* because M already contains enough information to determine the value of all these nodes. Hence, for all $y^0 \in \mathbb{O}^{V_{in}}, y^0 \sqsupseteq m_{in}, x^0 \in \mathbb{O}^V, x^0 \sqsupseteq m$,

$$m = G(y^0, x^0)|_{dom(m)}. \tag{4}$$

The properties (3) and (4) will ensure that our Mu-Component is correct. To show this, we make use of the following lemma.

Lemma 3 *Let D and E be complete lattices, $f : D \to D$ a monotonic function, and $p : D \to E$ a monotonic, surjective function s.t. for all $y \in E$ the preimage $p^{-1}(y)$ of y is a complete sublattice of E. Assume that y is an element of E s.t.*

$$\forall x \in p^{-1}(y).y = p(f(x)) \tag{5}$$

and

$$y \leq p(\mu x.f(x)) \tag{6}$$

then

$$y = p(\mu x.f(x)).$$

Proof: From (5) it follows that for all $x \in p^{-1}(y)$, $f(x) \in p^{-1}(y)$, hence f restricts to a monotonic function $f|_{p^{-1}(y)} : p^{-1}(y) \to p^{-1}(y)$. Let $x^* = \mu x.f|_{p^{-1}(y)} \in p^{-1}(y)$. Then

$$f(x^*) = f|_{p^{-1}(y)}(x^*) = x^*,$$

hence x^* is a fixed-point of f and therefore $\mu x.f(x) \leq x^*$. By monotonicity of p we get

$$p(\mu x.f(x)) \leq p(x^*) = y.$$

As by assumption (6) $y \leq p(\mu x.f(x))$ the result follows. □

We have by (3) and (4), that for all $x^0 \sqsupseteq m, y^0 \sqsupseteq m_{in}$:

$$m = G(y^0, x^0)|_{dom(m)} \text{ and } m \leq \mu x.G(y^0, x)|_{dom(m)},$$

hence by lemma 3 with $p : \mathbb{O}^V \to \mathbb{O}^{dom(m)}$ chosen to be the projection $p(x) = x|_{dom(m)}$, and $f(x') = G(y^0, x')$, we get

$$\forall y^0 \sqsupseteq m_{in}.\ m = \mu x.G(y^0, x)|_{dom(m)}.$$

We are now able to prove the correctness of K.

Theorem 4 (Correctness of K) *If K is stable with marking $M = m_{in} \cdot m$, then*

$$\forall y^0 \sqsupseteq m_{in}.\ m = \mu x.G(y^0, x)|_{dom(m)}.$$

On K any sequence of operations different from initialisation can be performed with amortised cost $O((|G| + N)log|G|)$, where N is the number of operations performed. Stability can always be reached with fewer than $|G|$ operations.

Proof: (Sketch) It is straightforward, but very tedious, to show that I is indeed an invariant, and from the discussion above we know that when K is stable then $\forall y^0 \sqsupseteq m_{in}.\ m = \mu x.G(y^0, x)|_{dom(m)}$.

For the complexity, we assume that the partial maps are implemented as balanced search trees, yielding the logarithmic factor for searches and insertions, and we observe that no edge is visited more than twice, from which the result follows by amortised analysis (see for example Cormen et al. [7]). □

As an immediate consequence we have a local algorithm for computing one fixed-point: If there are no input nodes, K computes the minimal fixed-point $\mu x.G(x)$ in time $|G| \log |G|$.

5 Connecting Two Components

It is straightforward to get a Nu-Component computing maximal fixed-points, by dualising everything of the Mu-Component: zeroes are replaced with ones, disjunctions with conjunctions etc.

Now, assume that K is a Mu-Component as described in the previous section, and that L is a Nu-Component, s.t. K represents a (k, n)-m.b.g. G and L represents a (n, k)-m.b.g. H. Define $g : \mathbb{O}^{V_L} \times \mathbb{O}^{V_K} \to \mathbb{O}^{V_K}$, and $h : \mathbb{O}^{V_K} \times \mathbb{O}^{V_L} \to \mathbb{O}^{V_L}$ by

$$\begin{aligned} g(y, x) &= G(\varphi_{in,K} \circ \pi_{out,L}(y), x), \quad \text{and} \\ h(x, y) &= H(\varphi_{in,L} \circ \pi_{out,K}(x), y). \end{aligned}$$

The map $\varphi_{in,K} \circ \pi_{out,L}$ expresses how K and L are connected by mapping output nodes of L to input nodes of K (we use subscripts K and L to indicate the component in question) and vice versa for $\varphi_{in,L} \circ \pi_{out,K}$. We will show how one might connect these components such that

$$\mu x.g(\nu y.h(x, y), x)$$

can be computed. Figure 2 shows the situation for two $(2, 2)$-m.b.g.'s.

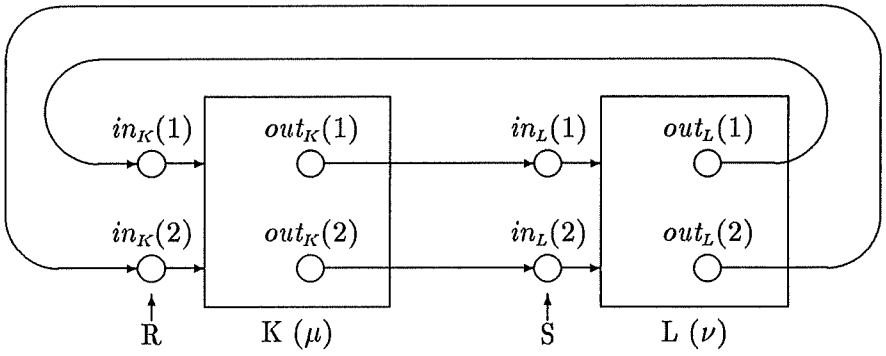

Figure 2: Connecting a Mu- and a Nu-component.

First lemma 3 will be used to make an useful observation. Assume that K and L are stable. By the properties of Mu- and Nu-components expressed in theorem 4 we know that for all $x^0 \in \mathbb{O}^{V_{in,K}}, y^0 \in \mathbb{O}^{V_{in,L}}$:

$$x^0 \sqsupseteq m_{in,L} \Rightarrow m_L = \nu y.H(x^0, y)|_{dom(m_L)}, \tag{7}$$
$$y^0 \sqsupseteq m_{in,K} \Rightarrow m_K = \mu x.G(y^0, x)|_{dom(m_K)}, \tag{8}$$

Now, suppose that for all $u \in dom(m_{in,K}), v \in dom(m_{in,L})$,

$$m_{in,K}(u) = (\varphi_{in,K} \circ \pi_{out,L}(m_L))(u), \tag{9}$$
$$m_{in,L}(v) = (\varphi_{in,L} \circ \pi_{out,K}(m_K))(v), \tag{10}$$

i.e. the input marking of node u in K when defined, agrees with the output marking of node u in L, and vice versa for v, then

$$\begin{aligned}
x^1 \sqsupseteq m_K &\Rightarrow \varphi_{in,L} \circ \pi_{out,K}(x^1) \sqsupseteq \varphi_{in,L} \circ \pi_{out,K}(m_K) \\
&\Rightarrow \varphi_{in,L} \circ \pi_{out,K}(x^1) \sqsupseteq m_{in,L} \\
&\qquad \text{by (10)} \\
&\Rightarrow m_L = \nu y.H(\varphi_{in,L} \circ \pi_{out,K}(x^1), y)|_{dom(m_L)} = \nu y.h(x^1, y)|_{dom(m_L)} \\
&\qquad \text{by (7)} \\
&\Rightarrow \varphi_{in,K} \circ \pi_{out,L}(m_L) = \varphi_{in,K} \circ \pi_{out,L}(\nu y.h(x^1, y)|_{dom(m_L)}) \sqsupseteq m_{in,K} \\
&\qquad \sqsupseteq \text{ by (9)} \\
&\Rightarrow m_K = \mu x.G(\varphi_{in,K} \circ \pi_{out,L}(\nu y.h(x^1, y), x))|_{dom(m_K)} \\
&\qquad \text{by (8)} \\
&\Rightarrow m_K = \mu x.g(\nu y.h(x^1, y), x)|_{dom(m_K)}.
\end{aligned}$$

Moreover, assume that the two-components algorithm is constructed such that

$$\begin{aligned}
m_K &\leq \mu x.g(x, \nu y.h(x, y))|_{dom(m_K)} \\
&= \mu x'.\mu x.g(x, \nu y.h(x', y))|_{dom(m_K)},
\end{aligned} \tag{11}$$

where the last equality follows from basic fixed-point theory. In other words: m_K is always 'kept too small', then by lemma 3 taking $j : \mathbb{O}^{V_K} \to \mathbb{O}^{dom(m_K)}$ to be the restriction

$(\_)|_{dom(m_K)}$, and $f(x') = \mu x.g(\nu y.h(x', y), x)$ we conclude:

$$\begin{aligned} m_K &= \mu x'.\mu x.g(x, \nu y.h(x', y))|_{dom(m_K)} \\ &= \mu x.g(x, \nu y.h(x, y))|_{dom(m_K)}. \end{aligned}$$

I.e. m_K agrees with the alternating fixed-point wherever defined, which is precisely what we want. With this observation in mind the two components algorithm will be constructed

```
K.init()
r := K.find(j)
if r = • then R := ∅ else R := {r}
while R ≠ ∅ do (1)
        L.init()
        R' := ∅  S := ∅
        while R ≠ ∅ do
                select an r ∈ R  R := R \ r  R' := R' ∪ {r}
                s := L.find(r)
                while s ≠ • do
                        S := S ∪ {s}
                        if K.val_out(s) ≠ ⊥ then L.set(s, K.val_out(s)) else L.set(s, 0)
                        s := L.update()
                K.set(r, L.val_out(r))
                r := K.update()
                if r ≠ • then R := R ∪ {r}
        if ∃s ∈ S. L.val_in(s) ≠ K.val_out(s) then R := R'
```

Figure 3: The two-components algorithm.

such that (11) is valid and tentatively it should work as follows: Start searching in K for the value of output j. If input number r is needed, start searching for this in L (which is output number r of L). If in turn L now needs input number s from K, see if this is present, otherwise *just assume that it has value* 0, and continue searching in L until stability is reached, where-after searching is continued in K until stability is reached (which might involve new searches in L). When both K and L are stable, check whether any of the s's needed by L and found to be 0 has changed into 1, if so, L must be reinitialised and all r's recomputed. Repeat recomputing as above until K and L are stable, and none of the s's changed. This algorithm is presented in figure 3. We use • to denote 'no input-value needed'. On figure 2 the nodes which might be contained in R and S are indicated.

The choice of assuming that undefined outputs from K have value 0 is crucial to the validity of (11) expressing that m_K is always smaller than the needed result, because it ensures that m_L, when stable, is smaller than its final value. Taking the value 1 could result in this property being violated. Now, alternatively one might ask: Why, when an output s from K is needed, do we not just start searching for it in K? The reason for the failure of this approach is intricate: When L is not stable, m_L might be *bigger* than the true result for this component. (Recall that L is a Nu-Component, and approaches the maximal fixed-point from above.) Hence again (11) will be broken.

Figure 4 shows a simple example illustrating the point. Assume we are interested in determining the value of $out_K(2)$. As we are in a Mu-Component we assume that $out_K(2)$

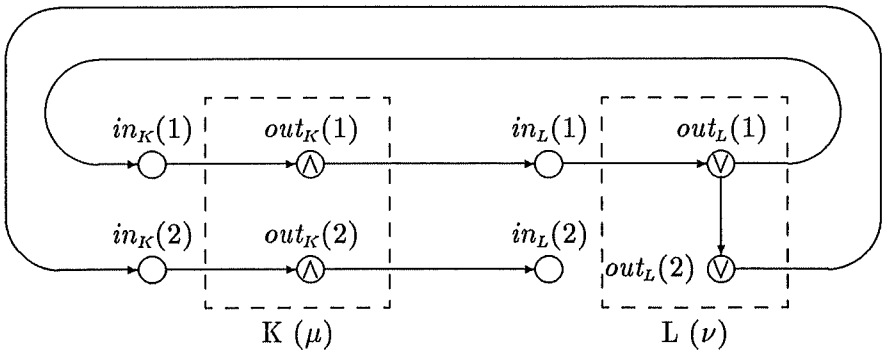

Figure 4: An intricate example. What is the correct value of $out_K(2)$?

has value 0, and investigate the son $in_K(2)$ to try to verify this. Now, $in_K(2)$ is connected to $out_L(2)$ whose value is assumed to be 1, and we find the value of $out_L(1)$, which is also assumed to be 1 and depends on $in_L(1)$ which in turn is connected to $out_K(1)$. Contrary to the algorithm which at this point assumes that $out_K(1)$ has value 0 and continues with L, we will proceed the search in K, trying to verify that the assumption of 0 is correct. Now, $in_K(1)$ – connected to $out_L(1)$ – is investigated. But $out_L(1)$ is already defined, with value 1, so $in_K(1)$ gets the value 1, hence $out_K(1)$ changes value to 1, thereby confirming that $out_L(1)$ and $out_L(2)$ are indeed 1, and also $out_K(2)$ must be changed to 1.

This is wrong! The fixed-point we are computing is really the very trivial one

$$\mu(x_1, x_2).(\nu(y_1, y_2).(x_1, y_1)),$$

which is easily seen to be $(0, 0)$ by computing approximants. The problem is the cycle $in_K(1)$–$out_K(1)$–$in_L(1)$–$out_L(1)$, which in this case is very obvious and perhaps could be handled properly, but in general can be much more involved.

It is, however, possible that by being careful, some searching in K is safe – if it does not involve inspecting output from L, which is too big. This, however, requires a new invariant and a new algorithm, and will not be further investigated here.

Theorem 5 *The algorithm in figure 3 correctly computes the value of $\mu x.g(x, \nu y.h(x, y))$ at j, more precisely when the algorithm terminates,*

$$m_K = \mu x.g(x, \nu y.h(x, y))|_{dom(m_K)},$$

and $j \in dom(m_K)$. In the worst-case this takes time $nN \log N$, where $N = |G| + |H|$ and n is the number of inputs of L from K.

Proof: (Sketch) Correctness is shown straightforwardly using Hoare logic by taking (11) as an invariant for the outermost while-loop, and applying the results of the previous discussion.

For the complexity, we note by theorem 4 that it is enough to count the number of initialisations. Clearly K is initialised only once, and L for each iteration of the outermost while-loop (marked with $^{(1)}$), which is bounded by the number of times output nodes from K can change. This yields an immediate bound of n. Hence, in the worst-case the algorithm takes time

$$|G| \log |G| + n|H| \log |H| \leq nN \log N,$$

where $N = |G| + |H|$. □

For the modal μ-calculus the translation in section 2 yields $N \leq |A||T|$ and $n \leq |S| \leq |T|$ (S being the set of states of T) for two unary fixed-points. So the model checker will run in worst-case time $O(|A|(|T|^2)\log(|A||T|))$ which is a significant improvement over exponential time algorithms.

6 Extensions

Extension of the algorithm to the full alternation depth two case, should not cause any problems and could be done along the lines of [2] where the local algorithm is generalised to full alternation depth one. For higher alternations we have situations like sketched in figure 5 (in general the picture will not be linear but more tree like). The generalisation

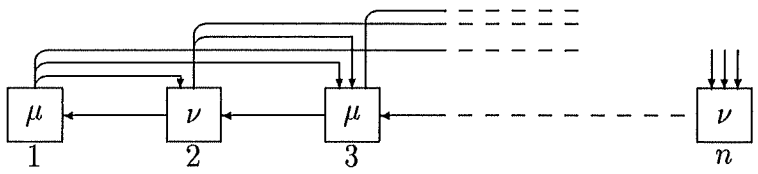

Figure 5: Higher alternation depth.

of the algorithm from the previous section could be done along the following lines:

Partition the input nodes $V_{in,i}$ of each Mu- or Nu-Component i into two: let $V^L_{in,i}$ be the set of input nodes connected to components to the left of i, and let $V^R_{in,i}$ be the set of input nodes connected to components to the right of i. Hence $V^L_{in,1} = \emptyset$ and $V^R_{in,1} = V_{in,1}$. Now, each component must have attached two sets of input nodes $R_i \subseteq V^R_{in,i}$ and $S_i \subseteq V^L_{in,i}$. The algorithm will start searching for the value of an output node of component 1, and when an input is needed start searching in the corresponding component. For the i'th component, whenever an input x in $V^R_{in,i}$ is needed, the value of which is undefined, computation is suspended in i and computation of the value of x is initiated. Whenever an input y in $V^L_{in,i}$ is needed, and the value of y is undefined, it is assumed to have the value 0 if y is an output node from a Mu-Component, and assumed to have the value 1 if it is a Nu-Component, and computation can proceed. The two sets R_i and S_i are used to collect these input nodes, and if output nodes of a component disagree with the connected input nodes (because of the 'assume 0/assume 1' strategy) components must be reinitialised, and values recomputed. Whenever a component is reinitialised all components to the right must be reinitialised too. As this brief explanation shows the details are subtle, and it seems hard to get a short description precise enough to actually prove the algorithm correct.

Nevertheless, we conjecture that this sketch can be formalised to a correct algorithm, and that properly implemented, for the modal μ-calculus it will run in worst-case time $O((|A||T|)^{ad}\log(|A||T|))$, thus have worst-case behaviour which is only a logarithmic factor worse than the best known global algorithm from [2].

7 Conclusion and Related Work

A central idea used in [2] and here, is to represent the monotone functions as monotone, boolean circuits, a special kind of directed graphs, which allows sharing of values and makes dependencies explicit. A similar idea was used, independently, by Cleaveland and Steffen [5, 6] to give another *global* $O(|A||T|)$ model checker for alternation depth one. Larsen and Xinxin [13] describes a translation from the model checking problem to the problem of finding solutions to modality free equations, similar to the one described here, but without utilising the possible benefits for a model checker. The translation can also be seen as a special case of the reduction for products in [3].

Other local algorithms are described by Larsen [11], Stirling and Walker [14], Cleaveland [4], and Winskel [16]. They all have bad worst-case complexities: exponential, even for alternation depth one. The local algorithm in [2] for alternation depth one runs in time $O(|A||T|\log(|A||T|))$.

The general problem of finding fixed-points of monotone functions on boolean lattices, might turn out to be useful in other contexts. For instance when finding fixed-points in program analysis. Moreover, the ideas of sharing and chasing dependencies are not inherently connected to boolean lattices. One might consider a generalised notion of circuits over a given basis of operations and values $\mathcal{B}$. With the value domain bigger than $\mathbf{O}$, nodes might have to be recomputed more than once – still this approach will avoid a lot of computations compared to the iterative method.

References

[1] P. Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, 1983.

[2] Henrik Reif Andersen. Model checking and boolean graphs. In B. Krieg-Brückner, editor, *Proceedings of 4'th European Symposium on Programming, ESOP'92, Rennes, France*, volume 582 of *LNCS*. Springer-Verlag, 1992.

[3] Henrik Reif Andersen and Glynn Winskel. Compositional checking of satisfaction. *Formal Methods In System Design*, 1992. To appear. Extended abstract in [12].

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

[5] Rance Cleaveland and Bernhard Steffen. Computing behavioural relations, logically. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Proceedings of ICALP*, volume 510 of *LNCS*, pages 127–138. Springer-Verlag, July 1991.

[6] Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In Larsen and Skou [12].

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

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

[9] 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.

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

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

[12] Kim G. Larsen and Arne Skou, editors. *Proceedings of the 3rd Workshop on Computer Aided Verification, July 1991, Aalborg*, volume 575 of *LNCS*. Springer-Verlag, 1992.

[13] 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.

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

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

[16] 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.

A Appendix. Summary of Notation

We use $A \rightharpoonup B$ to denote the set of partial functions from A to B. This set can be viewed as the set of total functions $A \to B_\perp = (B_\perp)^A$, where $\perp$ is assumed to be an element not in B denoting undefinedness and $B_\perp = B \cup \{\perp\}$. For $m : A \rightharpoonup B$ we let $dom(m)$ be the domain of definition of m. Then m can also be viewed as an element of $dom(m) \to B = B^{dom(m)}$ through the restriction $m|_{dom(m)}$. For partial maps $m, m' : A \rightharpoonup B$ we define $m \sqsubseteq m'$ 'm extends to m'' by

$$m \sqsubseteq m' \;\Leftrightarrow_{def}\; dom(m) \subseteq dom(m') \;\&\; \forall x \in dom(m).m(x) = m'(x).$$

Often we will view the map m as being at the same time an element of $A \rightharpoonup B$, $B^{dom(m)}$, and $(B_\perp)^A$. We will also use $\perp$ to denote the partial map with empty domain.

If $m \in A^B, m' \in A^C$ are total maps with $B \cap C = \emptyset$ then $m \cdot m' \in A^{B \cup C}$ is defined as $(m \cdot m')(x) = m(x)$ for $x \in B$ and $(m \cdot m')(x) = m'(x)$ for $x \in C$. (Note, that with the abovementioned conventions this construction also works for partial maps.) If $m \in A^B$ and $a \in A$ then $m[a/b] \in A^{B \cup \{b\}}$ is defined by $m[a/b](x) = x$ for $x \neq b$, $m[a/b](b) = a$.

For $k \in \mathbb{N}$, we let $\underline{k} = \{1, 2, \ldots, k\}$. The function $\pi_i : D_1 \times \ldots \times D_n \to D_i$ is the usual projection onto the i'th component.

B Appendix. The Mu-Component

In this appendix we describe the Mu-Component in a syntax which should be self-explanatory.

Data-structure: Mu-component

Variables:	$G = (V, V_{in}, in, out, E, L)$: (k,n)-m.b.g.	
	$A \subseteq V_{in} \cup V$	
	$susp \subseteq V$	
	$M = m_{in} \cdot m : V_{in} \cup V \rightharpoonup \mathbf{0}$	
	$p : V \rightharpoonup \mathbb{N}$	
	$d : V_{in} \cup V \rightharpoonup \mathcal{P}(V)$	
Invariant:	I	
Operations:	$init() \to ()$	
	$find(j) \to r$	$j \in \underline{n}, r \in \underline{k} \cup \{\bullet\}$
	$update() \to r$	$r \in \underline{k} \cup \{\bullet\}$
	$set(j, b) \to ()$	$j \in \underline{n}, b \in \{0, 1\}$
		if $M(j) = 1$ then only $b = 1$ is allowed
	$val_{in}(j) \to b$	$j \in \underline{k}, b \in \{\perp, 0, 1\}$
	$val_{out}(j) \to b$	$j \in \underline{n}, b \in \{\perp, 0, 1\}$
	$(visit(x) \to r)$	$x \in V, r \in \underline{k} \cup \{\bullet\}$, *only used internally*

$\text{init}() \to () : A, susp := \emptyset \;\; M, p, d := \perp$

find$(j) \rightarrow r$:

if $M(out(j)) \neq \perp$ **then** $r := update()$
ff $M(out(j)) = \perp$ **then**
$M := M[0/out(j)]$ $p := p[0/out(j)]$ $d := d[\emptyset/out(j)]$
$r := visit(out(j))$
if $r = \bullet$ **then** $r := update()$

update$() \rightarrow r$:

$r := \bullet$
while $susp \neq \emptyset$ *or* $A \neq \emptyset$ **do**
while $A \neq \emptyset$ **do**
select an $x \in A$
while $d(x) \neq \emptyset$ **do**
select a $y \in d(x)$ $d(x) := d(x) \setminus y$
if $L(y) = \vee$ & $M(y) = 0$ **then**
$M(y) := 1$ $A := A \cup \{y\}$ $susp := susp \setminus y$
ff $L(y) = \wedge$ **then**
$p(y) := p(y) + 1$
$r := visit(y)$
if $r \neq \bullet$ **then exit** *update*
$A := A \setminus x$
if $susp \neq \emptyset$ **then**
select an $x \in susp$ $susp := susp \setminus x$
if $M(x) = 0$ **then**
$r := visit(x)$
if $r \neq \bullet$ **then exit** *update*

set$(j, b) \rightarrow ()$:

if $M(in(j)) = 1$ & $b = 0$ **then error**
ff $M(in(j)) = \perp$ & $b = 0$ **then** $M(in(j)) := 0$ $d(in(j)) := \emptyset$
ff $M(in(j)) = 0$ & $b = 1$ **then** $M(in(j)) := 1$ $A := A \cup \{in(j)\}$
ff $M(in(j)) = \perp$ & $b = 1$ **then** $M := M[1/in(j)]$ $d := d[\emptyset/in(j)]$

val$_{in}(j) \rightarrow b : b := M(in(j))$
val$_{out}(j) \rightarrow b : b := M(out(j))$

visit$(x) \rightarrow r$:

loop
if $L(x) = \vee$ **then**
if $p(x) = |S(x)|$ **then**
$r := \bullet$ **exit loop**
ff $p(x) < |S(x)|$ **then**
$w := S(x)_{p(x)}$
if $M(w) = \perp$ & $w \in V_{in}$ **then**

$M := M[0/w]$ $d := d[\{x\}/w]$ $susp := susp \cup \{x\}$
$r := in(w)$ **exit loop**

ff $M(w) = \bot$ & $w \notin V_{in}$ **then**
$M := M[0/w]$ $p := p[0/x]$ $d := d[\{x\}/w]$
$susp := susp \cup \{x\}$ $x := w$

ff $M(w) = 0$ **then**
$d(w) := d(w) \cup \{x\}$ $p(x) := p(x) + 1$

ff $M(w) = 1$ **then**
$M(x) := 1$ $A := A \cup \{x\}$ $r := \bullet$ **exit loop**

ff $L(x) = \wedge$ **then**
if $p(x) = |S(x)|$ **then**
$M(x) := 1$ $A := A \cup \{x\}$ $r := \bullet$ **exit loop**

ff $p(x) < |S(x)|$ **then**
$w := S(x)_{p(x)}$
if $M(w) = \bot$ & $w \in V_{in}$ **then**
$M := M[0/w]$ $d := d[\{x\}/w]$
$r := in(w)$ **exit loop**

ff $M(w) = \bot$ & $w \notin V_{in}$ **then**
$M := M[0/w]$ $p := p[0/x]$ $d := d[\{x\}/w]$
$x := w$

ff $M(w) = 0$ **then**
$d(w) := d(w) \cup \{x\}$ $r := \bullet$ **exit loop**

ff $M(w) = 1$ **then**
$p(x) := p(x) + 1$