An Improved Algorithm for the Evaluation of Fixpoint Expressions*

David E. Long
AT&T Bell Laboratories

Anca Browne Edmund M. Clarke
Somesh Jha Will Marrero
Carnegie Mellon University
School of Computer Science

April 5, 1995

Abstract

Many automated finite-state verification procedures can be viewed as fixpoint computations over a finite lattice (typically the powerset of the set of system states). For this reason, fixpoint calculi such as those proposed by Kozen and Park have proven useful, both as ways to describe verification algorithms and as specification formalisms in their own right. We consider the problem of evaluating expressions in these calculi over a given model. A naive algorithm for this task

*This research was sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Material Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330.

may require time n^q, where n is the maximum length of a chain in the lattice and q is the depth of fixpoint nesting. In 1986, Emerson and Lei presented a method requiring about n^d steps, where d is the number of alternations between least and greatest fixpoints. More recent algorithms have succeeding in reducing the exponent by one or two, but the complexity has remained at about n^d. In this paper, we present a new algorithm that makes extensive use of monotonicity considerations to solve the problem in about $n^{d/2}$ steps.

1 Introduction

Many automated finite-state verification algorithms can be viewed as fixpoint computations over a finite lattice. Examples include: model checking procedures for logics such as CTL [6, 7] and PDL [14], methods for computing strong and weak bisimulation equivalence in CCS [19], and language containment and emptiness algorithms for ω-automata [5]. Approaches based on fixpoint logics such as the propositional μ-calculus [15] are tied even more directly to fixpoint computation. With the increasing use of binary decision diagrams (BDDs) [3] for finite-state verification [4, 11, 18], fixpoint-based algorithms have become even more important, since methods that require the manipulation of individual states do not take advantage of this representation. In this paper, we consider the complexity of evaluating fixpoint expressions over finite lattices. Our main result is a new algorithm that makes extensive use of monotonicity considerations to reduce the complexity of evaluation. The number of steps required by our method is roughly the square root of the number of steps required by the best previously known algorithms.

Numerous fixpoint calculi have been described in the literature [12, 15, 20], and our ideas for evaluating fixpoint expressions will work with any of them. However, for concreteness, we will be using the propositional μ-calculus of Kozen [15]. This logic is designed for expressing properties of transition systems, and formulas in the logic (with no free propositional variables) evaluate to sets of states. There have been many algorithms proposed for evaluating a formula of the logic with respect to a given transition system. These mostly fall into two categories: local and global. Local procedures are designed for proving that a specific state of the transition system satisfies the given formula. Because of this, it is not always necessary to examine all

the states in the transition system. However, the worst-case complexity of these approaches is generally larger than the complexity of the global methods. Tableau-based local approaches have been developed by Cleaveland [8], Stirling and Walker [21], and Winskel [23]. More recently, Andersen [1] and Larsen [16] have developed efficient local methods for a subset of the μ-calculus. Mader [17] has also proposed improvements to the tableau-based method of Stirling and Walker that seem to increase its efficiency (though Mader does not give a complexity bound). Global procedures generally work bottom-up through the formula, evaluating each subformula based on the value of its subformulas. Iteration is used to compute the fixpoints. Because of fixpoint nesting, a naive global algorithm may require about n^q steps to evaluate a formula, where n is the number of states in the transition system and q is the depth of nesting of the fixpoints. Emerson and Lei [13] improved on this by observing that the complexity of evaluating a formula really depends only the number of alternations of least and greatest fixpoints. That is, successively nested fixpoints of the same type do not increase the complexity of the computation. Emerson and Lei formalized this using the notion of alternation depth, and they gave an algorithm requiring only about n^d steps, where d is the alternation depth. In an implementation, bookkeeping and set manipulations may add another factor of n or so to the time required. Subsequent work by Cleaveland, Klein, Steffen, and Andersen [1, 9, 10] has reduced this extra complexity, but the overall number of steps has remained at about n^d. Our new algorithm is also a global method. By using extensive monotonicity considerations, we are able to show that only about $n^{d/2}$ steps are required to evaluate a formula with alternation depth d. Thus, our method requires only about the square root of the time needed by the earlier algorithms.

The remainder of the paper is organized as follows. Section 2 summarizes the syntax and semantics of the propositional μ-calculus and briefly reviews Emerson and Lei's work. In section 3 we give our new algorithm, present a proof of correctness, and show that it requires no more than about $n^{d/2}$ steps. Section 4 presents an example that shows that our complexity bound is tight, i.e., there are cases where the algorithm does use $n^{d/2}$ steps. We discuss some open questions and directions for future research in section 5.

2 The Propositional μ-Calculus

In the propositional μ-calculus, formulas are built up from:

1. atomic propositions p, p_1, p_2, ...;
2. atomic propositional variables R, R_1, R_2, ...;
3. logical connectives $\cdot \wedge \cdot$ and $\cdot \vee \cdot$;
4. modal operators $\langle a \rangle \cdot$ and $[a] \cdot$, where a is one of a set of program letters a, b, a_1, a_2, ...; and
5. fixpoint operators $\mu R_i.(\cdots)$ and $\nu R_i.(\cdots)$.

(We can also allow negations to be applied to atomic propositions, but this is not important for our purposes.) Formulas in this calculus are interpreted relative to a transition system that consists of:

1. a nonempty set of states $\top$ (throughout this paper the size of this set is denoted by n);
2. a mapping L that takes each atomic proposition to some subset of $\top$ (the states where the proposition is true); and
3. a mapping T that takes each program letter to a binary relation over $\top$ (the state changes that can result from executing the program).

The intuitive meaning of the formula $\langle a \rangle \phi$ is "it is possible to execute a and transition to a state where ϕ holds". $[\cdot]$ is the dual of $\langle \cdot \rangle$; for $[a]\phi$, the intended meaning is that "ϕ holds in all states reachable (in one step) by executing a." The μ and ν operators are used to express least and greatest fixpoints, respectively. To emphasize the duality between least and greatest fixpoints, we write the empty set of states as $\bot$.

Formally, a formula ϕ depending on free propositional variables R_1, R_2, ..., R_k is interpreted as a k-argument predicate transformer. (A predicate transformer is simply a mapping from sets of states to a set of states.) We write this predicate transformer as ϕ^M. ϕ^M is defined inductively by giving its value $\phi^M(\bar{S})$ for a vector $\bar{S} = (S_1, \ldots, S_k)$ of arguments.

1. $p^M(\bar{S}) = L(p)$.

2. $R_i^M(\bar{S}) = S_i$.

3. $(\phi \wedge \psi)^M(\bar{S}) = \phi^M(\bar{S}) \cap \psi^M(\bar{S})$. Disjunction is similar.

4. $(\langle a \rangle \phi)^M(\bar{S}) = \{ s \mid \exists t\, [(s,t) \in T(a) \wedge t \in \phi^M(\bar{S})] \}$.
   $([a]\phi)^M(\bar{S}) = \{ s \mid \forall t\, [(s,t) \in T(a) \rightarrow t \in \phi^M(\bar{S})] \}$.

5. $(\mu R.\, \phi)^M(\bar{S})$ is defined to be the least fixpoint of the predicate transformer $\tau: 2^{\mathsf{T}} \rightarrow 2^{\mathsf{T}}$ defined by:

$$\tau(S) = \phi^M(S, \bar{S}),$$

   where the first parameter of ϕ^M is the value for R. The interpretation of $\nu R.\, \phi$ is similar, except that we take the greatest fixpoint.

Within formulas, there is no negation (except potentially on the atomic propositions), and so the fixpoints are guaranteed to be well-defined. Formally, each possible τ is monotonic ($S \subseteq S'$ implies $\tau(S) \subseteq \tau(S')$). This is enough to ensure the existence of the fixpoints [22]. Further, since we will be evaluating formulas only over finite transition systems, monotonicity of τ implies that τ is also $\cup$-continuous and $\cap$-continuous, and hence the least and greatest fixpoints can be computed by iterative evaluation:

$$(\mu R.\, \phi)^M(\bar{S}) = \bigcup_i \tau^i(\bot) \qquad (\nu R.\, \phi)^M(\bar{S}) = \bigcap_i \tau^i(\top).$$

Since the domain is finite, the iteration must stop after a finite number of steps. More precisely, for some $i \leq |\top|$, the fixpoint is equal to $\tau^i(\bot)$ (for a least fixpoint) or $\tau^i(\top)$ (for a greatest fixpoint). To find the fixpoint, we repeatedly apply τ starting from $\bot$ or from $\top$ until the result does not change.

Since we will be using the concept of alternation depth, we briefly summarize Emerson and Lei's observations [13]. Consider the expression

$$\mu R_1.\, (\langle a \rangle R_1) \vee (\mu R_2.\, R_1 \vee p \vee \langle b \rangle R_2).$$

The subformula $\mu R_2.\, (\cdots)$ defines a monotonic predicate transformer τ taking one set (the value of R_1) to another (the value of the least fixpoint of R_2). When evaluating the outer fixpoint, we start with the approximation $\bot$ and then compute $\tau(\bot)$. Now R_1 is increased (say to S_1), and we want to compute the least fixpoint $\tau(S_1)$. Since $\bot \subseteq S_1$, by monotonicity we know that

$\tau(\bot) \subseteq \tau(S_1)$. To compute a least fixpoint, it is enough to start iterating with any approximation known to be below the fixpoint. Thus here, we can start iterating with $\tau(\bot)$ instead of $\bot$. At the next step, R_1 will be even larger, and so we will start the inner fixpoint computation with $\tau(S_1)$. We never restart the inner fixpoint computation, and so we can have at most about n increases in the value of the inner fixpoint variable. Overall, we only need about n steps to evaluate this expression, instead of n^2.

Emerson and Lei show that this type of simplification makes it possible to evaluate a formula ϕ in about n^d steps, where d is the alternation depth of the formula. The alternation depth of a formula is intuitively equal to the number of alternating nestings of least and greatest fixpoints. Formally, the alternation depth is defined as follows. Assume for simplicity that the formula does not contain any non-atomic subformulas that which do not contain free propositional variables (these can be independently evaluated and then treated as atomic propositions).

1. The alternation depth of an atomic proposition or propositional variable is 0;

2. The alternation depth for formulas like $\phi \wedge \psi$, $\phi \vee \psi$, $\langle a \rangle \phi$, etc., is the maximum alternation depth of the subformulas ϕ, ψ, etc.

3. The alternation depth of $\mu R.\, \phi$ is the maximum of: one, the alternation depth of ϕ, and one plus the alternation depth of any top-level ν-subformulas of ϕ. A top-level ν-subformula of ϕ is a subformula $\nu R'.\, \psi$ of ϕ that is not contained within any other fixpoint subformula of ϕ. The alternation depth for $\nu R.\, \phi$ is similarly defined.

3 The Algorithm

To simplify notation, we consider a fixpoint computation of the form:

$$
\begin{aligned}
F_1 &\equiv \mu R_1.\, \psi_1(R_1, F_2) \\
F_2 &\equiv \nu R_2.\, \psi_2(R_1, R_2, F_3) \\
F_3 &\equiv \mu R_3.\, \psi_3(R_1, R_2, R_3, F_4) \\
F_4 &\equiv \nu R_4.\, \psi_4(R_1, R_2, R_3, R_4, F_5) \\
&\vdots
\end{aligned}
$$

$$F_d \equiv \sigma_d R_d.\, \psi_q(R_1, R_2, \ldots, R_d),$$

where $\equiv$ denotes syntactic equality. Note that d is the alternation depth of this formula. We write $\sigma_k R_k.\,(\cdots)$ to mean $\mu R_k.\,(\cdots)$ if R_k is given by a least fixpoint, and to mean $\nu R_k.\,(\cdots)$ otherwise. Define $\phi_d \equiv \psi_d$, and let

$$\phi_k(R_1, \ldots, R_k) \equiv \psi_k(R_1, \ldots, R_k, F_{k+1})$$

for $k < d$. Note that ψ_k is a formula with $k+1$ free propositional variables (except for ψ_d); the last parameter gives the value of the inner fixpoint. Then ϕ_k is obtained from ψ_k by instantiating the last parameter with the inner fixpoint. For notational simplicity, we will identify syntactic formulas with their interpretations in the discussion below. So, depending on context, a formula like ϕ_k may mean either the actual formula or the k-argument predicate transformer ϕ_k^M.

3.1 The basic idea

Before going into details of our new algorithm, we illustrate the idea on a formula involving three fixpoints:

$$\mu R_1.\, \psi_1(R_1, \nu R_2.\, \psi_2(R_1, R_2, \mu R_3.\, \psi_3(R_1, R_2, R_3))).$$

To compute the outer fixpoint, we start with $R_1 = \bot$, $R_2 = \top$ and $R_3 = \bot$. Call these values R_1^0, R_2^{00}, and R_3^{000} respectively. The superscript on R_k gives the iteration indices for the fixpoints involving $R_1, \ldots R_k$. We then iterate to compute the inner fixpoint; call the value of this fixpoint $R_3^{00\omega}$. We now compute the next approximation R_2^{01} for R_2 by evaluating $\psi_2(R_1^0, R_2^{00}, R_3^{00\omega})$ and go back to the inner fixpoint. Eventually, we reach the fixpoint for R_2, having computed R_2^{00}, $R_3^{00\omega}$, R_2^{01}, $R_3^{01\omega}$, $\ldots$, $R_2^{0\omega}$, $R_3^{0\omega\omega}$. Now we proceed to $R_1^1 = \psi_1(R_1^0, R_2^{0\omega}, R_3^{0\omega\omega})$. We know that $R_1^0 \subseteq R_1^1$, and we are now going to compute $R_2^{1\omega}$. Note that the values $R_2^{0\omega}$ and $R_2^{1\omega}$ are given by

$$R_2^{0\omega} = \nu R_2.\, \psi_2(R_1^0, R_2, \mu R_3.\, \psi_3(R_1^0, R_2, R_3))$$

and

$$R_2^{1\omega} = \nu R_2.\, \psi_2(R_1^1, R_2, \mu R_3.\, \psi_3(R_1^1, R_2, R_3)).$$

By monotonicity, we know that $R_2^{1\omega}$ will be a superset of $R_2^{0\omega}$. However, since R_2 is computed by a greatest fixpoint, this information does not help; we still

must start computing with $R_2^{10} = \top$. At this point, we begin to compute the inner fixpoint again. But now let us look at $R_3^{00\omega}$ and $R_3^{10\omega}$. We have

$$R_3^{00\omega} = \mu R_3.\, \psi_3(R_1^0, R_2^{00}, R_3)$$

and

$$R_3^{10\omega} = \mu R_3.\, \psi_3(R_1^1, R_2^{10}, R_3).$$

Since $R_1^0 \subseteq R_1^1$ and $R_2^{00} \subseteq R_2^{10}$, monotonicity implies that $R_3^{00\omega} \subseteq R_3^{10\omega}$. Now R_3 is a least fixpoint, so starting the computation of $R_3^{10\omega}$ anywhere below the fixpoint value is acceptable. Thus, we can start the computation for $R_3^{10\omega}$ with $R_3^{100} = R_3^{00\omega}$. Since $R_3^{00\omega}$ is in general larger than $\bot$, we obtain faster convergence. Also note that since $R_1^0 \subseteq R_1^1$ and $R_3^{00\omega} \subseteq R_3^{10\omega}$, we will have $R_2^{01} \subseteq R_2^{11}$. This means that we can use the same trick when computing $R_3^{11\omega}$: we start the computation from $R_3^{110} = R_3^{01\omega}$. In general, we can start computing $R_3^{1j\omega}$ from $R_3^{1j0} = R_3^{0j\omega}$. Similarly, once we find R_1^2 (or in general, R_1^{k+1}), we can start computing the inner fixpoints from $R_3^{1m\omega}$ ($R_3^{km\omega}$).

If we use this idea, how many steps does the computation take? The dominating term is the number of steps made when computing the inner fixpoint. With previously known algorithms, this inner computation starts from $\bot$ each time, and hence may involve about n^3 steps (one factor of n for each of the three fixpoints). In our case, if we fix a particular j, then we have

$$R_3^{0j0} \subseteq R_3^{0j\omega} = R_3^{1j0} \subseteq R_3^{1j\omega} = R_3^{2j0} \subseteq \cdots = R_3^{\omega j0} \subseteq R_3^{\omega j\omega}.$$

This implies that for each j, we can have at most n strict inclusions among the values of R_3^{ijm} that we compute, and so for each j we take only about n steps. Since there can be up to n different j values, we take only about n^2 steps while computing the inner fixpoint, thus saving a factor of n.

The relationship between the different approximations to R_3 is shown in figure 1. The computation of least fixpoints proceeds from bottom to top, and the computation of greatest fixpoints proceeds from left to right. When computing with approximation R_1^j, we save the "frontier" values $R_3^{j-\omega}$ and use them as the initial approximations $R_3^{(j+1)-0}$ when computing with R_1^{j+1}. We have at most n strict inclusions within each vertical chain in the figure.

Note that we can build this type of table for arbitrarily nested fixpoints. Suppose, for example, that we were also computing an outer greatest fixpoint for a relation R_0. Figure 1 would correspond to a series of computations

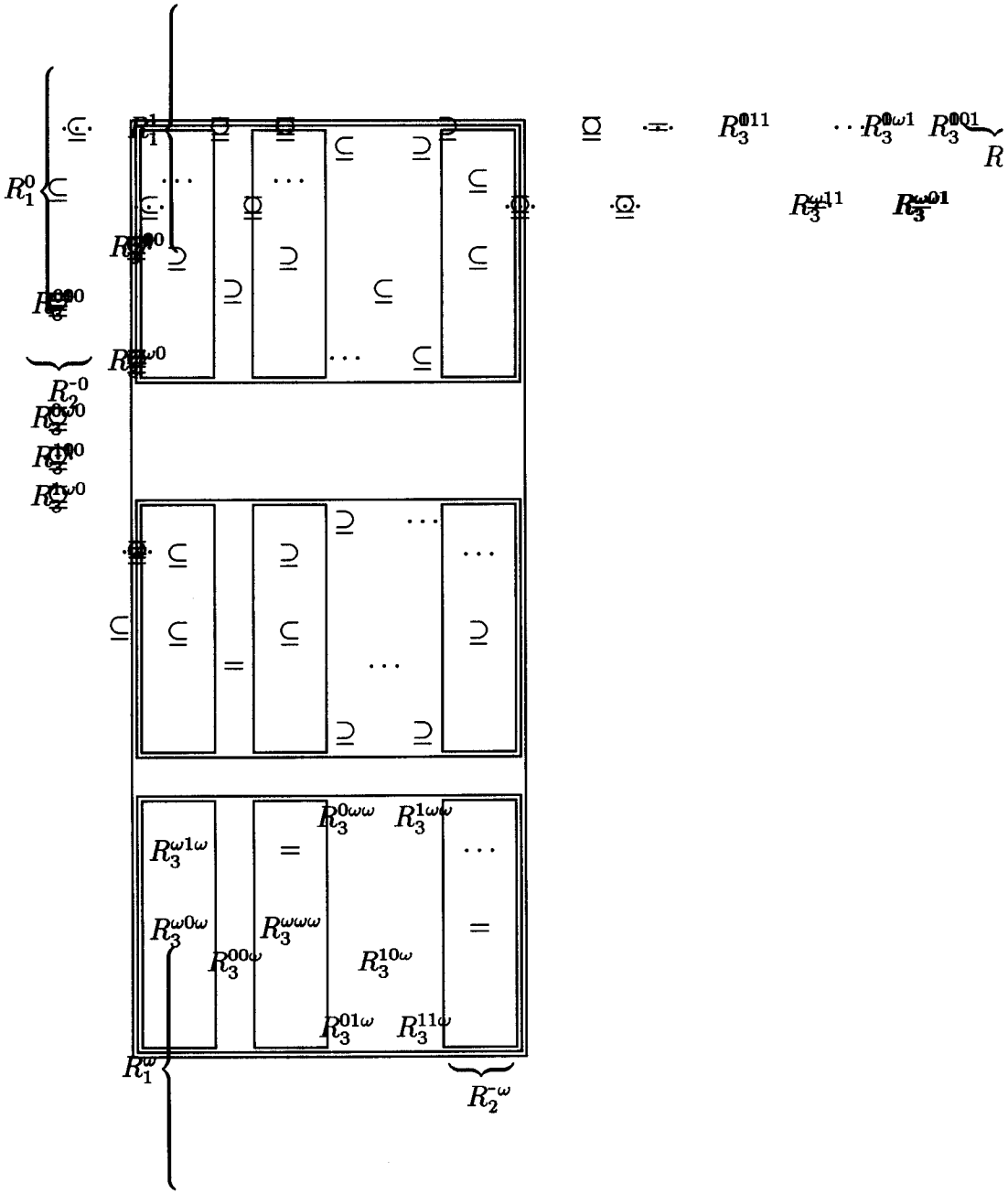

Figure 1: Relationships between approximations for R_3

with R_0 at $\top$. If we then compute the next approximation for R_0, it will be smaller than the initial approximation. Then by monotonicity, when we go through the computations for R_1, R_2, and R_3 again, we will get at each stage something smaller than during the first set of computations. For R_2, this means that we can use the frontier fixpoint values produced during the first set of computations as initial approximations when doing the second set of computations. The effect is to build a second table like the one in the figure to the right of the previous table. This process would be repeated for each new approximation for R_0. As before, we could argue that the number of strict inclusions along any chain (now running horizontally) would be bounded by n. At first it seems that these ideas must lead to a polynomial time algorithm for evaluating formulas of arbitrary alternation depth. This appears reasonable because we have a bound of n on the number of strict inclusions going both horizontally and vertically, so the number of distinct entries in a table should be about n^2. Unfortunately, this intuition is not correct. The problem arises because the chains may not "line up" due to fixpoint computations converging in less than n steps. The result is that we can only guarantee that the algorithm will take no more than about $n^{d/2}$ steps. We will give an example in section 4 that demonstrates that this bound is tight.

3.2 A simple version of the algorithm

Now we turn to the details of the algorithm. For reasons of notational simplicity, the first approach that we will describe will be one that always iterates n times when computing a fixpoint, even when convergence is achieved earlier. It will also only save frontier values for least fixpoints. Afterwards we will prove that terminating the computation of least fixpoints once convergence is achieved is allowable. This will be enough to give us the desired time bound on the algorithm. We will then give the general algorithm that saves frontier values and terminates once convergence is achieved for both types of fixpoints.

As in the discussion earlier, we will superscript relation names with vectors of iteration indices to show various approximations. Each iteration index will be a number between 0 and n, inclusive. We will let $\bar{\jmath}$ and $\bar{\imath}$ denote vectors of iteration indices. For example, we could write R_3^{000} as $R_3^{\bar{\jmath}}$ with $\bar{\jmath} = 000$. When we write such an expression, it is implicit that the length of the vector

corresponds in the right way to the depth of nesting of the fixpoint. Variables j and i will denote individual iteration indices. By the juxtaposition a vector of iteration indices with some expressions denoting individual indices, we mean the vector formed by concatenating the values of the expressions onto the end of the vector. So if $\bar{j} = 00$, then $R_3^{\bar{j}0}$ would mean R_3^{000}. The notation $\bar{0}$ will indicate a vector of all zeros of an appropriate length. Given a vector $\bar{j}$ of iteration indices, the notation $\mu(\bar{j})$ will be the vector of indices that correspond to least fixpoint variables. In our earlier example, R_1 and R_3 are the least fixpoints, so $\mu(123)$ would denote the vector 13. Similarly, $\nu(\bar{j})$ selects the indices corresponding to greatest fixpoints. We will also need a partial order on vectors of indices. The notation $\bar{\imath} \preceq \bar{\jmath}$ will mean that $\bar{\imath}$ and $\bar{\jmath}$ have the same length, that $\nu(\bar{\imath}) = \nu(\bar{\jmath})$, and that $\mu(\bar{\imath})$ is lexicographically less than or equal to $\mu(\bar{\jmath})$. If the first and third indices represent least fixpoints, then $312 \preceq 410$. The relation $\preceq$ holds between successive elements in the vertical chains in figure 1. The notation $p(\bar{j})$ will denote the immediate predecessor of $\bar{j}$ under $\preceq$. This exists whenever $\mu(\bar{j}) \neq \bar{0}$. Note that $\nu(p(\bar{j})) = \nu(\bar{j})$, and that $\mu(p(\bar{j}))$ is the immediate predecessor of $\mu(\bar{j})$ in the lexicographic ordering. The notation $h(\bar{j}, l)$ will denote the vector of length l that agrees with $\bar{j}$ on the initial indices. As an example, $h(312, 2) = 31$.

The algorithm will consist of computing the following approximations:

1. If $\sigma_k = \nu$:
   (a) $R_k^{\bar{j}0} = \top$.
   (b) $R_k^{\bar{j}(j+1)} = \psi_k(R_1^{h(\bar{j}j,1)}, \ldots, R_{k-1}^{h(\bar{j}j,k-1)}, R_k^{\bar{j}j}, R_{k+1}^{\bar{j}jn})$. (Note that $R_{k+1}^{\bar{j}jn}$ is an (inner) fixpoint since we must have convergence in at most n steps.)
2. If $\sigma_k = \mu$:
   (a) $R_k^{\bar{j}0} = \bot$ if $\mu(\bar{j}0) = \bar{0}$.
   (b) $R_k^{\bar{j}0} = R_k^{p(\bar{j}0)}$ if $\mu(\bar{j}0) \neq \bar{0}$.
   (c) $R_k^{\bar{j}(j+1)} = \psi_k(R_1^{h(\bar{j}j,1)}, \ldots, R_{k-1}^{h(\bar{j}j,k-1)}, R_k^{\bar{j}j}, R_{k+1}^{\bar{j}jn})$.

The evaluation is to be done in computation order. *Computation order* is simply the natural order of evaluation. Formally, it is the total order on vectors of iteration indices of potentially different lengths defined as follows. $\bar{\imath}$ occurs before $\bar{\jmath}$ in computation order when:

1. $\bar{\imath}$ and $\bar{\jmath}$ have the same length and $\bar{\imath}$ strictly precedes $\bar{\jmath}$ lexicographically, or

2. $\bar{\imath}$ is shorter than $\bar{\jmath}$, and $\bar{\imath}$ is lexicographically less than or equal to $h(\bar{\jmath}, |\bar{\imath}|)$, or

3. $\bar{\jmath}$ is shorter than $\bar{\imath}$, and $h(\bar{\imath}, |\bar{\jmath}|)$ strictly precedes $\bar{\jmath}$ lexicographically.

For our earlier three relation example, computation order is: 0, 00, 000, 001, ..., 00n, 01, 010, ..., 0nn, 1, 10, etc. Thus, we would compute R_1^0, R_2^{00}, R_3^{000}, ..., R_3^{00n}, R_2^{01}, etc. The result returned from the algorithm is R_1^n. Notice that $\bar{\imath} \leq \bar{\jmath}$ implies that $\bar{\imath}$ preceded $\bar{\jmath}$ in the computation order. This fact is used throughout the paper.

3.3 Proof of correctness

As a preliminary step to proving the correctness of the above algorithm, we show that the elements in a vertical chain are related by set inclusion.

Lemma 1 *For all $\bar{\imath}$, $\bar{\jmath}$ and k, if $\bar{\imath} \preceq \bar{\jmath}$, then $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$.*

Proof The proof proceeds by induction on $\bar{\jmath}$ in computation order.

1. If $\sigma_k = \nu$ and $\bar{\jmath} = \bar{\jmath}'0$, then $R_k^{\bar{\jmath}} = \top$, and hence $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$ for all $\bar{\imath} \preceq \bar{\jmath}$.

2. Suppose $\sigma_k = \nu$ and $\bar{\jmath} = \bar{\jmath}'(j+1)$. Then

$$R_k^{\bar{\jmath}} = \psi_k(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'j,k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$

   Since $\bar{\imath} \preceq \bar{\jmath}$, we know that $\bar{\jmath}$ and $\bar{\imath}$ have the same values for the ν indices. This implies that $\bar{\imath} = \bar{\imath}'(j+1)$ and that $R_k^{\bar{\imath}}$ must be defined by:

$$R_k^{\bar{\imath}} = \psi_k(R_1^{h(\bar{\imath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\imath}'j,k-1)}, R_k^{\bar{\imath}'j}, R_{k+1}^{\bar{\imath}'jn}).$$

   Now $h(\bar{\imath}'j, l) \preceq h(\bar{\jmath}'j, l)$ for $l < k$, and so by the induction hypothesis

$$R_l^{h(\bar{\imath}'j,l)} \subseteq R_l^{h(\bar{\jmath}'j,l)}.$$

   Similarly, $R_k^{\bar{\imath}'j} \subseteq R_k^{\bar{\jmath}'j}$ and $R_{k+1}^{\bar{\imath}'jn} \subseteq R_{k+1}^{\bar{\jmath}'jn}$. Since ψ_k is monotonic, $R_k^{\bar{\imath}} \subseteq R_k^{\bar{\jmath}}$.

3. If $\sigma_k = \mu$ and $\mu(\bar{j}) = \bar{0}$, then $R_k^{\bar{j}} = \perp$. Since $\bar{\imath} \preceq \bar{j}$, $\mu(\bar{\imath}) = \bar{0}$, so $R_k^{\bar{\imath}} = \perp$ also.

4. Suppose $\sigma_k = \mu$, $\mu(\bar{j}) \neq \bar{0}$, and $\bar{j}$ has the form $\bar{j}'0$. If $\bar{\imath} = \bar{j}$, then $R_k^{\bar{\imath}} = R_k^{\bar{\imath}}$. Otherwise, $\bar{\imath} \preceq p(\bar{j})$. Then by the induction hypothesis, $R_k^{\bar{\imath}} \subseteq R_k^{p(\bar{j})}$. But by definition $R_k^{\bar{j}} = R_k^{p(\bar{j})}$, and so $R_k^{\bar{\imath}} \subseteq R_k^{\bar{j}}$.

5. If $\sigma_k = \mu$ and $\bar{j}$ has the form $\bar{j}'(j+1)$, then

$$R_k^{\bar{j}} = \psi_k(R_1^{h(\bar{j}'j,1)}, \ldots, R_{k-1}^{h(\bar{j}'j,k-1)}, R_k^{\bar{j}'j}, R_{k+1}^{\bar{j}'jn}).$$

When $\bar{\imath} = \bar{j}$, we have $R_k^{\bar{\imath}} = R_k^{\bar{j}}$; otherwise $\bar{\imath} \preceq \bar{j}'j$. By the induction hypothesis $R_k^{\bar{\imath}} \subseteq R_k^{\bar{j}'j}$. Hence we just need to show that $R_k^{\bar{j}'j} \subseteq R_k^{\bar{j}'(j+1)}$. If $j > 0$, then

$$R_k^{\bar{j}'j} = \psi_k(R_1^{h(\bar{j}'(j-1),1)}, \ldots, R_{k-1}^{h(\bar{j}'(j-1),k-1)}, R_k^{\bar{j}'(j-1)}, R_{k+1}^{\bar{j}'(j-1)n}).$$

Now $h(\bar{j}'(j-1), l) = h(\bar{j}'j, l)$ for $l < k$, and hence $R_l^{h(\bar{j}'(j-1),l)} = R_l^{h(\bar{j}'j,l)}$. Since $\bar{j}'(j-1) \preceq \bar{j}'j$, the induction hypothesis implies $R_k^{\bar{j}'(j-1)} \subseteq R_k^{\bar{j}'j}$. Similarly, $R_{k+1}^{\bar{j}'(j-1)n} \subseteq R_{k+1}^{\bar{j}'jn}$. Then monotonicity of ψ_k implies that $R_k^{\bar{j}'j} \subseteq R_k^{\bar{j}}$, which is the desired result.

For the case $j = 0$, we want to show that $R_k^{\bar{j}'0} \subseteq R_k^{\bar{j}'1}$. We have two cases, depending on the value of $\mu(\bar{j}'0)$.

(a) If $\mu(\bar{j}'0) = \bar{0}$, then $R_k^{\bar{j}'0} = \perp$, and trivially $R_k^{\bar{j}'0} \subseteq R_k^{\bar{j}1}$.

(b) If $\mu(\bar{j}'0) \neq \bar{0}$, then $R_k^{\bar{j}'0} = R_k^{p(\bar{j}'0)}$. Define $\bar{j}''$ so that $\bar{j}''n = p(\bar{j}'0)$. By definition,

$$R_k^{\bar{j}'0} = \psi_k(R_1^{h(\bar{j}''(n-1),1)}, \ldots, R_{k-1}^{h(\bar{j}''(n-1),k-1)}, R_k^{\bar{j}''(n-1)}, R_{k+1}^{\bar{j}''(n-1)n}).$$

Also

$$R_k^{\bar{j}'1} = \psi_k(R_1^{h(\bar{j}'0,1)}, \ldots, R_{k-1}^{h(\bar{j}'0,k-1)}, R_k^{\bar{j}'0}, R_{k+1}^{\bar{j}'0n}).$$

Now $h(\bar{j}''(n-1), l) \preceq h(\bar{j}'0, l)$ for $l < k$, and so the induction hypothesis implies that $R_l^{h(\bar{j}''(n-1),l)} \subseteq R_l^{h(\bar{j}'0,l)}$. Similarly, $R_k^{\bar{j}''(n-1)} \subseteq R_k^{\bar{j}'0}$ and $R_{k+1}^{\bar{j}''(n-1)n} \subseteq R_{k+1}^{\bar{j}'0n}$. Then by monotonicity of ψ_k, we have $R_k^{\bar{j}'0} \subseteq R_k^{\bar{j}'1}$, which is the desired result. □

Now we show that the fixpoints are correctly computed.

Theorem 1 *For all $\bar{\jmath}$ and k, if $\bar{\jmath}$ has the form $\bar{\jmath}'n$, then*

$$R_k^{\bar{\jmath}} = \sigma_k R_k.\, \phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

In particular,

$$R_1^n = \mu R_1.\, \phi_1(R_1).$$

Proof Again, we proceed by induction on $\bar{\jmath}$ in computation order.

1. Assume $\sigma_k = \nu$. $R_k^{\bar{\jmath}}$ is computed by the sequence $R_k^{\bar{\jmath}'0} = \top$, $R_k^{\bar{\jmath}'1}$, $\ldots$, $R_k^{\bar{\jmath}'n}$. We have

   $$R_k^{\bar{\jmath}'(j+1)} = \psi_k(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'j,k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$

   Note that $h(\bar{\jmath}'j, l) = h(\bar{\jmath}', l)$ for $l < k$. Hence, we can rewrite the above equation as

   $$R_k^{\bar{\jmath}'(j+1)} = \psi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}^{\bar{\jmath}'jn}).$$

   By the induction hypothesis,

   $$R_{k+1}^{\bar{\jmath}'jn} = \sigma_{k+1} R_{k+1}.\, \phi_{k+1}(R_1^{h(\bar{\jmath}'j,1)}, \ldots, R_k^{h(\bar{\jmath}'j,k)}, R_{k+1}).$$

   Now $h(\bar{\jmath}', l) = h(\bar{\jmath}'j, l)$ for $l < k$, and we also have that $h(\bar{\jmath}'j, k) = \bar{\jmath}'j$. Substituting gives

   $$R_{k+1}^{\bar{\jmath}'jn} = \sigma_{k+1} R_{k+1}.\, \phi_{k+1}(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k^{\bar{\jmath}'j}, R_{k+1}).$$

   Now we see that $R_k^{\bar{\jmath}'0}, \ldots, R_k^{\bar{\jmath}'n}$ is just the standard iterative computation of the fixpoint. Since we must have convergence within n steps,

   $$R_k^{\bar{\jmath}'n} = \nu R_k.\, \phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

2. The case when $\sigma = \mu$ and $\mu(\bar{\jmath}') = \bar{0}$ is analogous to the previous one: we start with $R_k^{\bar{\jmath}'0} = \bot$ and compute the standard series of approximations.

3. Assume $\sigma = \mu$ and $\mu(\bar{\jmath}') \neq \bar{0}$. Here, we start the computation with $R_k^{\bar{\jmath}'0} = R_k^{p(\bar{\jmath}'0)}$. Define $\bar{\jmath}''$ so that $\bar{\jmath}''n = p(\bar{\jmath}'0)$. To ensure that $R_k^{\bar{\jmath}}$ is equal to the desired fixpoint, we first show that the initial approximation $R_k^{p(\bar{\jmath}'0)}$ is a subset of the fixpoint. By the induction hypothesis, we know that

$$R_k^{\bar{\jmath}''n} = \mu R_k.\, \phi_k(R_1^{h(\bar{\jmath}'',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}'',k-1)}, R_k).$$

We want to know that this is a subset of

$$\mu R_k.\, \phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

By the definition of $\bar{\jmath}''$, we have that $\bar{\jmath}'' \preceq \bar{\jmath}'$. Hence $h(\bar{\jmath}'', l) \preceq h(\bar{\jmath}', l)$ for all $l < k$. Using the previous lemma, we see that $R_l^{h(\bar{\jmath}'',l)} \subseteq R_l^{h(\bar{\jmath}',l)}$. Then monotonicity implies

$$R_k^{\bar{\jmath}''n} \subseteq \mu R_k.\, \phi_k(R_1^{h(\bar{\jmath}',1)}, \ldots, R_{k-1}^{h(\bar{\jmath}',k-1)}, R_k).$$

Now we simply proceed as in the previous two cases, using the fact that to compute a least fixpoint with iteration, it is only necessary that the initial approximation be a subset of the fixpoint. □

As mentioned, the algorithm given above does not stop the computation of a fixpoint when convergence is achieved. As a result, it will take about n^d steps when evaluating the alternating fixpoint formula that we are considering. To see what potential problem might arise with terminating the computation early, let us consider how an implementation of the algorithm would work. The idea will be to keep a table of frontier values for each least fixpoint. This table is indexed by the iteration indices for the enclosing greatest fixpoints, and initially all the entries are $\perp$. When computing a least fixpoint, we take the table value corresponding to the current greatest fixpoint iteration indices and use that as our initial approximation. After computing the fixpoint, we store the result back in the same spot in the table. In our earlier example with three fixpoints, the table for R_3 would have $n + 1$ entries, since there is one enclosing greatest fixpoint. During the computation of the $R_2^{0^-}$, the table entries will be used and replaced. Just after we compute R_1^1, the jth entry in the table will contain the fixpoint value R_3^{0jn}. Suppose now that we cut off the computation of $R_2^{0^-}$ as soon as convergence

is detected. Then some of the table values for R_3 will not be updated. For our three fixpoint example, this is not too complex, but in the general case it is hard to argue that "stale" table values will not cause problems. Fortunately, we can easily prove that when computing a least fixpoint, we can stop as soon as convergence is detected. To do this, we just need to show that once we obtain convergence, none of the table values for the inner fixpoints can change. More precisely, we consider a situation in which $R_k^{\bar{\jmath}(j+1)} = R_k^{\bar{\jmath}j}$ for the least fixpoint variable R_k. A table value for an inner least fixpoint variable R_l after the computation of $R_k^{\bar{\jmath}j}$ is of the form $R_l^{\bar{\jmath}'}$, where $\bar{\jmath}'$ agrees with $\bar{\jmath}j$ on the first k indices and where $\mu(\bar{\jmath}') = \mu(\bar{\jmath}j)n \ldots n$ (the indices for least fixpoint variables nested between R_k and R_l are n). The same table value after the computation of $R_k^{\bar{\jmath}(j+1)}$ is $R_l^{\bar{\jmath}''}$, where $\bar{\jmath}''$ differs from $\bar{\jmath}'$ only at position k, and at that position, $\bar{\jmath}''$ has $j+1$ instead of j. We claim that these two table values must be equal.

Theorem 2 *Assume $\sigma_k = \mu$ and $R_k^{\bar{\jmath}(j+1)} = R_k^{\bar{\jmath}j}$. Then for all $l > k$ and $\bar{\jmath}'$ and $\bar{\jmath}''$ such that:*

1. $h(\bar{\jmath}', k-1) = h(\bar{\jmath}'', k-1) = \bar{\jmath}$,
2. $\mu(\bar{\jmath}') = \mu(\bar{\jmath}j)n \ldots n$,
3. $\mu(\bar{\jmath}'') = \mu(\bar{\jmath}(j+1))n \ldots n$, *and*
4. $\nu(\bar{\jmath}') = \nu(\bar{\jmath}'')$,

we have $R_l^{\bar{\jmath}'} = R_l^{\bar{\jmath}''}$. (Note that we are not requiring $\sigma_l = \nu$ here.)

Proof We proceed by induction on $\bar{\jmath}'$ in computation order. Note that since $\bar{\jmath}'$ and $\bar{\jmath}''$ differ only on the index for R_k, fixing $\bar{\jmath}'$ also fixes $\bar{\jmath}''$.

1. Assume $\sigma_l = \nu$ and $\bar{\jmath}'$ has the form $\bar{\imath}'0$. Then $\bar{\jmath}''$ has the form $\bar{\imath}''0$, and so $R_l^{\bar{\jmath}'} = R_l^{\bar{\jmath}''} = \top$.
2. Assume $\sigma_l = \nu$ and $\bar{\jmath}'$ has the form $\bar{\imath}'(m+1)$. Let $\bar{\jmath}''$ have the form $\bar{\imath}''(m+1)$. We have

$$R_l^{\bar{\jmath}'} = \psi_l(R_1^{h(\bar{\imath}'m,1)}, \ldots, R_{l-1}^{h(\bar{\imath}'m,l-1)}, R_l^{\bar{\imath}'m}, R_{l+1}^{\bar{\imath}'mn})$$

and

$$R_l^{\bar{j}''} = \psi_l(R_1^{h(\bar{\imath}''m,1)}, \ldots, R_{l-1}^{h(\bar{\imath}''m,l-1)}, R_l^{\bar{\imath}''m}, R_{l+1}^{\bar{\imath}''mn}).$$

Now $h(\bar{\imath}'m, o) = h(\bar{\imath}''m, o)$ for $o < k$. By hypothesis, $R_k^{h(\bar{\imath}'m,k)} = R_k^{h(\bar{\imath}''m,k)}$(we have that $h(\bar{\imath}'m, k) = \bar{\jmath}j$ and $h(\bar{\imath}''m, k) = \bar{\jmath}(j+1)$) By the induction hypothesis, $R_o^{h(\bar{\imath}'m,o)} = R_o^{h(\bar{\imath}''m,o)}$ for $k < o < l$. Also by the induction hypothesis, $R_l^{\bar{\imath}'m} = R_l^{\bar{\imath}''m}$ and $R_{l+1}^{\bar{\imath}'mn} = R_{l+1}^{\bar{\imath}''mn}$. Hence $R_l^{\bar{j}'} = R_l^{\bar{j}''}$.

3. Assume $\sigma_l = \mu$. Let $\bar{\imath}'$ and $\bar{\imath}''$ be chosen so that $\bar{\jmath}' = \bar{\imath}'n$ and $\bar{\jmath}'' = \bar{\imath}''n$. By the previous theorem, we have

$$R_l^{\bar{j}'} = \mu R_l.\, \phi_l(R_1^{h(\bar{\imath}',1)}, \ldots, R_{l-1}^{h(\bar{\imath}',l-1)}, R_l)$$

and

$$R_l^{\bar{j}''} = \mu R_l.\, \phi_l(R_1^{h(\bar{\imath}'',1)}, \ldots, R_{l-1}^{h(\bar{\imath}'',l-1)}, R_l).$$

As in the previous case, we have: $h(\bar{\imath}', o) = h(\bar{\imath}'', o)$ for $o < k$; $R_k^{h(\bar{\imath}',k)} = R_k^{h(\bar{\imath}'',k)}$; and $R_o^{h(\bar{\imath}',o)} = R_o^{h(\bar{\imath}'',o)}$ for $k < o < l$. Hence $R_l^{\bar{j}'} = R_l^{\bar{j}''}$. □

Note that this result, combined with our earlier correctness theorem and monotonicity lemma, implies that once $R_k^{\bar{j}(j+1)} = R_k^{\bar{j}j}$, then $R_l^{\bar{\imath}} = R_l^{\bar{\imath}'}$ for any $\bar{\imath}$ and $\bar{\imath}'$ with $\nu(\bar{\imath}) = \nu(\bar{\imath}')$ and with $\bar{\imath}$ and $\bar{\imath}'$ between $\bar{\jmath}(j+1)$ and $\bar{\jmath}n \ldots n$ in computation order. In other words, no table values will change if we keep iterating for R_k, and hence we can safely stop this part of computation.

3.4 Complexity analysis

Now we analyze the complexity of the algorithm. We will just be counting the number of approximations produced to avoid details such as how relations are represented. Let T_k denote the maximum number of approximations that can be produced for R_k. Obviously $T_1 = n + 1$ and $T_2 = (n+1)^2$. If T_{k+1} is a greatest fixpoint, then each time we evaluate it, we produce $n + 1$ approximations. The fixpoint can be evaluated at most T_k times, so when $\sigma_{k+1} = \nu$, $T_{k+1} \leq (n+1)T_k$. For $\sigma_{k+1} = \mu$, the fixpoint can also be evaluated at most T_k times. For this fixpoint, there are $(n+1)^{\lfloor (k+1)/2 \rfloor}$ table entries (since there are $\lfloor (k+1)/2 \rfloor$ enclosing greatest fixpoints). Over all the evaluations, each of these increases monotonically, so the total number

of changes in these table values is bounded by $n(n+1)^{\lfloor (k+1)/2 \rfloor}$. During each of the T_k evaluations, we can also make one extra step to detect convergence. The overall number of steps is bounded by the sum of these numbers: $T_{k+1} \leq T_k + n(n+1)^{\lfloor (k+1)/2 \rfloor} \leq T_k + (n+1)^{1+\lfloor (k+1)/2 \rfloor}$. In our fixpoint formula, k is even for least fixpoints, so this simplifies to $T_{k+1} \leq T_k + (n+1)^{(k+2)/2}$. Overall, we get

$$T_{2k} \leq k(n+1)^{k+1}, \qquad T_{2k+1} \leq (k+1)(n+1)^{k+1},$$

and so

$$T_k \leq \lceil k/2 \rceil (n+1)^{1+\lfloor k/2 \rfloor}.$$

Summing the T_k for all d of the fixpoints in our formula, we get a total of $O(d^2(n+1)^{1+\lfloor d/2 \rfloor})$ steps to do the evaluation.

3.5 The general algorithm

Figure 2 is a pseudo-code version of the algorithm that works for formulas of arbitrary form, and that saves information and stops iterating after detecting termination for both types of fixpoints. The potential problems with stale table values are avoided by storing frontier values in queues; the length of a queue tells us the number of approximations were produced when computing the frontier represented by the queue. We will not give a proof of correctness here. In the algorithm, the frontier values for starting the fixpoint computations are stored on stacks. There are two stacks for each fixpoint variable R, one associated with the current frontier (I_R) and one associated with the frontier being constructed (F_R). Each stack element is either a set of states (representing an earlier fixpoint value), or a queue (representing a frontier). The queue elements may themselves be either queues (representing sub-frontiers) or sets of states. We will write stacks using angle brackets and queues using square brackets, with the top of a stack and the head of a queue being on the left. Initially, the I_R stack for a top-level fixpoint variable R holds either $\top$ or $\bot$, depending on whether R is a greatest or least fixpoint. The stack I_R for a least fixpoint variable R nested inside k greatest fixpoints holds $\bot$ nested inside k queues ($[[\ldots[[\bot]]\ldots]]$). The initial value for a stack corresponding to a greatest fixpoint variable nested inside a number of least fixpoints is similarly defined.

As an example of the algorithm's operation, we consider a formula with three fixpoints: $\mu R_1.\, \nu R_2.\, \mu R_3.\, (\cdots)$. Initially, $I_{R_1} = \langle \bot \rangle$, $I_{R_2} = \langle [\top] \rangle$, and

```
1  function eval(φ)

2  Handle base cases, logical operations, etc.
3  if φ = μR.ψ(R) then
4       set R to the top element of I_R
5       for each inner ν variable R'
6            push [] on F_R'
7       repeat
8            for each inner ν variable R'
9                 let Q be the queue on top of I_R'
10                dequeue e from Q
11                if Q is now empty, enqueue e again
12                push e on I_R'
13           R_old := R
14           R := eval(ψ)
15           for each inner ν variable R'
16                pop e from F_R'
17                let Q be the queue on top of F_R'
18                enqueue e in Q
19                pop I_R'
20           if R ≠ R_old then
21                for each inner μ variable R'
22                     pop I_R'
23                     pop e from F_R'
24                     push e on I_R'
25      until R = R_old
26      push R on F_R
27      return R
28 if φ = νR.ψ(R) then
29      Analogous code to the above
```

Figure 2: Pseudo-code for the general algorithm

$I_{R_3} = \langle [\bot] \rangle$. All the stacks F_{R_1}, F_{R_2}, and F_{R_3} are empty. The computation proceeds as shown in figure 3. In the figure, $\mu R_1 : \bot$ denotes a call to the evaluation routine for the formula $\mu R_1 . (\cdots)$ with $\bot$ on the top of the stack I_{R_1} (i.e., the evaluation of the fixpoint starting from $\bot$). The notations "start R_1" and "end R_1" denote the start of an iteration for computing R_1 and the end of an iteration, respectively. The notation "return $R_2^{0\omega}$" indicates returning a fixpoint value for R_2. Finally, $\mu R_3 : \bot \rightarrow R_3^{00\omega}$ denotes the evaluation of $\mu R_3 \ldots$ starting with $\bot$ and yielding $R_3^{00\omega}$ as the result. The figure shows how the state of the stacks evolves during the computation. During each iteration for the fixpoint $\mu R . (\cdots)$, we pull out the next sub-frontier for the inner ν variables (lines 8–12), recursively evaluate the inner fixpoints, and build up sub-frontiers for subsequent evaluations (lines 15–19). If the computation of R has not yet converged, we discard the old frontiers for the inner μ fixpoints and replace them with the new frontiers that have been built up (lines 21–24). Note that with two successive μ fixpoints, this simply results in picking up the inner fixpoint from the previous stopping point. Hence the algorithm also makes use of Emerson and Lei's observation [13].

4 A worst-case example

We now give an example that shows that even the general algorithm (figure 2) may take about $n^{d/2}$ steps. Consider the transition system shown in figure 4. There is a transition on a from state s_{i+1} to state s_i for all $i \geq 0$, and there is a transition on b from state s_0 to state s_i for all $i \geq 0$. We also assume there is an atomic proposition that is true only in state s_0. We will abuse notation and denote this proposition by s_0 as well.

For the formula to evaluate, we take:

$$
\begin{aligned}
F_1 &\equiv \bot \wedge \mu R_1 . s_0 \vee \langle a \rangle R_1 \\
&\qquad\qquad \vee (\bot \wedge \nu R_1' . \langle a \rangle R_1' \vee F_2) \\
F_2 &\equiv \bot \wedge \mu R_2 . s_0 \wedge [b](R_1 \vee R_1') \vee \langle a \rangle R_2 \\
&\qquad\qquad \vee (\bot \wedge \nu R_2' . \langle b \rangle (R_1 \wedge R_1') \vee \langle a \rangle R_2' \vee F_3) \\
&\;\;\vdots \\
F_{i+1} &\equiv \bot \wedge \mu R_{i+1} . s_0 \wedge [b](R_i \vee R_i') \vee \langle a \rangle R_{i+1} \\
&\qquad\qquad \vee (\bot \wedge \nu R_{i+1}' . \langle b \rangle (R_i \wedge R_i') \vee \langle a \rangle R_{i+1}' \vee F_{i+2})
\end{aligned}
$$

	I_{R_1}	F_{R_1}	I_{R_2}	F_{R_2}	I_{R_3}	F_{R_3}
start	$\langle\bot\rangle$	$\langle\rangle$	$\langle[\top]\rangle$	$\langle\rangle$	$\langle[\bot]\rangle$	$\langle\rangle$
$\mu R_1\colon \bot$	$\langle\bot\rangle$	$\langle\rangle$	$\langle[\top]\rangle$	$\langle[]\rangle$	$\langle[\bot]\rangle$	$\langle\rangle$
start R_1	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle[\bot]\rangle$	$\langle\rangle$
$\nu R_2\colon \top$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle[\bot]\rangle$	$\langle[]\rangle$
start R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle\bot,[\bot]\rangle$	$\langle[]\rangle$
$\mu R_3\colon \bot \to R_3^{00\omega}$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle\bot,[\bot]\rangle$	$\langle R_3^{00\omega},[]\rangle$
end R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle[\bot]\rangle$	$\langle[R_3^{00\omega}]\rangle$
start R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle\bot,[\bot]\rangle$	$\langle[R_3^{00\omega}]\rangle$
$\mu R_3\colon \bot \to R_3^{01\omega}$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle\bot,[\bot]\rangle$	$\langle R_3^{01\omega},[R_3^{00\omega}]\rangle$
end R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle[\bot]\rangle$	$\langle[R_3^{00\omega},R_3^{01\omega}]\rangle$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\vdots$	$\vdots$
$\mu R_3\colon \bot \to R_3^{0\omega\omega}$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle\bot,[\bot]\rangle$	$\langle R_3^{0\omega\omega},[R_3^{00\omega},\ldots]\rangle$
end R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[]\rangle$	$\langle[\bot]\rangle$	$\langle[R_3^{00\omega},\ldots,R_3^{0\omega\omega}]\rangle$
return $R_2^{0\omega}$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle R_2^{0\omega},[]\rangle$	$\langle[\bot]\rangle$	$\langle[R_3^{00\omega},\ldots,R_3^{0\omega\omega}]\rangle$
end R_1	$\langle\bot\rangle$	$\langle\rangle$	$\langle[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle[R_3^{00\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle\rangle$
start R_1	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle[R_3^{00\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle\rangle$
$\nu R_2\colon \top$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle[R_3^{00\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle[]\rangle$
start R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle R_3^{00\omega},[R_3^{01\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle[]\rangle$
$\mu R_3\colon R_3^{00\omega} \to R_3^{10\omega}$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle R_3^{00\omega},[R_3^{01\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle R_3^{10\omega},[]\rangle$
end R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle[R_3^{01\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle[R_3^{10\omega}]\rangle$
start R_2	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle R_3^{01\omega},[R_3^{02\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle[R_3^{10\omega}]\rangle$
$\mu R_3\colon R_3^{01\omega} \to R_3^{11\omega}$	$\langle\bot\rangle$	$\langle\rangle$	$\langle\top,[\top]\rangle$	$\langle[R_2^{0\omega}]\rangle$	$\langle R_3^{01\omega},[R_3^{02\omega},\ldots,R_3^{0\omega\omega}]\rangle$	$\langle R_3^{11\omega},[R_3^{10\omega}]\rangle$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\vdots$	$\vdots$
end R_1	$\langle\bot\rangle$	$\langle\rangle$	$\langle[\top]\rangle$	$\langle[R_2^{0\omega},\ldots,R_2^{\omega\omega}]\rangle$	$\cdots$	$\langle[R_3^{\omega 0\omega},\ldots,R_3^{\omega\omega\omega}]\rangle$
return R_1^{ω}	$\langle\bot\rangle$	$\langle R_1^{\omega}\rangle$	$\langle[\top]\rangle$	$\langle[R_2^{0\omega},\ldots,R_2^{\omega\omega}]\rangle$	$\cdots$	$\langle[R_3^{\omega 0\omega},\ldots,R_3^{\omega\omega\omega}]\rangle$

Figure 3: Example computation of the algorithm in figure 2

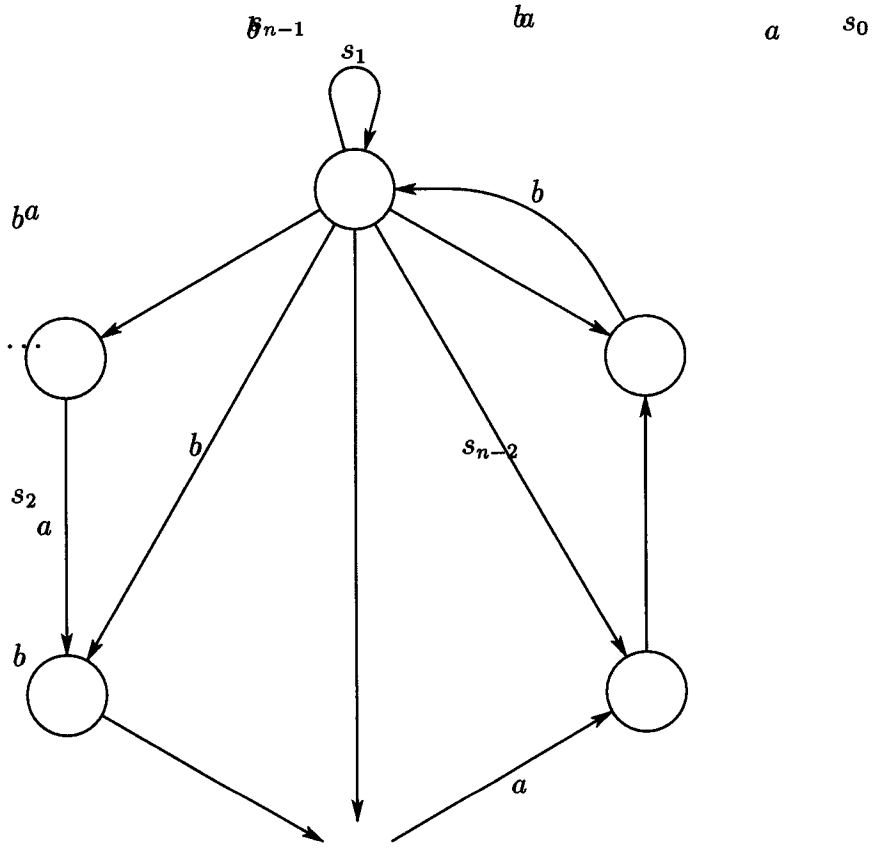

Figure 4: Transition system for worst-case example

$$\vdots$$
$$F_q \equiv \bot.$$

This formula is obviously equivalent to a much simpler formula, but it will serve to illustrate the kind of frontier structure that can cause exponential behavior. To understand how this expression works, consider F_{i+1}. When the union of R_i and R'_i is $\top$, the fixpoint for R_{i+1} will be $\top$ as well, and otherwise it will be $\bot$. Similarly, when R_i and R'_i have a nonempty intersection, the fixpoint for R'_{i+1} will be $\top$, and otherwise it will be $\bot$. Consider the following pairs of (R_i, R'_i) values:

$$(\bot, \top),\ (\{s_0\}, \top - \{s_0\}),\ \ldots,\ (\top - \{s_{n-1}\}, \{s_{n-1}\}),\ (\top, \bot).$$

If we imagine storing all the fixpoint approximations in tables as in figure 1, then we can think of these pairs as analogous to "diagonal" elements. Above the diagonal (i.e., if we add elements to either set in the pair), the union of R_i and R'_i is $\top$ and they also have a nonempty intersection. Hence the fixpoints for both R_{i+1} and R'_{i+1} will be $\top$. Below the diagonal (if we remove elements from either set in the pair), the union is not $\top$ and the intersection is empty. Here, both R_{i+1} and R'_{i+1} will have a fixpoint of $\bot$. On the diagonal, R_{i+1} has a fixpoint of $\top$ and R'_{i+1} has a fixpoint of $\bot$. Thus, if R_{i+1} starts at $\bot$ and R'_{i+1} starts at $\top$, then we will do about n^2 evaluations of F_{i+2} while computing these two fixpoints. Of these n^2 evaluations, n of them will represent diagonal values for (R_{i+1}, R'_{i+1}), and for these elements, we will get the same effect within the computation of F_{i+2}. Let us say that a configuration for all the fixpoint variables R_1, R'_1, R_2, $\ldots$, is a "diagonal configuration" when each pair (R_i, R'_i) is a diagonal element. Then adding another pair of fixpoints will increase the number of possible diagonal configurations by a factor of n. To see that all these diagonal configurations will occur during the computation, we note that any two diagonal configurations for the variables R_1, R'_1, $\ldots$, R_i, R'_i are incomparable with respect to set containment. Hence computing the fixpoints for R_{i+1} and R'_{i+1} for one of these configurations will give us no information about the fixpoint values for any of the other configurations. This implies that the computations for R_{i+1} and R'_{i+1} will always start from $\bot$ and $\top$, respectively (for diagonal configurations of R_1, $\ldots$, R'_i), and thus that all diagonal configurations for R_1, $\ldots$, R_{i+1}, R'_{i+1} will occur. We can analyze exactly the number of times

during the computation in which a fixpoint variable iterates from $\bot$ to $\top$ or from $\top$ to $\bot$ while running the algorithm of figure 2. For R_i, this occurs $(n+1)^{i-1}$ times. For R'_i, it happens $(n+1)^i$ times. This implies that F_q is evaluated $(n+1)^q$ times. The alternation depth of the formula is $d = 2q-2$, so this is $(n+1)^{1+d/2}$ steps.

5 Conclusion

We have presented a new algorithm for evaluating a formula in the proposition μ-calculus with respect to a finite transition system. Our algorithm takes about $n^{d/2}$ steps, where d is the alternation depth of the formula. The best previously known algorithms required about n^d steps. A straightforward implementation of our algorithm would require an extra factor of n or so for bookkeeping and set manipulations, but we believe that methods such as those used by Cleaveland, Klein, Steffen, and Andersen [1, 9, 10] could be used to reduce this extra complexity. It is not as clear whether efficient local procedures can be developed that make use of our ideas, but this is an interesting question. It would also be interesting to see whether it is possible to make even more use of monotonicity considerations. This is certainly possible, at least conceptually. Suppose, for example, that $\mu R.\, \phi(R, R_1, \ldots, R_k)$ is a subformula of the formula that we are evaluating. We could imagine saving every fixpoint value computed for R together with the values for the parameters $R_1, \ldots, R_k$. Then before evaluating the subformula for some new set of parameters, we take the union of all previous fixpoints for R for which the parameters used were all subsets of the current parameters. This becomes the initial approximation. Note though, that an algorithm based on this idea would still take about $n^d/2$ steps on the example described in section 4. Thus, pure monotonicity considerations seem unlikely to lead to a polynomial time algorithm.

This suggests another line of research: trying to place lower bounds on the complexity of the evaluation process. Suppose we cast the problem as a language recognition problem, where a string gives a formula, a transition system, and a designated state, and a string is in the language if the designated state of the transition system is in the interpretation of the formula. We note that the recognition problem is in NP. The basic idea for showing this is to use an algorithm that computes least fixpoints by iterating, and

that guesses greatest fixpoints. The guess for a greatest fixpoint can be easily checked to see that it really is a fixpoint. Further, while we cannot verify that it is the greatest fixpoint, we know that the greatest fixpoint must contain any verified guess. Then by monotonicity, the final value computed by this nondeterministic algorithm will be a subset of the real interpretation of the formula. If the designated state is in the computed result, the string is accepted, and if not, the string is rejected. When the greatest fixpoints are correctly guessed, the string is accepted iff the designated state is in the real interpretation of the formula. Also note that we can negate formulas (the syntax we gave allows negation only at the atomic proposition level, but we can always drive negations inwards using semantic equivalences). Hence the complexity of recognizing the language is the same as the complexity of recognizing the complement of the language. Thus, the problem is in the intersection of NP and co-NP. This suggests that it would be very difficult to prove that there is no polynomial time algorithm for the problem. However, it might be possible to prove something about a restricted class of algorithms. A natural class to consider is "oblivious" algorithms. These are methods that only make use of the structure of the nesting of fixpoints, and perhaps the fixpoint values. Formally, given a formula like $\mu R_1.\psi_1(R_1, \nu R_2.\psi_2(R_1, R_2))$, we would view ψ_1 and ψ_2 as being given by oracles. The complexity of an algorithm would be measured in the number of calls to the oracles. This is a natural class of methods. For example, both Emerson and Lei's original algorithm and our new one can be viewed as members of this class. A proof that no algorithm of this class can make do with just a polynomial number of oracle queries would imply that any polynomial time algorithm would have to do something clever based on the structure of the formula.

References

[1] H. R. Andersen. Model checking and boolean graphs. In B. Krieg-Bruckner, editor, *Proceedings of the Fourth European Symposium on Programming*, volume 582 of *Lecture Notes in Computer Science*. Springer-Verlag, February 1992.

[2] G. V. Bochmann and D. K. Probst, editors. *Proceedings of the Fourth Workshop on Computer-Aided Verification*, volume 663 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1992.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, June 1990.

[5] E. M. Clarke, I. A. Draghicescu, and R. P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. In A. Arnold and N. D. Jones, editors, *Proceedings of the 15th Colloquium on Trees in Algebra and Programming*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1990.

[6] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

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

[9] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In Bochmann and Probst [2].

[10] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2(2):121–147, April 1993.

[11] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Proceedings of the 1989 International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1989.

[12] J. W. de Bakker. *Mathematical Theory of Program Correctness.* Prentice-Hall, 1980.

[13] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the First Annual Symposium on Logic in Computer Science.* IEEE Computer Society Press, June 1986.

[14] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[15] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.

[16] K. G. Larsen. Efficient local correctness checking. In Bochmann and Probst [2].

[17] A. Mader. Tableau recycling. In Bochmann and Probst [2].

[18] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, 1992.

[19] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[20] D. M. R. Park. Finiteness is mu-ineffable. Theory of Computation Report No. 3, University of Warwick, 1974.

[21] C. Stirling and D. J. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, October 1991.

[22] A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[23] G. Winskel. Model checking in the modal ν-calculus. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming*, 1989.