# An Improved Algorithm for the Evaluation of Fixpoint Expressions*

David E. Long[1], Anca Browne[2], Edmund M. Clarke[2],
Somesh Jha[2], Wilfredo R. Marrero[2]

[1] AT&T Bell Laboratories, Murray Hill, NJ 07974
[2] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

**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). Hence, fixpoint calculi such as the propositional $\mu$-calculus 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 a fixpoint calculus over a given model. A naive algorithm for this task 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 reduced 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. Thus, the time required by our method is only about the square root of the time required by the earlier algorithms.

## 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] and PDL [12], methods for computing strong and weak bisimulation equivalence in CCS [16], and language containment and emptiness algorithms for $\omega$-automata [5]. Approaches based on fixpoint logics such as the

* 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, and in part by the Semiconductor Research Corportation (SRC) under contract 92-DJ-294, and in part by the National Science Foundation under contract number CCR-9217549. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory, the U. S. Government, the Semiconductor Research Corporation, or the National Science Foundation. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

propositional $\mu$-calculus [13] are tied even more directly to fixpoint computation. With the increasing use of binary decision diagrams (BDDs) [3] for finite-state verification [4, 10], algorithms based on set manipulations and fixpoints have become even more important, since methods that require the manipulation of individual states do not take advantage of the 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.

Our ideas are independent of the particular fixpoint calculus used, but for concreteness, we will be using the propositional $\mu$-calculus of Kozen [13]. 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, like those developed by Cleaveland [7], Stirling and Walker [17], and Winskel [19], 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, though recent work by Andersen [1], Larsen [14], and Mader [15] has improved the bounds. 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 [11] improved on this by observing that the complexity of evaluating a formula really depends only on the number of alternations $d$ of least and greatest fixpoints. Emerson and Lei gave an algorithm requiring only about $n^d$ steps. Subsequent work by Cleaveland, Klein, Steffen, and Andersen [1, 8, 9] has reduced the overhead, 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 $d$ alternations.

The remainder of this paper is organized as follows. Section 2 summarizes the syntax and semantics of the propositional $\mu$-calculus and reviews Emerson and Lei's work. In Sect. 3 we present our new algorithm and discuss its complexity. We consider some open questions and directions for future research in Sect. 4.

## 2 The Propositional $\mu$-Calculus

In the propositional $\mu$-calculus, formulas are built up from:

1. atomic propositions $p, p_1, p_2, \ldots$;

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 $\nu R_i. (\cdots)$.

Formulas in this calculus are interpreted relative to a transition system that consists of:

1. a nonempty set of states $\top$;
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 $\phi$ holds." $[\cdot]$ is the dual of $\langle \cdot \rangle$; for $[a]\phi$, the intended meaning is that "$\phi$ holds in all states reachable (in one step) by executing $a$." The $\mu$ and $\nu$ operators are used to express least and greatest fixpoints, respectively. We could also allow negation (with some restrictions); in this case, greatest fixpoints could be expressed using the duality $\nu R. \phi(R) = \neg \mu R. \neg \phi(\neg R)$. To emphasize this duality, we write the empty set of states as $\bot$.

Formally, a formula $\phi$ over the 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 denote this predicate transformer by $\phi^M$. $\phi^M$ is defined inductively by giving its value for the arguments $S_1$, ..., $S_k$. We write this value as $\phi^M(\bar{S})$.

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 the least fixpoint of the predicate transformer $\tau : 2^\top \rightarrow 2^\top$ defined by:

$$\tau(S) = \phi^M(S, S_1, \ldots, S_k) \;,$$

   where the first parameter of $\phi^M$ 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, and so the fixpoints are guaranteed to be well-defined. Formally, each possible $\tau$ is monotonic ($S \subseteq S'$ implies $\tau(S) \subseteq \tau(S')$). This is enough to ensure the existence of the fixpoints [18]. For finite transition systems, the fixpoints can be computed by iterative evaluation. More precisely, for some $i \leq n = |\top|$, the fixpoint is equal to $\tau^i(\bot)$ (for a least fixpoint) or $\tau^i(\top)$ (for a greatest fixpoint). In what follows, we will often abuse notation and identify the formula $\phi$ with its meaning $\phi^M$.

Since we will be using the concept of alternation depth, we briefly summarize Emerson and Lei's observations [11]. 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 $\tau$ taking one set (the value of $R_1$) to another (the value of $\mu R_2.\,(\cdots)$). When evaluating the outer fixpoint, we start with the approximation $\perp$ and then compute $\tau(\perp)$. Now $R_1$ is increased (say to $S_1$), and we want to compute the least fixpoint $\tau(S_1)$. Since $\perp \subseteq S_1$, we know that $\tau(\perp) \subseteq \tau(S_1)$. To compute a least fixpoint, it is enough to start iterating with any approximation known to be below the fixpoint. This implies that we can start iterating with $\tau(\perp)$ instead of $\perp$. 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 $n$ 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 showed that this type of simplification makes it possible to evaluate a formula $\phi$ 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. For the formula above, the alternation depth is 1, so $n^1$ steps suffice. Note: throughout this paper, when we speak of the number of steps used by an algorithm, we mean the number of fixpoint approximations produced during the evaluation process. Thus, we avoid details of how sets and relations are represented and manipulated.

## 3 The Algorithm

We first illustrate the essential idea behind our new algorithm on a formula involving three fixpoints (with alternation depth three):

$$\mu R_1.\,\psi_1(R_1, \nu R_2.\,\psi_2(R_1, R_2, \mu R_3.\,\psi_3(R_1, R_2, R_3)))\ . \tag{1}$$

To compute the outer fixpoint, we start with $R_1 = \perp$, $R_2 = \top$ and $R_3 = \perp$. Call these values $R_1^0$, $R_2^{00}$, and $R_3^{000}$ respectively. The superscript on $R_i$ gives the iteration indices for the fixpoints involving $R_1, \ldots, R_i$. So $R_3^{000}$ means that all three fixpoints are at their the initial approximations. We then iterate to compute the inner fixpoint; call the value of this fixpoint $R_3^{00\omega}$. (The $\omega$ stands for whatever number of steps were needed for the fixpoint iteration to converge.) 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 then we 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 $R_3^{00\omega}$ (i.e., we take $R_3^{100} = R_3^{00\omega}$). Since $R_3^{00\omega}$ is in general larger than $\bot$, we obtain faster convergence. Also note that

$$R_2^{01} = \psi_2(R_1^0, R_2^{00}, R_3^{00\omega})$$

and

$$R_2^{11} = \psi_2(R_1^1, R_2^{10}, R_3^{10\omega}) \ .$$

Since $R_1^0 \subseteq R_1^1$, $R_2^{00} \subseteq R_2^{10}$, 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 $R_3^{11\omega}$. Thus, we will use $R_3^{01\omega}$ for the approximation $R_3^{110}$. In general, we can start computing $R_3^{1j\omega}$ from $R_3^{1j0} = R_3^{0j\omega}$. Eventually we find another fixpoint for $R_2$. Then, once we compute $R_1^2$ (or in general, $R_1^{k+1}$), we can use the fixpoints $R_3^{1j\omega}$ ($R_3^{kj\omega}$) as the initial approximations $R_3^{2j0}$ ($R_3^{(k+1)j0}$) to $R_3^{2j\omega}$ ($R_3^{(k+1)j\omega}$).

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 $n$. (Again, we are using "number of steps" to mean the number of fixpoint approximations produced.)

The relationship between the different approximations to $R_3$ is shown in Fig. 1. The computation of least fixpoints proceeds from bottom to top, and the computation of greatest fixpoints proceeds from left to right. The chain mentioned above corresponds to one of the vertical columns in this figure. When computing with approximation $R_1^j$, we save the "frontier" values $R_3^{j0\omega}, \ldots, R_3^{j\omega\omega}$ and use them as the initial approximations $R_3^{(j+1)00}, \ldots, R_3^{(j+1)\omega 0}$ when computing with $R_1^{j+1}$. We have at most $n$ strict inclusions within each vertical chain in the figure.

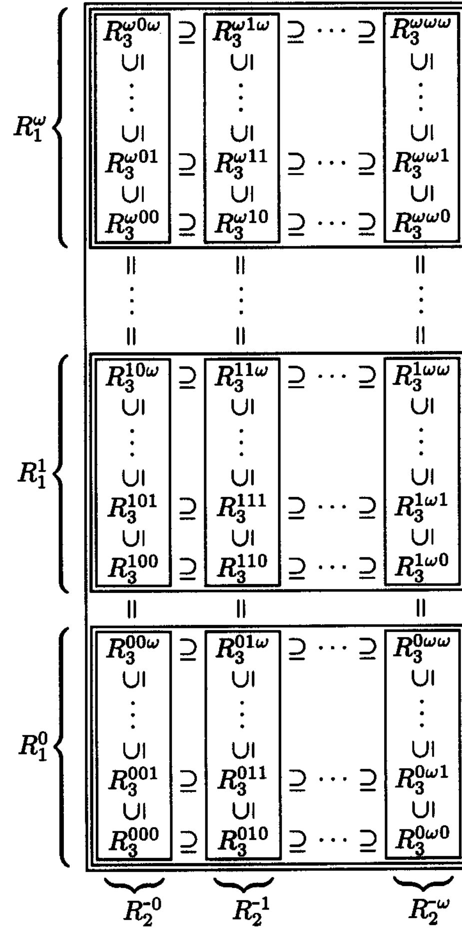



**Fig. 1.** Relationships between approximations for $R_3$

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 $R_0$. Fig. 1 would correspond to a series of computations 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.

To argue in more detail that $n^{d/2}$ steps suffices to evaluate a formula with alternation depth $d$, we now present a special-case algorithm. This algorithm

handles strictly alternating fixpoints and only saves frontier values for least fixpoints. Assume that the formula that we wish to evaluate has the form:

$$\begin{aligned} F_1 &\equiv \mu R_1 .\, \psi_1(\nu R_1' .\, \psi_1'(F_2)) \\ F_2 &\equiv \mu R_2 .\, \psi_2(\nu R_2' .\, \psi_1'(F_3)) \\ &\vdots \\ F_q &\equiv \mu R_q .\, \psi_q(\nu R_q' .\, \psi_q'(R_1, R_1', \ldots, R_q, R_q')) \ , \end{aligned}$$

where $\equiv$ denotes syntactic equality. This formula has alternation depth $d = 2q$.

The special-case algorithm is given in Fig. 2. The algorithm uses an array $A_i$ to store the frontier values for the fixpoint variable $R_i$. The array $A_i$ is indexed by iteration indices for all the greatest fixpoints enclosing $R_i$. There are $i - 1$ of such enclosing fixpoints. Each iteration index is between 0 and $n$ (inclusive), and so $A_i$ has $(n + 1)^{i-1}$ entries. Initially, all array values are $\perp$. When evaluating $R_i$, we start with the array value indicated by the current iteration indices for the enclosing greatest fixpoints, and iterate until convergence. At the end of the iteration, the array holds the fixpoint value. For each greatest fixpoint variable $R_i'$, we have an associated iteration index $j_i$. When evaluating $R_i'$, we start with $\top$ and iterate $n + 1$ times (even if convergence is achieved earlier). We update $j_i$ after each iteration.

```
function eval(φ)
Handle atomic propositions, logical operations, etc.
if φ = μRᵢ. ψᵢ(···) then
    Rᵢ := Aᵢ[j₁, ..., jᵢ₋₁]
    repeat
        Oᵢ := Rᵢ
        Rᵢ := eval(ψᵢ)
        Aᵢ[j₁, ..., jᵢ₋₁] := Rᵢ
    until Rᵢ = Oᵢ
    return Rᵢ
else if φ = νR'ᵢ. ψ'ᵢ(···)
    R'ᵢ := ⊤
    for jᵢ from 0 to n
        R'ᵢ := eval(ψ'ᵢ)
    endfor
    return R'ᵢ
endif
```

**Fig. 2.** Pseudo-code for the special-case algorithm

Note that this algorithm implements the ideas described previously. For the three fixpoint example that we used earlier, the array for $R_3$ would have $n + 1$ entries because $R_3$ is within one enclosing greatest fixpoint. Initially these are all $\perp$, corresponding to the values $R_3^{0j0}$ (for $0 \leq j \leq n$). During the computation of the fixpoint for $R_2$, the entries are updated to hold the values $R_3^{0j\omega}$. When we

compute the approximation $R_1^1$ and begin computing the inner fixpoints again, the entries are used as the initial approximations $R_3^{1j0}$.

In proving this algorithm correct, we would show that any given array entry increases monotonically. While space limitations prevent us from giving the proof, we can use this fact to derive a complexity bound. Let $T_i$ denote the number of approximations computed for $R_i$, and let $T_i'$ denote the number of approximations for $R_i'$. Clearly, $T_1 \leq n+1$. From the algorithm, we see that the fixpoint for $R_i'$ is evaluated $T_i$ times, and for each evaluation, we produce $n+1$ approximations. Thus, $T_i' \leq (n+1)T_i$. For $R_i$, each entry in $A_i$ increases monotonically, so for any one entry, we can make at most $n$ steps in which the value strictly increases. There are $(n+1)^{i-1}$ entries in $A_i$, so this gives at most $(n+1)^i$ steps. We evaluate the fixpoint for $R_i$ at most $T_{i-1}'$ times. Thus, we make at most $T_{i-1}'$ extra steps to detect convergence. In total, we have $T_i \leq (n+1)^i + T_{i-1}'$. Expanding out the values, we get

$$
\begin{aligned}
T_1 &\leq n+1 \\
T_1' &\leq (n+1)T_1 = (n+1)^2 \\
T_2 &\leq (n+1)^2 + T_1' = 2(n+1)^2 \\
T_2' &\leq (n+1)T_2 = 2(n+1)^3 \\
&\vdots \\
T_q &\leq q(n+1)^q \\
T_q' &\leq q(n+1)^{q+1} .
\end{aligned}
$$

Summing over all fixpoints and expressing the result in terms of the alternation depth $d = 2q$, we get $O(d^2(n+1)^{d/2+1})$ steps. In contrast, previously known algorithms may require about $n^d$ steps to evaluate this formula. Generalizing to formulas with odd alternation depth yields the bound $O(d^2(n+1)^{\lfloor d/2 \rfloor+1})$.

In the general algorithm, we handle arbitrary formulas, save information for both types of fixpoints and always stop computations on detecting convergence. This version of the algorithm does not use tables to store the frontier values, since just initializing the tables requires about $n^{d/2}$ steps. If all of the fixpoint computations converged immediately, this would represent mostly wasted effort. Instead, frontiers will be represented by queues. We will write queues using square brackets, with the head of the queue at the left. The last element in the queue corresponds to a fixpoint and is conceptually replicated as many times as required. As an example, consider (1) again. During the computations with $R_1^0$, we will be building up the frontier values $R_3^{00\omega}$, $R_3^{01\omega}$, etc. Within $n$ steps, we will find the fixpoint value $R_2^{0\omega}$. Say this happens after three steps. Then the frontier for $R_3$ will be represented by the queue

$$[R_3^{00\omega}, R_3^{01\omega}, R_3^{02\omega}, R_3^{03\omega}] .$$

If we were to continue iterating on $R_2$, the value of $R_2$ would not change, and so $R_3^{0j\omega}$ would be equal to $R_3^{03\omega}$ for all $j > 3$. Rather than actually computing these values, we just view $R_3^{03\omega}$ as being duplicated as many times as needed.

Now when we start computing with $R_1^1$, we will pull elements from the above queue to start each computation of $R_3$. If $R_2$ now takes more than three steps to converge, we will use $R_3^{03\omega}$ more than once. Each time we pull it from the queue and find that the queue is now empty, we put another copy back in the queue in case another iteration on $R_2$ is required.

More deeply nested fixpoints give rise to nested queues. Consider a formula with five fixpoints: $\mu R_1 . \nu R_2 . \mu R_3 . \nu R_4 . \mu R_5 . (\cdots)$. For $R_5$, a frontier will be represented by a queue, each element of which is a queue of state sets. The elements of the outer queue are subfrontiers corresponding to the values $R_2^{i0}$, $R_2^{i1}$, $R_2^{i2}$, etc. The subfrontier corresponding to $R_2^{ij}$ holds fixpoints corresponding to $R_4^{ijk0}$, $R_4^{ijk1}$, $R_4^{ijk2}$, etc. In general, a frontier for a least fixpoint nested inside $q$ greatest fixpoints is represented by queues nested to depth $q$.

As the computation proceeds, existing frontiers will be decomposed to get at the starting fixpoint approximations, and we will be constructing new frontiers from the fixpoint values. In order to keep track of this decomposition and construction process, we use two stacks for each fixpoint variable $R$. One stack, $I_R$, will be associated with the frontier being decomposed, and the other stack, $F_R$, will hold the frontier being constructed. Each stack element is either a set of states (representing an approximation or a fixpoint value), or a queue (representing a subfrontier). We will write stacks using angle brackets, with the top of a stack on the left. Initially, the $I_R$ stack for a top-level fixpoint variable $R$ holds either $\top$ or $\perp$, depending on whether $R$ is a greatest or least fixpoint. The stack $I_R$ for a least fixpoint variable $R$ nested inside $q$ greatest fixpoints holds $\perp$ nested inside $q$ queues. The initial value for a stack corresponding to a greatest fixpoint variable nested inside a number of least fixpoints is similarly defined.

Pseudo-code for the algorithm is shown in Fig. 3. During each iteration for the fixpoint $\mu R . \psi(\cdots)$, we pull out the next sub-frontier for the inner $\nu$ variables (lines 8–12). These subfrontiers are pushed onto the initial stacks corresponding to the $\nu$ variables. For a top-level $\nu$-subformula $\nu R' . (\cdots)$ of $\psi$, these subfrontiers will be state sets representing the initial approximation to use for $R'$. We then recursively evaluate the inner fixpoints. Afterwards, we build up sub-frontiers for subsequent evaluations of the inner greatest fixpoints (lines 15–19). If the computation of $R$ has not yet converged, we discard the old frontiers for the inner $\mu$ fixpoints and replace them with the new frontiers that have been built up (lines 21–24). Note that with two successive $\mu$ 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 [11].

As an example of the algorithm's operation, we consider (1) again. Initially, $I_{R_1} = \langle \perp \rangle$, $I_{R_2} = \langle [\top] \rangle$, and $I_{R_3} = \langle [\perp] \rangle$. All the stacks $F_{R_1}$, $F_{R_2}$, and $F_{R_3}$ are empty. The computation proceeds as shown in Fig. 4. In the figure, $\mu R_1 \colon \perp$ denotes a call to the evaluation routine for the formula $\mu R_1 . (\cdots)$ with $\perp$ on the top of the stack $I_{R_1}$ (i.e., the evaluation of the fixpoint starting from $\perp$). 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, $\mu R_3 \colon \perp \rightarrow R_3^{00\omega}$ denotes the evaluation

```
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
```

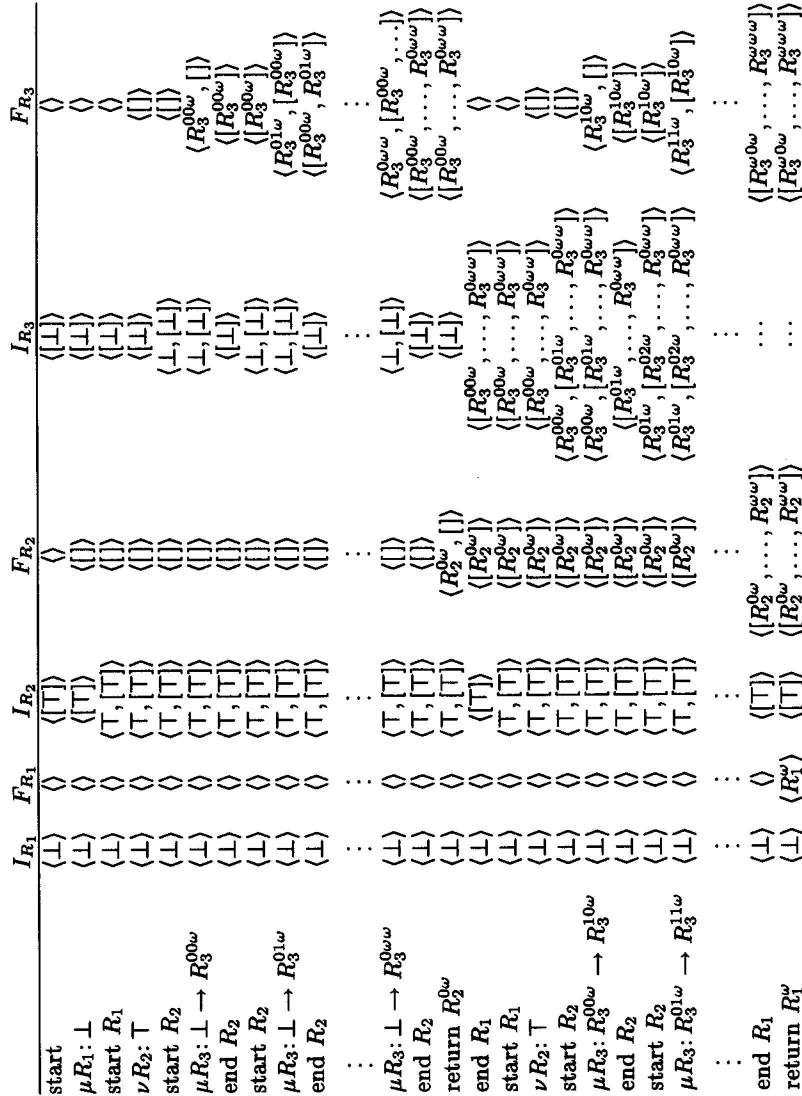
**Fig. 3.** Pseudo-code for the general algorithm

of $\mu R_3.(\cdots)$ starting with $\perp$ and yielding $R_3^{00\omega}$ as the result. The figure shows how the stacks evolve during the computation.

Unfortunately, the general algorithm still has a worst case complexity of about $n^{d/2}$. We have constructed transition systems and classes of formulas where the algorithm takes at least $(n+1)^{d/2+1}$ steps to evaluate a formula in the class with alternation depth $d$ (even). We do not have space to present the construction, but we will try to give an intuitive idea of the kind of behavior that leads to the high complexity. The formulas involve $d/2$ pairs $(R_i, R'_i)$ of fixpoint variables. $R_i$ is a least fixpoint variable, and $R'_i$ is a greatest fixpoint variable. Let the states of the system be $\{s_0, \ldots, s_{n-1}\}$. Suppose that each of the pairs $(R_1, R'_1)$, ..., $(R_i, R'_i)$ has one of the following values:

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

Call such a situation a "diagonal configuration." Obviously, the number of diagonal configurations for these variables is $(n+1)^{i/2}$. Any two diagonal con-



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

**Fig. 4.** Example computation of the algorithm in Fig. 3

figurations are incomparable with respect to pairwise set containment. Hence, computing the inner fixpoints for one diagonal configuration gives us no information about the inner fixpoints for a different diagonal configuration. We can also arrange for all inner fixpoints to be $\top$ above the diagonal (i.e., when $R_i \cup R'_i = \top$ and they have a nonempty intersection) and to be $\bot$ below the diagonal. Under these circumstances, we can show that all diagonal configurations for $(R_1, R'_1)$, ..., $(R_{d/2}, R'_{d/2})$ will occur during the computation. This means that the number of steps must be at least $(n+1)^{d/2}$.

Our method uses more space than previous approaches, since frontier values must be stored. In the worst case, we may have to store about $n^{d/2}$ state sets. There does not seem to be any way to avoid this, since we cannot rearrange the order in which the fixpoint approximations are computed. Note though that the space complexity is generally much better than the time complexity (since we only store "slices" of the approximations that have been computed). If needed, we can trade time for space during long computations by simplifying or discarding some of the frontiers.

## 4 Conclusion

We have presented a new algorithm for evaluating a formula in the propositional $\mu$-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, 8, 9] 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.

Another line of research involves trying to place lower bounds on the complexity of the evaluation process. It can be shown that the language recognition version of the problem is in NP intersect 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. Given a formula like $\mu R_1 . \psi_1(R_1, \nu R_2 . \psi_2(R_1, R_2))$, we would view $\psi_1$ and $\psi_2$ 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. Another way of exploring the complexity of the problem is to look for links with classical complexity theory. Jha has obtained some results connecting fixpoint alternation and alternating Turing machines (ATMs). For example, a fixpoint formula with $k$ alternations can be used to simulate an ATM with $k$ alternations.

## 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 $\omega$-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. R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, 1990.
8. R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In Bochmann and Probst [2].
9. 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.
10. 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.
11. 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.
12. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.
13. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.
14. K. G. Larsen. Efficient local correctness checking. In Bochmann and Probst [2].
15. A. Mader. Tableau recycling. In Bochmann and Probst [2].
16. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
17. C. Stirling and D. J. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, October 1991.
18. A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
19. G. Winskel. Model checking in the modal $\nu$-calculus. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming*, 1989.


This article was processed using the LaTeX macro package with LLNCS style