Model Checking Algorithms for the μ-Calculus

Sergey Berezin Edmund Clarke
Somesh Jha Will Marrero

September 23, 1996
CMU-CS-96-180

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The propositional μ-calculus is a powerful language for expressing properties of transition systems by using least and greatest fixpoint operators. Recently, the μ-calculus has generated much interest among researchers in computer-aided verification. This interest stems from the fact that many temporal and program logics can be encoded into the μ-calculus. In addition, important relations between transition systems, such as weak and strong bisimulation equivalence, also have fixpoint characterizations. Wide-spread use of binary decision diagrams has made fixpoint based algorithms even more important, since methods that require the manipulation of individual states do not take advantage of this representation.

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.

Keywords: automatic verification, temporal logic, model checking, binary decision diagrams, bisimulation

1 Introduction

The propositional μ-calculus is a powerful language for expressing properties of transition systems by using least and greatest fixpoint operators. Recently, the μ-calculus has generated much interest among researchers in computer-aided verification. This interest stems from the fact that many temporal and program logics can be encoded into the μ-calculus. In addition, important relations between transition systems, such as weak and strong bisimulation equivalence, also have fixpoint characterizations [17].

Another source of interest in the μ-calculus comes from the existence of efficient model checking algorithms for this formalism. As a consequence, verification procedures for many temporal and modal logics can be succinctly described by translating into the μ-calculus. Wide-spread use of binary decision diagrams has made fixpoint based algorithms even more important, since methods that require the manipulation of individual states do not take advantage of this representation.

Several versions of the propositional μ-calculus have been described in the literature, and the ideas in this paper will work with any of them. For the sake of concreteness, we will use the propositional μ-calculus of Kozen [12]. Closed formulas in this logic evaluate to sets of states. A considerable amount of research has focused on finding techniques for evaluating such formulas efficiently, and many algorithms have been proposed for this purpose. These algorithms generally 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 [19], and Winskel [21]. More recently, Andersen [1] and Larsen [13] have developed efficient local methods for a subset of the μ-calculus. Mader [15] has also proposed improvements to the tableau-based method of Stirling and Walker that seem to increase its efficiency.

In this paper, we restrict ourselves to global model checking procedures. Global procedures generally work bottom-up through the formula, evaluating each subformula based on the values of its subformulas. Iteration is used to compute the fixpoints. Because of fixpoint nesting, a naive global algorithm may require about $O(n^k)$ iterations to evaluate a formula, where n is the number of states in the transition system and k is the depth of nesting of the fixpoints. Emerson and Lei [11] improve on this by observing that successively nested fixpoints of the same type do not increase the complexity of the computation. They formalize this observation using the notion of *alternation depth* and give an algorithm requiring only about $O(n^d)$ iterations, 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 iterations has remained about $O(n^d)$. In [14] the authors have improved on this by giving an algorithm that uses only $O(n^{d/2})$ iterations to compute a formula with alternation depth d, thus requiring only about the square root of the time needed by earlier algorithms.

This paper describes the propositional μ-calculus and general algorithms for evaluating μ-calculus formulas. Examples of verification problems that can be encoded within the language of the μ-calculus are also provided. The remainder of this paper is organized as follows. A formal syntax and semantics for the propositional μ-calculus is given in Section 2. Section 3 discusses different algorithms for evaluation of μ-calculus formulas and their complexities. A brief description of Ordered Binary Decision Diagrams (OBDDs) is given in Section 4. Section 5 presents the algorithm for encoding μ-calculus formulas with OBDDs. The syntax and semantics for CTL and for CTL with fairness constraints is given in Section 6, while a translation of these logics into the μ-calculus is given in Section 7. Definitions for different kinds of simulation preorders and bisimulation equivalences are given in Section 8 along with encodings for these relations in the μ-calculus. Finally, Section 9 concludes the paper and discusses some open problems.

2 The Propositional μ-Calculus

In the propositional μ-calculus, formulas are constructed as follows:

- atomic propositions $AP = \{p, p_1, p_2, \ldots\}$
- relational variables $VAR = \{R,\ R_1,\ R_2,\ \ldots\}$
- logical connectives $\neg\cdot$, $\cdot \wedge \cdot$ and $\cdot \vee \cdot$
- modal operators $\langle a \rangle \cdot$ and $[a]\cdot$, where a is an action in the set $Act = \{a, b, a_1, a_2, \ldots\}$
- fixpoint operators $\mu R_i.(\cdots)$ and $\nu R_i.(\cdots)$. Relational variables bound by the fixpoint operators must be in the scope of the even number of negations.

There is a stardand notion of free and bound variables (by fixpoint operators) in the formulas. Closed formulas are the formulas without free variables. Formulas in this calculus are interpreted relative to a transition system $M = (\top, T, L)$ that consists of:

- a nonempty set of states $\top$
- a mapping $L : AP \to 2^{\top}$ that takes each atomic proposition to some subset of $\top$ (the states where the proposition is true)
- a mapping $T : Act \to 2^{\top \times \top}$ that takes each action to a binary relation over $\top$ (the state changes that can result from making an action)

The intuitive meaning of the formula $\langle a \rangle \phi$ is "it is possible to make an a-action 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 making an a-action." 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 $\perp$. Also, in the rest of this paper, we will use the more intuitive notation $s \xrightarrow{a} s'$ to mean $(s, s') \in T(a)$.

Formally, a formula ϕ is interpreted as a set of states in which ϕ is true. We write such set of states as $[\![\phi]\!]_M\, e$, where M is a transition system and $e : VAR \to 2^{\top}$ is an environment. We denote by $e\,[R \leftarrow S]$ a new environment which is the same as e except that $e\,[R \leftarrow S]\,(R) = S$. The set $[\![\phi]\!]_M\, e$ is defined recursively as follows.

- $[\![p]\!]_M\, e = L(p)$
- $[\![R]\!]_M\, e = e(R)$
- $[\![\neg\phi]\!]_M\, e = \top - [\![\phi]\!]_M\, e$
- $[\![\phi \wedge \psi]\!]_M\, e = [\![\phi]\!]_M\, e \cap [\![\psi]\!]_M\, e$
- $[\![\phi \vee \psi]\!]_M\, e = [\![\phi]\!]_M\, e \cup [\![\psi]\!]_M\, e$
- $[\![\langle a\rangle\phi]\!]_M\, e = \{\, s \mid \exists t\, [s \xrightarrow{a} t \text{ and } t \in [\![\phi]\!]_M\, e]\,\}$
  $[\![[a]\,\phi]\!]_M\, e = \{\, s \mid \forall t\, [s \xrightarrow{a} t \text{ implies } t \in [\![\phi]\!]_M\, e]\,\}$
- $[\![\mu R.\phi]\!]_M\, e$ is the least fixpoint of the predicate transformer $\tau\colon 2^{\top} \to 2^{\top}$ defined by:
  $$\tau(S) = [\![\phi]\!]_M\, e\,[R \leftarrow S]$$
- The interpretation of $\nu R.\phi$ is similar, except that we take the greatest fixpoint.

Within formulas, the negation is restricted in use, and so the fixpoints are guaranteed to be well-defined. Formally, every logical connective except negation is monotonic ($\phi \to \phi'$ implies $\phi\wedge\psi \to \phi'\wedge\psi$, $\phi\vee\psi \to \phi'\vee\psi$, $\langle a\rangle\phi \to \langle a\rangle\phi'$, and $[a]\phi \to [a]\phi'$), and all the negations can be pushed down to the atomic propositions using De Morgan's laws and dualities ($\neg[a]\phi \equiv \langle a\rangle\neg\phi$, $\neg\langle a\rangle\phi \equiv [a]\neg\phi$, $\neg\mu R.\phi(R) \equiv \nu R.\neg\phi(\neg R)$, $\neg\nu R.\phi(R) \equiv \mu R.\neg\phi(\neg R)$). Since bound variables are under even number of negations, they will be negation free after this process. Thus, each possible formula in a fixpoint operator is monotonic and hence each possible τ is also monotonic ($S \subseteq S'$ implies $\tau(S) \subseteq \tau(S')$). This is enough to ensure the existence of the fixpoints [20]. Furthermore, since we will be evaluating formulas 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:

$$[\![\mu R.\phi]\!]_M\, e = \bigcup_i \tau^i(\bot) \qquad [\![\nu R.\phi]\!]_M\, e = \bigcap_i \tau^i(\top).$$

($\tau^i(S)$ can be defined recursively as $\tau^0(S) = S$ and $\tau^{i+1}(S) = \tau(\tau^i(S))$) Since the domain $\top$ 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.

The *alternation depth* of a formula is intuitively equal to the number of alternations in the nesting of least and greatest fixpoints, when all negations are applied only to propositions. There are other more elaborate definitions of alternation depth [1, 2, 9], that take into account the possibility that nested fixpoints may still be independent. Such fixpoints do not depend on the value of approximations to outer fixpoints. Consequently, they only need to be evaluated once. This type of nesting does not increase the effective alternation depth. However, to simplify our presentation we will use the definition of alternation depth given by Emerson and Lei [11]. Formally, the alternation depth is defined as follows:

Definition 2.1

- The alternation depth of an atomic proposition or a relational variable is 0;
- 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 and ψ.
- 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* of $\nu R.\phi$ is similarly defined.

Example 2.1 Consider the following formula which will be discussed in Section 7.

$$\nu Y.(P \wedge \langle a \rangle [\mu X.(P \wedge \langle a \rangle X) \vee (h \wedge Y)])$$

This formula expresses the property "P holds continuously along some fair a-path" and has an alternation depth of two.

Because of the duality,

$$\nu R.\phi(\cdots, R, \cdots) \quad = \quad \neg \mu R.\neg \phi(\cdots, \neg R, \cdots)$$

we could have defined the propositional μ-calculus with just the least fixpoint operator and negation. In order to give a succinct description of certain constructions we sometimes use the dual formulation. However, the concept of alternation depth is easier to define using the formulation given earlier.

3 Evaluating Fixpoint Formulas

We define *model checking* as a technique of verifying a model relative to its specification in the μ-calculus. This is the same as evaluating a formula in a model, i.e., finding the set of states of the model where the formula is true. Figure 1 presents the naive, straightforward, recursive algorithm for evaluating μ-calculus formulas. The time complexity of the algorithm in Figure 1 is exponential in the length of the formula. To see this, we analyze the behavior of the algorithm when computing nested fixpoints. The algorithm computes fixpoints by iteratively computing approximations. These successive approximations form a chain ordered by inclusion. Since the number of strict inclusions in such a chain is limited by the number of possible states, we have that the loop will execute at most $n+1$ times, where $n = |\top|$. Each iteration of the loop involves a recursive call to evaluate the body of the fixpoint with a different value for the fixpoint variable. If in turn, the subformula being evaluated contains a fixpoint, the evaluation of its body will also involve a loop containing up to $n+1$ recursive calls. Thus, the total number of recursive calls will be $O(n^2)$. In general, the body of the innermost fixpoint will be evaluated $O(n^k)$ times where k is the maximum nesting depth of fixpoint operators in the formula.

Note that we have only considered the number of iterations required when evaluating fixpoints and not the number of steps required to evaluate a μ-calculus formula. While each

```
function eval($\phi$, $e$)
if $\phi = p$ then return $L(p)$
if $\phi = R$ then return $e(R)$
if $\phi = \psi_1 \wedge \psi_2$ then
    return eval($\psi_1, e$)$\cap$ eval($\psi_2, e$)
if $\phi = \psi_1 \vee \psi_2$ then
    return eval($\psi_1, e$)$\cup$ eval($\psi_2, e$)
if $\phi = \langle a \rangle \psi$ then
    return $\{ s \mid \exists t \, [s \xrightarrow{a} t$ and $t \in$ eval($\psi, e$)$] \}$
if $\phi = [a]\psi$ then
    return $\{ s \mid \forall t \, [s \xrightarrow{a} t$ implies $t \in$ eval($\psi, e$)$] \}$

if $\phi = \mu R.\psi(R)$ then
    $R_{\mathrm{val}} := \bot$
    repeat
        $R_{\mathrm{old}} := R_{\mathrm{val}}$
        $R_{\mathrm{val}} :=$ eval($\psi, e\,[R \leftarrow R_{\mathrm{val}}]$)
    until $R_{\mathrm{val}} = R_{\mathrm{old}}$
    return $R_{\mathrm{val}}$

if $\phi = \nu R.\psi(R)$ then
    $R_{\mathrm{val}} := \top$
    repeat
        $R_{\mathrm{old}} := R_{\mathrm{val}}$
        $R_{\mathrm{val}} :=$ eval($\psi, e\,[R \leftarrow R_{\mathrm{val}}]$)
    until $R_{\mathrm{val}} = R_{\mathrm{old}}$
    return $R_{\mathrm{val}}$
```

Figure 1: Pseudocode for the naive algorithm

fixpoint may only take $O(n)$ iterations, each individual iteration can take up to $O(|M||\phi|)$ steps, where $M = (\top, T, L)$ is the model and $|M| = |\top| + \sum_{a \in Act} |T(a)|$. In general, then, this algorithm has time complexity $O[|M||\phi|n^k]$.

A result by Emerson and Lei demonstrates that the value of a fixpoint formula can be computed with $O((|\phi|n)^d)$ iterations, where d is the alternation depth of ϕ. Their algorithm is similar to the straightforward one described above, except when a fixpoint is nested directly within the scope of another fixpoint of the same type. In this case, the fixpoints are computed slightly differently.

A simple example will suffice to demonstrate the idea. When discussing the evaluation of fixpoint formulas, we will use $R_1, \ldots, R_k$ as the fixpoint variables, with R_1 being the outermost fixpoint variable and R_k being the innermost. We will use the notation $R_j^{i_1 \cdots i_j}$ to denote the value of the i_j-th approximation for R_j after having computed the i_l-th approximation for R_l for $1 \leq l < j$. We use $i_j = \omega$ to indicate that we are considering the final approximation (the actual fixpoint value) for R_j. For example, R_1^ω is the value of the fixpoint for R_1 and R_2^{30} is the initial approximation for R_2 after having computed the third approximation for R_1. Consider the formula

$$\mu R_1.\psi_1(R_1, \mu R_2.\psi_2(R_1, R_2)).$$

The subformula $\mu R_2.\psi_2(R_1, R_2)$ 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 initial approximation $R_1^0 = \bot$ and then compute $\tau(R_1^0)$. This is done by iteratively computing approximations for the inner fixpoint also starting from $R_2^{00} = \bot$ until we reach a fixpoint $R_2^{0\omega}$. Now R_1 is increased to R_1^1, the result of evaluating $\psi_1(R_1^0, R_2^{0\omega})$. We next compute the least fixpoint $\tau(R_1^1)$. Since $R_1^0 \subseteq R_1^1$, by monotonicity we know that $\tau(R_1^0) \subseteq \tau(R_1^1)$. Now note that the following lemma holds:

Lemma 3.1 If $S \subseteq \bigcup_i \tau^i(\bot)$ then $\bigcup_i \tau^i(S) = \bigcup_i \tau^i(\bot)$.

In other words, to compute a least fixpoint, it is enough to start iterating with any approximation known to be below the fixpoint. Thus , we can start iterating with $R_2^{10} = R_2^{0\omega} = \tau(R_1^0)$ instead of $R_2^{10} = \bot$. When we compute the fixpoint $R_2^{1\omega}$, we next compute the new approximation to R_1, which is R_1^2, the result of evaluating $\psi_1(R_1^1, R_2^{1\omega})$. Again, we know that $R_1^1 \subseteq R_1^2$ which implies that $\tau(R_1^1) \subseteq \tau(R_1^2)$. But $\tau(R_1^1) = R_2^{1\omega}$, the value of the last inner fixpoint computed, and $\tau(R_1^2) = R_2^{2\omega}$ the fixpoint to be computed next. Again, we can start iterating with any approximation below the fixpoint. So to compute $R_2^{2\omega}$ we begin with $R_2^{20} = R_2^{1\omega} = \tau(R_1^1)$. In general, when computing $R_2^{i\omega}$ we always begin with $R_2^{i0} = R_2^{(i-1)\omega}$. Since we never restart the inner fixpoint computation, we can have at most n increases in the value of the inner fixpoint variable. Overall, we only need $O(n)$ iterations to evaluate this expression, instead of $O(n^2)$. In general, this type of simplification leads to an algorithm that computes fixpoint formulas in time exponential in the alternation depth of the formula since we only reset an inner fixpoint computation when there is an alternation in fixpoints in the formula.

Thus, this algorithm for evaluating μ-calculus formulas is identical to the naive algorithm except in the case when the main connective is a fixpoint operator. The pseudocode for this algorithm is given in Figure 2. Note that unlike the naive algorithm, the approximation values $A[i]$ are not reset when evaluating the subformula $\mu R_i.\psi(R_i)$ ($\nu R_i.\psi(R_i)$). Instead, we reset all top-level greatest (least) fixpoint variables contained in ψ. Recall that by the top-level fixpoints in a formula we mean all the fixpoints of the same type (μ or ν) that are not in the scope of the other type of fixpoints. This guarantees that when we evaluate a top-level fixpoint subformula of the same type, we do not start the computation from $\bot$ or $\top$, but from the previously computed value as in our example.

In [14] the authors observe that by storing even more intermediate values, the time complexity for evaluating fixpoint formulas can be reduced to $O(n^{\lfloor d/2 \rfloor + 1})$ where again d is the alternation depth of the formula. To simplify our discussion, we consider formulas with strict alternation of fixpoints. We present a small example to illustrate the idea behind this algorithm.

Consider the formula:

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

function eval(ϕ, e)

$N :=$ The number of fixpoint operators in ϕ
for $i := 1$ to N do $A[i] :=$ if the i-th fixpoint of ϕ is μ then $\bot$ else $\top$
return evalrec(ϕ, e)

Where *evalrec* is defined recursively as

function evalrec(ϕ, e)

if $\phi = p$ then return $L(p)$
if $\phi = R$ then return $e(R)$
if $\phi = \psi_1 \wedge \psi_2$ then
    return evalrec(ψ_1, e)$\cap$ evalrec(ψ_2, e)
if $\phi = \psi_1 \vee \psi_2$ then
    return evalrec(ψ_1, e)$\cup$ evalrec(ψ_2, e)
if $\phi = \langle a \rangle \psi$ then
    return $\{ s \mid \exists t \, [s \xrightarrow{a} t$ and $t \in$ evalrec$(\psi, e)] \}$
if $\phi = [a]\psi$ then
    return $\{ s \mid \forall t \, [s \xrightarrow{a} t$ implies $t \in$ evalrec$(\psi, e)] \}$

if $\phi = \mu R_i . \psi(R_i)$ then
    For all top-level greatest fixpoint subformulas $\nu R_j . \psi'(R_j)$ of ψ
        do $A[j] := \top$
    repeat
        $R_{\text{old}} := A[i]$
        $A[i] :=$ evalrec$(\psi, e \, [R_i \leftarrow A[i]])$
    until $A[i] = R_{\text{old}}$
    return $A[i]$

if $\phi = \nu R_i . \psi(R_i)$ then
    For all top-level least fixpoint subformulas $\mu R_j . \psi'(R_j)$ of ψ
        do $A[j] := \bot$
    repeat
        $R_{\text{old}} := A[i]$
        $A[i] :=$ evalrec$(\psi, e \, [R_i \leftarrow A[i]])$
    until $A[i] = R_{\text{old}}$
    return $A[i]$

Figure 2: Pseudocode for the Emerson and Lei algorithm

To compute the outer fixpoint, we start with $R_1 = \bot$, $R_2 = \top$ and $R_3 = \bot$. As in the previous case, we denote these values by 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}, $R_3^{01\omega}$, ..., $R_2^{0\omega}$, $R_3^{0\omega\omega}$. 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 $\perp$, we obtain faster convergence. In addition, we have

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

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 $R_3^{11\omega}$: we start the computation from $R_3^{110} = R_3^{01\omega}$. And again, since $R_1^0 \subseteq R_1^1$, $R_2^{01} \subseteq R_2^{11}$, and $R_3^{01\omega} \subseteq R_3^{11\omega}$, we will have $R_2^{02} \subseteq R_2^{12}$. In general, we will have $R_2^{0j} \subseteq R_2^{1j}$ and $R_3^{0j\omega} \subseteq R_3^{1j\omega}$ so 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 $R_3^{2m0} = R_3^{1m\omega}$ ($R_3^{(k+1)m0} = R_3^{km\omega}$).

The table in Figure 3 illustrates this by showing the relationship between all the different possible approximation values for R_3. Each row can have at most $n+1$ entries, one for each approximation to $\nu R_2.\psi_2$. At first glance, it seems possible that each column could have as many as n^2 entries. However, each chain represented by each column can have at most $n+1$ distinct values. Repeated values only appear when convergence is reached ($R_3^{ij\omega} = R_3^{ij(\omega-1)}$) and when we start a computation from a previously computed fixpoint ($R_3^{(i+1)j0} = R_3^{ij\omega}$). Convergence is reached every time the fixpoint is evaluated, and this fixpoint is evaluated once for every outer greatest fixpoint approximation of which there can be no more than $n+1$. Since there can be no more than $n+1$ evaluations, we can start from a previously computed fixpoint no more than n times. So the number of repeated values is bounded by

$2n+1$. Thus, the total number of entries in any column is bound by $3n+2$ and the total number of assignments to R_3 during the entire computation is bound by $(3n+2)(n+1)$. This means that there are at most $O(n^2)$ iterations performed to compute the innermost fixpoint.

$$
\begin{array}{ccccc}
R_3^{\omega 0 \omega} & \supseteq & R_3^{\omega 1 \omega} & \supseteq \cdots \supseteq & R_3^{\omega\omega\omega} \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
\vdots & & \vdots & & \vdots \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
R_3^{\omega 01} & \supseteq & R_3^{\omega 11} & \supseteq \cdots \supseteq & R_3^{\omega\omega 1} \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
R_3^{\omega 00} & \supseteq & R_3^{\omega 10} & \supseteq \cdots \supseteq & R_3^{\omega\omega 0} \\
\| & & \| & & \| \\
\vdots & & \vdots & & \vdots \\
\| & & \| & & \| \\
R_3^{10\omega} & \supseteq & R_3^{11\omega} & \supseteq \cdots \supseteq & R_3^{1\omega\omega} \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
\vdots & & \vdots & & \vdots \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
R_3^{101} & \supseteq & R_3^{111} & \supseteq \cdots \supseteq & R_3^{1\omega 1} \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
R_3^{100} & \supseteq & R_3^{110} & \supseteq \cdots \supseteq & R_3^{1\omega 0} \\
\| & & \| & & \| \\
R_3^{00\omega} & \supseteq & R_3^{01\omega} & \supseteq \cdots \supseteq & R_3^{0\omega\omega} \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
\vdots & & \vdots & & \vdots \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
R_3^{001} & \supseteq & R_3^{011} & \supseteq \cdots \supseteq & R_3^{0\omega 1} \\
\rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} & & \rotatebox{-90}{$\supseteq$} \\
R_3^{000} & \supseteq & R_3^{010} & \supseteq \cdots \supseteq & R_3^{0\omega 0}
\end{array}
$$

Figure 3: Monotonicity constraints on approximations to R_3

Again, this algorithm for evaluating a μ-calculus formula is identical to the naive algorithm except when the main connective is a fixpoint operator. To facilitate explanation, we consider only formulas with strict alternation of fixpoints, and in particular, with the form:

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

The pseudocode for this part of the algorithm is given in Figure 4. For computing the outermost fixpoint (corresponding to R_1) we follow the naive algorithm, i.e., start with $\perp$

if $\phi = \mu R_i.\psi_i(R_i)$ and $i \geq 2$ then
    $R_{\text{val}} := \mathcal{T}_i[k_1] \cdots [k_{i-1}]$
    repeat
        $R_{\text{old}} := R_{\text{val}}$
        $R_{\text{val}} := \text{evalrec}(\psi_i, e\,[R_i \leftarrow R_{\text{old}}])$
    until $R_{\text{val}} = R_{\text{old}}$
    $\mathcal{T}_i[k_1] \cdots [k_{i-1}] := R_{\text{val}}$
    return R_{val}

if $\phi = \nu R'_i.\psi'_i(R'_i)$ then
    $k_i := 0$
    $R_{\text{val}} := \top$
    repeat
        $R_{\text{val}} := \text{evalrec}(\psi'_i, e\,[R'_i \leftarrow R_{\text{val}}])$
        $k_i := k_i + 1$
    until $k_i = |\top|$
    return R_{val}

Figure 4: Pseudocode for the efficient algorithm

and iterate until convergence. The algorithm uses a table $\mathcal{T}_i$ to store the last computed fixpoint values for the μ-variables R_i (for $i \geq 2$). Initially, all entries in $\mathcal{T}_i$ are $\bot$. The table $\mathcal{T}_i$ is a multi-dimensional table. For the i-th least fixpoint (corresponding to R_i) we index the table $\mathcal{T}_i$ by the iteration counters $k_1, \cdots, k_{i-1}$ of the greatest fixpoints in which the i-th least fixpoint is nested. When evaluating R_i, we start with the corresponding table value and iterate until convergence. At the end of the iteration, the table holds the fixpoint value. When evaluating R'_i, we always begin with $\top$ and iterate until convergence. Note that this algorithm implements the ideas in the previous example.

If we use these ideas, how many steps does the computation take? To try to answer this question, we look at the number of approximations computed for the R_is and R'_is in the algorithm. Let T_i denote the number of approximations for R_i, and let T'_i denote the number of approximations for R'_i. The fixpoint for R'_i is evaluated at most T_i times (the number of approximations to the enclosing R_i). Each evaluation can take at most $n+1$ iterations for a total of $(n+1)T_i$ approximations. Thus, $T'_i \leq (n+1)T_i$. The fixpoint for R_i has a table $\mathcal{T}_i$ with $(n+1)^{i-1}$ entries. Because of the monotonicity constraints, each entry can go through at most $n+1$ distinct values. Since there are $(n+1)^{i-1}$ entries, we have a total of $(n+1)^i$ iterations. These iterations correspond to the case when the loop test is false. In addition, each time we evaluate the fixpoint for R_i we will take one extra step to detect convergence which will not result in a new value for the corresponding table entry. We evaluate the fixpoint for R_i at most T'_{i-1} times. Thus we make at most T'_{i-1} iterations when the loop test is true. In total, we have $T_i \leq (n+1)^i + T'_{i-1}$. Solving this recurrence, we get:

$$\begin{aligned} T_i &\leq i(n+1)^i \\ T_i' &\leq i(n+1)^{i+1} \end{aligned}$$

Summing over all fixpoints and expressing the result in terms of the alternation depth $d = 2q$, we get that the algorithm takes $O\left(d(n+1)^{d/2+1}\right)$ iterations when computing the fixpoints in a formula with strict alternation. In comparison, previously known algorithms may require $O(n^d)$ iterations.

4 Ordered Binary Decision Diagrams (OBDDs)

In this section we give a brief description of an efficient data structure for representing boolean functions. Consider the space $\mathcal{BF}_n$ of boolean functions on n variables $x_0, x_1, \cdots, x_{n-1}$. We assume that there is a total ordering on the boolean variables. The ordering is given by the index, i.e., x_i is ordered before x_j iff $i < j$. The symbol OBDD(f) will denote the Ordered Binary Decision Diagram (OBDD) for the boolean function f [4]. OBDDs have the following *canonicity* property:

Theorem 4.1 (Canonicity Theorem): Given two boolean functions f and g in the space $\mathcal{BF}_n$, OBDD(f) = OBDD(g) iff $f = g$.

A detailed proof is given in [4].

We will give a succinct explanation of how OBDDs work through an example. For a more thorough treatment see [4, 6]. Consider the following boolean function f:

$$f = x_0 \oplus x_1 \oplus x_2$$

Figure 5 gives the binary tree T corresponding to the boolean function f. Notice that the binary subtree which we get by following the paths $(0,1)$ and $(1,0)$ from the root are the same. The same is true if we follow the paths $(0,0)$ and $(1,1)$. Figure 6 reflects this sharing. Notice that the number of nodes is reduced from 15 to 7. In general, the binary tree corresponding to the parity of n bits has $2^{n+1} - 1$ nodes. The OBDD for the same function has $2n + 1$ nodes. Therefore, in some cases OBDD can be exponentially more succinct than the straightforward representation. We will use $|\text{OBDD}(f)|$ to denote the size of the OBDD for f, i.e., the number of nodes in OBDD(f). In addition to being a canonical representation, OBDDs support the usual operations on boolean functions efficiently. The complexity of some of the operations is shown below:

- Given the OBDDs for f and g, the OBDD for $f \vee g$ and $f \wedge g$ can be computed in time $O(|\text{OBDD}(f)| \cdot |\text{OBDD}(g)|)$.
- Given the OBDD for f, the OBDD for $\neg f$ can be computed in time $O(|\text{OBDD}(f)|)$.
- Given the OBDD for f, the OBDDs for $\exists x_i f$ and $\forall x_i f$ can be computed in time $O(|\text{OBDD}(f)|^2)$.

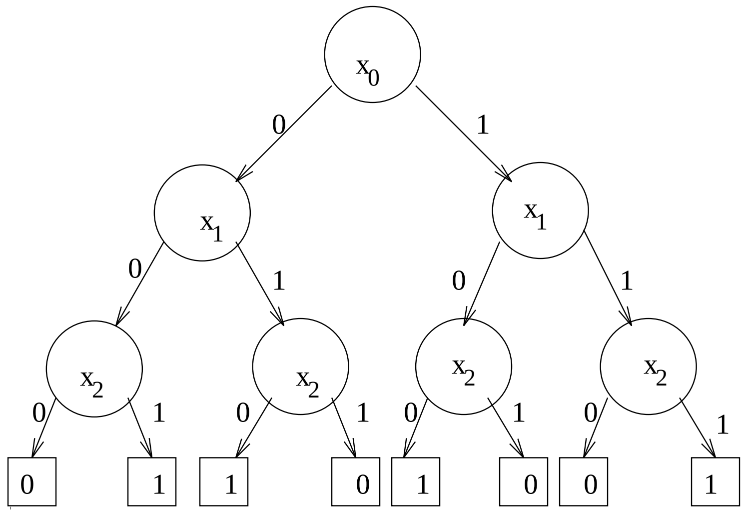

Figure 5: Tree for the 3 bit parity function

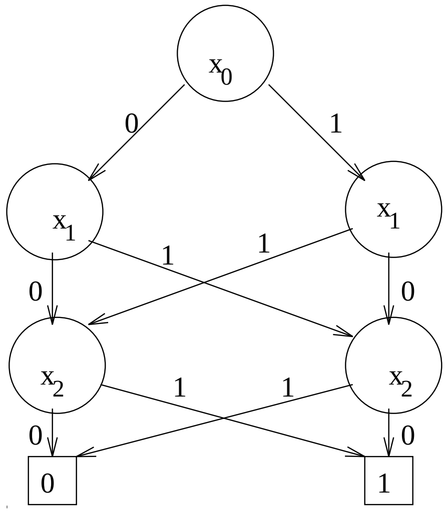

Figure 6: OBDD for the 3 bit parity function

Variable ordering is extremely important in OBDDs. For example, consider the following boolean function :

$$f(x_1, \cdots, x_n, x'_1, \cdots, x'_n) \quad = \quad \bigwedge_{i=1}^{n} (x_i = x'_i)$$

The OBDD for f with the variable ordering

$$x_1 < x'_1 < x_2 < x'_2 < \cdots x_n < x'_n$$

has size $3n + 2$. As the following lemma shows, the OBDD for f can have size exponential in n under some variable orderings. Moreover, there are some functions whose OBDDs have exponential size under any variable ordering [4].

Lemma 4.1 Let $f(x_1, \cdots, x_n, x'_1, \cdots, x'_n)$ be the following boolean function:

$$\bigwedge_{i=1}^{n} (x_i = x'_i)$$

Let F be the OBDD for f such that all the unprimed variables are ordered before all the primed variables. In this case $|F| \geq 2^n$.

Proof: Consider two distinct assignments $(b_1, \cdots, b_n)$ and $(c_1, \cdots, c_n)$ to the boolean vector $(x_1, \cdots, x_n)$. These two assignments can be distinguished because of the following equation:

$$f(b_1, \cdots, b_n, b_1, \cdots, b_n) \neq f(c_1, \cdots, c_n, b_1, \cdots, b_n)$$

Let v_1 and v_2 be the nodes reached after following the path $(b_1, \cdots, b_n)$ and $(c_1, \cdots, c_n)$ from the top node. Since these two assignments can be distinguished, $v_1 \neq v_2$. There are 2^n different assignments to the boolean vector $(x_1, \cdots, x_n)$ and each of them corresponds to a different node (at level n) in the OBDD F. Therefore, the number of nodes at level n in the OBDD F is greater than or equal to 2^n. □

5 Translating the μ-Calculus into OBDDs

In this section we describe how to use OBDDs in the model checking algorithms described earlier. First, we show how to encode a transition system $M = (\top, T, L)$ into OBDDs. The domain $\top$ is encoded by the set of values of the n boolean variables $x_1, \cdots, x_n$, i.e., $\top$ is now the space of 0-1 vectors of length n. Each variable x_i has a corresponding primed variable x'_i. Instead of writing $x_1, \cdots, x_n$, we sometimes use the vector notation $\vec{x}$. For example, we write $\text{OBDD}_p(x_1, \cdots, x_n)$ as $\text{OBDD}_p(\vec{x})$. Given an interpretation we build the OBDDs corresponding to closed μ-calculus formulas in the following manner.

- Each atomic proposition p has an OBDD associated with it. We will denote this OBDD by $\text{OBDD}_p(\vec{x})$. $\text{OBDD}_p(\vec{x})$ has the property that $\vec{y} \in \{0,1\}^n$ satisfies OBDD_p iff $\vec{y} \in L(p)$.
- Each program letter a has an ordered binary decision diagram $\text{OBDD}_a(\vec{x}, \vec{x}')$ associated with it. A 0-1 vector $(\vec{y}, \vec{z}) \in \{0,1\}^{2n}$ satisfies OBDD_a iff

$$(\vec{y}, \vec{z}) \in T(a)$$

Now we describe the encoding of the semantic sets of formulas into OBDDs. Assume that we are given a μ-calculus formula ϕ with free relational variables $R_1, \cdots, R_k$. $\mathcal{A}[R_i]$ gives the OBDD corresponding to the relational variable R_i. $\mathcal{A}\langle R \leftarrow B_R\rangle$ creates a new association by adding a relational variable R and associating an OBDD B_R with R. In other words, $\mathcal{A}$ can be considered as an environment with OBDD representation. The procedure B given below takes a μ-calculus formula ϕ and an association list $\mathcal{A}$ ($\mathcal{A}$ assigns an OBDD to each free relational variable occuring in ϕ) and returns an OBDD corresponding to the semantics of ϕ.

- $B(p, \mathcal{A}) = \mathrm{OBDD}_p(\vec{x})$.
- $B(R_i, \mathcal{A}) = \mathcal{A}[R_i]$.
- $B(\neg\phi, \mathcal{A}) = \neg B(\phi, \mathcal{A})$
- $B(\phi \wedge \psi, \mathcal{A}) = B(\phi, \mathcal{A}) \wedge B(\psi, \mathcal{A})$.
- $B(\phi \vee \psi, \mathcal{A}) = B(\phi, \mathcal{A}) \vee B(\psi, \mathcal{A})$.
- $B(\langle a \rangle \phi, \mathcal{A}) = \exists \vec{x'}(\mathrm{OBDD}_a(\vec{x}, \vec{x}') \wedge B(\phi, \mathcal{A})(\vec{x}'))$
- $B([a]\phi, \mathcal{A}) = B(\neg\langle a \rangle \neg\phi, \mathcal{A})$.
  The second equation uses the dual formulation for $[a]$.
- $B(\mu R.\phi, \mathcal{A}) = \mathit{FIX}(\phi, \mathcal{A}, \text{FALSE-BDD})$.
- $B(\mu R.\phi, \mathcal{A}) = \mathit{FIX}(\phi, \mathcal{A}, \text{TRUE-BDD})$.

The OBDDs for the boolean functions **false** and **true** are denoted by FALSE-BDD and TRUE-BDD respectively. Notice that ϕ has an extra free relational variable R. *FIX* is described in Figure 7.

function $\mathit{FIX}(\phi, \mathcal{A}, B_R)$

result-bdd = B_R
do
    old-bdd = result-bdd
    result-bdd = $B(\phi, \mathcal{A}\langle R \leftarrow \text{old-bdd}\rangle)$
while (not-equal(old-bdd, result-bdd))
return(result-bdd)

Figure 7: Pseudocode for the function *FIX*

Now we give a short example to illustrate our point.

Example 5.1 Assume that the state space $\top$ is encoded by n boolean variables $x_1, \cdots, x_n$. Consider the following formula:

$$\phi \quad = \quad \mu Z.(q \wedge Y \vee \langle a \rangle Z)$$

Notice that the variable Y is free in ϕ. Assume that the interpretation for q is an OBDD $\mathrm{OBDD}_q(\vec{x})$. Similarly, the OBDD corresponding to the program letter a is $\mathrm{OBDD}_a(\vec{x}, \vec{x'})$. Also assume that we are given an association list $\mathcal{A}$ which pairs the OBDD $B_Y(\vec{x})$ with Y. In the routine *FIX* the OBDD result-bdd is initially set to:

$$N^0(\vec{x}) \quad = \quad \text{FALSE-BDD}$$

Let N^i be the value of result-bdd at the i-th iteration in the loop of the function *FIX*. At the end of the iteration the value of result-bdd is given by:

$$N^{i+1}(\vec{x}) \quad = \quad \mathrm{OBDD}_q(\vec{x}) \wedge B_Y(\vec{x}) \vee \exists \vec{x}'(\mathrm{OBDD}_a(\vec{x}, \vec{x}') \wedge N^i(\vec{x}'))$$

The iteration stops when $N^i(\vec{x}) = N^{i+1}(\vec{x})$.

6 Branching Time Temporal Logics

Let AP be a set of atomic propositions. A Kripke structure over AP is a triple $M = (S, T, L)$, where

- S is a finite set of *states*,
- $T \subseteq S \times S$ is a *transition relation*, which must be total (i.e., for every state s_1 there exists a state s_2 such that $(s_1, s_2) \in T$).
- $L : S \rightarrow 2^{AP}$ is a *labeling function* which associates with each state a set of atomic propositions that are true in the state.

There are two types of formulas in the temporal logic $CTL^\star$: *state formulas* (which are true in a specific state) and *path formulas* (which are true along a specific path). The state operators in $CTL^\star$ are: **A** ("for all computation paths"), **E** ("for some computation paths"). The path operators in $CTL^\star$ are: **G** ("always"), **F** ("sometimes"), **U** ("until"), and **V** ("unless"). Let AP be a set of atomic propositions. A state formula is either:

- p, if $p \in AP$;
- $\neg f$ or $f \vee g$, where f and g are state formulas; or
- $\mathbf{E}(f)$ where f is a path formula.

Path formulas are defined as follows:

- every state formula is a path formula; and
- if f and g are path formulas, then $\neg f$, $f \vee g$, $\mathbf{X}\, f$, $f\, \mathbf{U}\, g$, and $f\, \mathbf{V}\, g$ are path formulas.

$CTL^\star$ is the set of state formulas generated by the above rules.

We define the semantics of $CTL^\star$ with respect to a Kripke structure $M = (S, T, L)$. A *path* in M is an infinite sequence of states $\pi = s_0, s_1, ...$ such that, for every $i \geq 0, (s_i, s_{i+1}) \in T$. π^i denotes the *suffix* of π starting at s_i. $\pi[i]$ denotes the i-th state on the path π. The starting state of path π is $\pi[0]$. We use the standard notation to indicate that a state formula f holds in a structure. $M, s \models f$ means that f holds at the state s in the structure M. Similarly, $M, \pi \models f$ means that the path formula f is true along the path π. Assume that f_1 and f_2 are state formulas and g_1 and g_2 are path formulas, then the relation $\models$ is defined inductively as follows:

1. $s \models p \Leftrightarrow p \in L(s)$

2. $s \models \neg f_1 \Leftrightarrow s \not\models f_1$

3. $s \models f_1 \vee f_2 \Leftrightarrow s \models f_1 \,\mathrm{or}\, s \models f_2$

4. $s \models \mathbf{E}(g_1) \Leftrightarrow$ there exists a path π starting with s such that $\pi \models g_1$

5. $\pi \models f_1 \Leftrightarrow \pi[0] \models f_1$

6. $\pi \models \neg g_1 \Leftrightarrow \pi \not\models g_1$

7. $\pi \models g_1 \vee g_2 \Leftrightarrow \pi \models g_1$ or $\pi \models g_2$

8. $\pi \models \mathbf{X}\, g_1 \Leftrightarrow \pi^1 \models g_1$

9. $\pi \models g_1 \,\mathbf{U}\, g_2 \Leftrightarrow$ there exists $k \geq 0$ such that $\pi^k \models g_2$ and for all $0 \leq j < k, \pi^j \models g_1$.

10. $\pi \models g_1 \,\mathbf{V}\, g_2 \Leftrightarrow$ for every $k \geq 0$, if $\pi^j \not\models g_1$ for all $0 \leq j < k$, then $\pi^k \models g_2$.

CTL is the subset of $CTL^\star$ in which the path formulas are restricted to be:

- if f and g are state formulas, then $\mathbf{X}\, f$, $f \,\mathbf{U}\, g$, and $f \,\mathbf{V}\, g$ are path formulas.

The basic modalities of CTL are $\mathbf{EX}\, f$, $\mathbf{EG}\, f$, and $\mathbf{E}(f \,\mathbf{U}\, g)$, where f and g are again CTL formulas. The operator $\mathbf{E}(f \,\mathbf{V}\, g)$ can be expressed as follows:

$$\begin{aligned} \mathbf{E}(f \,\mathbf{V}\, g) &= \mathbf{E}((\neg f \wedge g) \,\mathbf{U}\, f \wedge g) \vee \mathbf{EG}(\neg f \wedge g) \\ \mathbf{EF}\, f &= \mathbf{E}(true \,\mathbf{U}\, f) \end{aligned}$$

The operators $\mathbf{AG}\, f$, $\mathbf{AF}\, f$ and $\mathbf{A}(f \,\mathbf{U}\, g)$ can be expressed in terms of the basic modalities described above.

$$\begin{aligned} \mathbf{AG}\, f &= \neg\, \mathbf{EF} \neg f \\ \mathbf{AF}\, f &= \neg\, \mathbf{EG} \neg f \\ \mathbf{A}(f \,\mathbf{U}\, g) &= \neg \mathbf{E}(\neg f \,\mathbf{V}\, \neg g) \end{aligned}$$

Next, we discuss the issue of *fairness.* In many cases, we are only interested in the correctness along paths with certain conditions. For example, if we are verifying a protocol with a scheduler, we may wish to consider only executions where processes are not ignored by the scheduler, i.e., every process is given a chance to run infinitely often. This type of fairness constraint cannot be expressed in CTL [7]. In order to handle such properties we have to modify the semantics of CTL. A *fairness constraint* can be an arbitrary set of states, usually described by a CTL formula. Generally, there will be several fairness constraints. In this paper we will denote the set of all fairness constraints by $H = \{h_1, \cdots, h_n\}$. We have the following definition of a *fair* path.

Definition 6.1 Given a Kripke Structure $M = (S, T, L)$ and a set of fairness constraints $H = \{h_1, \cdots, h_n\}$, a path π in M is called *fair* iff each CTL formula h_i is satisfied infinitely often on the path π.

The semantics of CTL has to be modified to handle fairness constraints H. The basic idea is to restrict path quantifiers to fair paths. The formal definition is given below:

- $s \models \mathbf{EX}_H\, f$ iff there exists a fair path π starting from the state s such that $\pi[1] \models f$.
- $s \models \mathbf{E}(g_1\, \mathbf{U}_H\, g_2)$ iff there exists a fair path π starting from the state s and there exists $k \geq 0$ such that $\pi[k] \models g_2$ and for all $0 \leq j < k, \pi[j] \models g_1$.
- $s \models \mathbf{EG}_H\, f$ iff there exists a fair path π starting from the state s such that for all $i \geq 0$, $\pi[i] \models f$.

7 Translating CTL into the μ-Calculus

In this section we give a translation of CTL into the propositional μ-calculus. The algorithm Tr takes as its input a CTL formula and outputs an equivalent μ-calculus formula with only one action a.

- $Tr(p) = p$.
- $Tr(\neg f) = \neg Tr(f)$.
- $Tr(f \wedge g) = Tr(f) \wedge Tr(g)$.
- $Tr(\mathbf{EX}\, f) = \langle a \rangle Tr(f)$.
- $Tr(\mathbf{E}(f\, \mathbf{U}\, g)) = \mu Y.(Tr(g) \vee (Tr(f) \wedge \langle a \rangle Y))$.
- $Tr(\mathbf{EG}\, f) = \nu Y.(Tr(f) \wedge \langle a \rangle Y)$.

Note, that any resulting μ-calculus formula is closed. Therefore, we can omit the environment in the set $[\![\phi]\!]_M$.

Lemma 7.1 Let $M = (S, T, L)$ be a Kripke Structure, f be a CTL formula, and a be an action with interpretation T. Consider the predicate transformer τ.

$$\begin{aligned} \tau(Z) &= f \wedge \langle a \rangle Z \\ &= \{s \in S \mid s \models f \ \wedge \ \exists s' \in S((s, s') \in T \ \wedge \ s' \in Z)\} \end{aligned}$$

τ satisfies the following conditions:

- τ is monotonic.
- Let $\tau^{i_0}(\top)$ be the limit of the sequence $\top \subseteq \tau(\top) \subseteq \cdots$. For every $s \in S$, if $s \in \tau^{i_0}(\top)$ then $s \models f$, and there is a state s' such that $(s, s') \in T$ and $s' \in \tau^{i_0}(\top)$.

Proof: Let $P_1 \subseteq P_2$. In this case $\langle a \rangle P_1 \subseteq \langle a \rangle P_2$, i.e., the successor relation is monotonic. Therefore, $\tau(P_1) \subseteq \tau(P_2)$. Since $\tau^{i_0}(\top)$ is the fixpoint of the predicate transformer τ, we have the following equation:

$$\tau(\tau^{i_0}(\top)) \;\; = \;\; \tau^{i_0}(\top)$$

Let $s \in \tau^{i_0}(\top)$. Using the equation given above we get that $s \in \tau(\tau^{i_0}(\top))$. By definition of τ we get that $s \models f$ and there exists a state s', such that $(s, s') \in T$ and $s' \in \tau^{i_0}(\top)$. $\square$

The theorem given below proves the correctness of the translation algorithm Tr.

Theorem 7.1 Let $M = (S, T, L)$ be the underlying Kripke Structure. Let ϕ be a CTL formula. Let the interpretation of the action a be T. An atomic proposition p in $Tr(\phi)$ has the interpretation $L(p)$. The set of states $\top$ is S. In this case, for all $s \in S$

$$s \models \phi \;\; \Leftrightarrow \;\; s \in [\![Tr(\phi)]\!]_M$$

Proof: The proof is by structural induction on ϕ.

- $\phi = p$: In this case the result is true by definition.
- $\phi = \neg f$: By definition $[\![Tr(\phi)]\!]_M = S - [\![Tr(f)]\!]_M$. The result follows by using the induction hypothesis on f.
- $\phi = f \wedge g$: By definition $[\![Tr(\phi)]\!]_M = [\![Tr(f)]\!]_M \cap [\![Tr(g)]\!]_M$. The result follows by using the induction hypothesis on f and g.
- $\phi = \mathbf{EX}\, f$: Let S_f be the set of states where f is true. By the induction hypothesis, $[\![Tr(f)]\!]_M = S_f$. The set of states satisfying ϕ is the set of states S_1 which have a successor in S_f. It is clear from the semantics of $\langle a \rangle$ that $[\![Tr(\phi)]\!]_M = S_1$.
- $\phi = \mathbf{EG}\, f$: Let Y_1 be the set of states s such that $s \models \mathbf{EG}\, f$. Let $\tau : 2^S \rightarrow 2^S$ be the following predicate transformer

  $$\tau(Z) \;\; = \;\; [\![Tr(f)]\!]_M \cap ([\![\langle a \rangle X]\!]_M \, e\, [X \leftarrow Z])$$

  By definition, the greatest fixpoint of τ is given by $\bigcap_i \tau^i(\top)$, where $\tau^0(\top) = \top$, and $\tau^{i+1}(\top) = \tau(\tau^i(\top))$. Using the semantics of **EG** we get that if $s \in Y_1$, then there exists a path π starting from s such that each state on the path satisfies f. Therefore, if $s \in Y_1$, then s has a successor s' such that $(s, s') \in T$, $s \models f$, and $s' \models \mathbf{EG}\, f$. Hence Y_1 is a fixpoint for the predicate transformer τ, i.e.,

  $$\tau(Y_1) \;\; = \;\; Y_1$$

  Since $\bigcap_i \tau^i(\top)$ is the greatest fixpoint of τ, we have the following inclusion:

  $$Y_1 \;\; \subseteq \;\; \bigcap_i \tau^i(\top)$$

Now assume that $s \in \bigcap_i \tau^i(\top)$. By Lemma 7.1, s is the start of an infinite path π such that each state s' on the path π satisfies f. Therefore, we have the following inclusion:

$$Y_1 \;\supseteq\; \bigcap_i \tau^i(\top)$$

Using the two equations we get that Y_1 is the greatest fixpoint of the predicate transformer τ.

- $\phi = \mathbf{E}(f\,\mathbf{U}\,g)$: Let S_1 be the set of states s such that $s \models \mathbf{E}(f\,\mathbf{U}\,g)$. Let $\tau : 2^S \to 2^S$ be the following predicate transformer:

$$\tau(Z) \;=\; [\![Tr(g)]\!]_M \cup ([\![Tr(f)]\!]_M \cap ([\![\langle a \rangle X]\!]_M\, e\, [X \leftarrow Z]))$$

First, we will show that S_1 is a fixpoint of τ, i.e.,

$$\tau(S_1) \;= S_1$$

By definition, $s \models \mathbf{E}(f\,\mathbf{U}\,g)$ iff there exists a path π starting from s such that there exists a $k \geq 0$ with the property that $\pi^k \models g$ and $\pi^i \models f$ (for $0 \leq i < k$). Equivalently, $s \models \mathbf{E}(f\,\mathbf{U}\,g)$ iff $s \models g$ or $s \models f$ and there exists a state s_1 such $(s, s_1) \in T$ and $s_1 \models \mathbf{E}(f\,\mathbf{U}\,g)$. From this condition it is clear that S_1 is a fixed point of the predicate transformer τ. By definition, the least fixpoint of τ is given by

$$\bigcup_i \tau^i(\perp)$$

Since S_1 is a fixpoint for τ, we have that

$$S_1 \;\supseteq\; \bigcup_i \tau^i(\perp)$$

Next we prove that

$$S_1 \;\subseteq\; \bigcup_i \tau^i(\perp)$$

which proves that S_1 is equal to the least fixpoint of the predicate transformer τ. By definition, if $s \in S_1$, then there exists a path π and a $k \geq 0$ such that $\pi^k \models g$ and $\pi^j \models f$ (for $j < k$). We will prove by induction on k that $s \in \tau^k(\perp)$. The basis case is trivial. If $k = 0$, then $s \models g$ and therefore $s \in \tau(\perp)$, which is equal to $[\![Tr(g)]\!]_M \cup ([\![Tr(f)]\!]_M \cap [\![\langle a \rangle \perp]\!]_M) = [\![Tr(g)]\!]_M$.

For the inductive step, assume that the above claim holds for every s and every $k \leq m$. Let s be the start of a path $\pi = s_0, s_1, \cdots$ such that $s_{m+1} \models g$ and for every $i < m+1$, $s_i \models f$. By induction hypothesis $s_1 \in \tau^m(\perp)$. Notice that $s_0 = s \in [\![Tr(f)]\!]_M$ and $s \in \langle a \rangle \tau^m(\perp)$. Therefore, by definition $s \in \tau^{m+1}(\perp)$. Hence, if $s \in S_1$, then $s \in \bigcup_i \tau^i(\perp)$.

Using Theorem 7.1 we have the following result:

Theorem 7.2 Given a Kripke Structure $M = (S, T, L)$, an initial state $s_0 \in S$, and a *CTL* formula f, one can decide in $O(|S||f|)$ iterations whether $M, s_0 \models f$. Where $|f|$ denotes the number of symbols in the formula f.

Proof: Consider the following formula:

$$\nu Y.(\mu Z.(q \vee (p \wedge \langle a \rangle Z)) \wedge \langle a \rangle Y)$$

Notice that the formula given above is $Tr(\mathbf{EG}(\mathbf{E}(p\ \mathbf{U}\ q)))$. Since the inner least fixpoint does not use the relational variable Y (associated with the outer greatest fixpoint), we can compute it first and reuse that value in the outer fixpoint computation. Therefore, if we compute the inner fixpoint first, we can evaluate the formula given above in $O(2|S|)$ iterations. Notice that given a *CTL* formula f, $Tr(f)$ has the property that the inner fixpoints never use the variables associated with the outer fixpoint. By evaluating the fixpoints in the nesting order (evaluating the inner fixpoints first), we do not have to recompute the fixpoints. Therefore, the total complexity is the sum of the complexities for evaluating each fixpoint independently. This is bounded by $O(|S||f|)$.[1] $\Box$

Given fairness constraints $H = \{h_1, \cdots, h_n\}$, we extend the translation algorithm Tr in the following way:

- $Tr(\mathbf{EG}_H\, f) = \nu Y.\Big(Tr(f) \wedge \langle a \rangle \bigwedge_{i=1}^{n} \mu X.\, [(Tr(f) \wedge \langle a \rangle X) \vee (Y \wedge Tr(h_i))]\Big)$

We introduce the following formula which is satisfied at a state s iff there is a fair path π starting from s.

- $fair \ = \ \mathbf{EG}_H\ True$
- $Tr(\mathbf{EX}_H\, f) = \langle a \rangle (Tr(f) \wedge Tr(fair))$.
- $Tr(\mathbf{E}(f\ \mathbf{U}_H\ g)) = \mu Y.(Tr(g) \wedge Tr(fair) \vee (Tr(f) \wedge \langle a \rangle Y))$.

We will give an informal proof of correctness for the $\mathbf{EG}_H$ case. Consider the following formula:

$$\nu Y.(P \wedge \langle a \rangle \mu X.\, [(P \wedge \langle a \rangle X) \vee (Y \wedge h)])$$

This corresponds to the formula $Tr(\mathbf{EG}_H\, f)$, where $H = \{h\}$ and $P = Tr(f)$. First, note that the condition "h holds infinitely often along a path" is equivalent to saying that from any point along that path in a finite number of steps we will reach a state where h holds. To understand the formula given above, notice that $\mu X.((P \wedge \langle a \rangle X) \vee (Y \wedge h))$ means that "P holds until $Y \wedge h$, and $Y \wedge h$ is reachable in a finite number of steps". Since the outer fixed point $\nu Y.(P \wedge \cdots)$ indicates that this property holds globally along the path, the formula exactly corresponds to the desired property.

[1] By definition of alternation depth given in [1], the formula $Tr(f)$ always has alternation depth one. Hence, the linear complexity of *CTL* model checking follows directly from the algorithm in [1].

8 Simulation Preorders and Bisimulation Equivalences.

8.1 Simulation and bisimulation.

In this section we will use essentially the same definition of a transition system that was introduced in Section 2, except for two special program letters τ and ε. The letter τ represents the *idle* action; its interpretation is always fixed: $T(\tau) = \{(s,s) \mid s \in S\}$. The program letter ε denotes the *invisible* action from CCS [16] and will be used in the definition of the weak simulation and bisimulation relations [17].

Definition 8.1 A relation $\mathcal{E} \subseteq S \times S$ is called a *simulation relation*, if for every $(s,s') \in \mathcal{E}$ the following condition holds:

$$\forall a \in Act. \forall q \in S. \text{ if } s \xrightarrow{a} q \text{ then } \exists q' \in S. s' \xrightarrow{a} q' \text{ and } (q,q') \in \mathcal{E}.$$

Definition 8.2 A relation $\mathcal{E} \subseteq S \times S$ is called a *bisimulation* if $\mathcal{E}$ and $\mathcal{E}^{-1}$ are both simulation relations. In other words, $\mathcal{E}$ satisfies the following conditions: $(s,s') \in \mathcal{E}$ iff

$$\begin{array}{ll} (i) & \forall a \in Act. \forall q \in S. \text{ if } s \xrightarrow{a} q \text{ then } \exists q' \in S. s' \xrightarrow{a} q' \text{ and } (q,q') \in \mathcal{E}; \\ (ii) & \forall a \in Act. \forall q' \in S. \text{ if } s' \xrightarrow{a} q' \text{ then } \exists q \in S. s \xrightarrow{a} q \text{ and } (q,q') \in \mathcal{E}. \end{array}$$

We define the *simulation preorder* as follows:

$$s \preceq s' \text{ iff there exists a simulation relation } \mathcal{E} \text{ such that } (s,s') \in \mathcal{E}.$$

We define *bisimulation equivalence* in a similar manner:

$$s \sim s' \text{ iff there exists a bisimulation relation } \mathcal{E} \text{ such that } (s,s') \in \mathcal{E}.$$

It is straightforward to check that $\preceq$ is a preorder. In fact, it is the maximal simulation relation under inclusion. It is also possible to show that bisimulation equivalence $\sim$ is an equivalence relation. Moreover, it is the maximal bisimulation relation under inclusion.

8.2 Encoding simulation and bisimulation into the μ-calculus.

In order to check if the initial states of two transition systems are bisimilar using the propositional μ-calculus, we first need to construct a new transition system. Given two transition systems $M = (S,T,L)$ over Act and $M' = (S,T',L')$ over Act, we define the product $\tilde{M} = M \times M'$ over $\tilde{Act}$ as follows: $\tilde{M} = (\tilde{S}, \tilde{T}, \tilde{L})$, where

- $\tilde{Act} = Act \times Act = \{a\tau, \tau a \mid a \in Act \text{ and } b \in Act\}$,
- $\tilde{S} = S \times S$,
- $(s,s') \xrightarrow{ab} (q,q')$ iff $s \xrightarrow{a} q$ and $s' \xrightarrow{b} q'$.
- $\tilde{L}$ may be arbitrary in this case.

We assume that M and M' have the same state and action sets. This is a technical issue because we can always define the transition systems on larger state and action sets.

Theorem 8.1 Let s and s' be the states of the two transition systems M and M'. Then $s \preceq s'$ iff the following formula holds in the state (s, s') of the transition system $\tilde{M}$:

$$\nu X.\Big(\bigwedge_{a \in Act} [a\tau]\,\langle\tau a\rangle X\Big)$$

Proof: Consider the definition of a simulation relation:

$$(s, s') \in \mathcal{E} \text{ iff } \forall a \in Act.\forall q \in S. \text{ if } s \xrightarrow{a} q \text{ then } \exists q' \in S.s' \xrightarrow{a} q' \text{ and } (q, q') \in \mathcal{E}.$$

This is the same as the equation

$$\mathcal{E} \equiv \bigwedge_{a \in Act} [a\tau]\,\langle\tau a\rangle \mathcal{E}$$

in the transition system $\tilde{M}$ (see the semantics of modalities in Section 2, and definition of $\tilde{M}$). Therefore, $\mathcal{E}$ is a simulation relation iff it is a fixpoint of the above equation. We show that $\preceq$ is the greatest fixpoint. Let $\mathcal{Y}$ denote the set $\nu X.\Big(\bigwedge_{a \in Act} [a\tau]\,\langle\tau a\rangle X\Big)$.

$\subseteq$: $s \preceq s'$ implies that $(s, s') \in \mathcal{E}$ for some simulation relation $\mathcal{E}$. Since $\mathcal{E}$ is a fixpoint of the equation, we have $\mathcal{E} \subseteq \mathcal{Y}$ by definition of the greatest fixpoint, therefore $(s, s') \in \mathcal{Y}$.

$\supseteq$: Let $(s, s') \in \mathcal{Y}$. Since $\mathcal{Y}$ satisfies the fixpoint equation, it is a simulation relation, hence $s \preceq s'$.

$\square$

Theorem 8.2 Two states s and s' are bisimilar ($s \sim s'$) iff the following formula holds in the state (s, s') of the model $\tilde{M}$:

$$\nu X.\Big(\bigwedge_{a \in Act} [a\tau]\,\langle\tau a\rangle X \wedge [\tau a]\,\langle a\tau\rangle X\Big)$$

The proof of this theorem is almost identical to the proof of the previous theorem.

Obviously, the alternation depth of the formulas is one, therefore the complexity is $O(|\tilde{S}|)$ iterations, where the size of $\tilde{S}$ is $|S|^2$. The time complexity is $O(|\tilde{S}||\tilde{Act}||\tilde{M}|) = O(|S|^2|Act||M||M'|)$. An algorithm for bisimulation equivalence with time complexity $O(|Act|(|T| + |T'|)\log(|S|))$ is given in [18]. However, it is not clear if this algorithm can be modified to compute the simulation preorder or if it can be adapted to use OBDDs.

8.3 Weak simulation and bisimulation.

Weak simulation preorder and weak bisimulation equivalence require a more elaborate encoding. The definition of weak (bi)simulation is similar to (bi)simulation. The difference is that each of the transition systems is allowed to perform an unbounded but finite number of invisible actions ε. Formally, first define a relation $\Rightarrow$ by

$$s \overset{a}{\Rightarrow} q \text{ iff } \exists s_1, s_2.s \xrightarrow{\varepsilon^*} s_1 \xrightarrow{a} s_2 \xrightarrow{\varepsilon^*} q,$$

and $s \xrightarrow{\varepsilon^*} q$ means that q is reachable from s by 0 or more ε-transitions.

Definition 8.3 A relation $\mathcal{E} \subseteq S \times S$ is called a *weak simulation* with invisible action ε, when $(s, s') \in \mathcal{E}$ iff

$$\forall a \in Act. \forall q \in S. \text{ if } s \stackrel{a}{\Rightarrow} q \text{ then } \exists q' \in S. s' \stackrel{a}{\Rightarrow} q' \text{ and } (q, q') \in \mathcal{E}.$$

Definition 8.4 A relation $\mathcal{E} \subseteq S \times S$ is called a *weak bisimulation* with invisible action ε, if $(s, s') \in \mathcal{E}$ iff

$$\begin{array}{rl} (i) & \forall a \in Act. \forall q \in S. \text{ if } s \stackrel{a}{\Rightarrow} q \text{ then } \exists q' \in S. s' \stackrel{a}{\Rightarrow} q' \text{ and } (q, q') \in \mathcal{E}; \\ (ii) & \forall a \in Act. \forall q' \in S. \text{ if } s' \stackrel{a}{\Rightarrow} q' \text{ then } \exists q \in S. s \stackrel{a}{\Rightarrow} q \text{ and } (q, q') \in \mathcal{E}. \end{array}$$

As before, we introduce a preorder called *weak simulation preorder*:

$$s \preceq\!\!\!\lessdot\; s' \text{ iff there exists a weak simulation relation } \mathcal{E} \text{ such that } (s, s') \in \mathcal{E},$$

and an equivalence called *weak bisimulation equivalence*:

$$s \approx s' \text{ iff there exists a weak bisimulation relation } \mathcal{E} \text{ such that } (s, s') \in \mathcal{E}.$$

To encode the weak (bi)simulation in the propositional μ-calculus we again make use of the transition system $\tilde{M}$. Define the abbreviations:

$$\begin{array}{l} \langle \varepsilon^*; a; \varepsilon^* \rangle \phi \equiv_{df} \mu X.(\langle a \rangle (\mu Y.\phi \vee \langle \varepsilon \rangle Y) \vee \langle \varepsilon \rangle X) \\ [\varepsilon^*; a; \varepsilon^*]\, \phi \equiv_{df} \neg \langle \varepsilon^*; a; \varepsilon^* \rangle \neg \phi \end{array}$$

To understand the formulas better, notice that informally they can be viewed as translations of $\mathbf{EF}(\langle a \rangle\, \mathbf{EF}\, \phi)$ and $\mathbf{AG}([a]\, \mathbf{AG}\, \phi)$, where CTL operators refer to ε-paths. Now, it is straightforward to show that the following theorems hold:

Theorem 8.3 Let s and s' be states of the two transition systems. Then $s \preceq\!\!\!\lessdot\; s'$ iff the following formula holds in the state (s, s') of the transition system $\tilde{M}$:

$$\nu X. \Big(\bigwedge_{a \in Act} [(\varepsilon\tau)^*; a\tau; (\varepsilon\tau)^*]\, \langle (\tau\varepsilon)^*; \tau a; (\tau\varepsilon)^* \rangle X \Big)$$

Theorem 8.4 Two states s and s' are weakly bisimilar ($s \approx s'$) iff the following formula holds in the state (s, s') of the model $\tilde{M}$:

$$\nu X. \Big(\bigwedge_{a \in Act} \begin{array}{l} [(\varepsilon\tau)^*; a\tau; (\varepsilon\tau)^*]\, \langle (\tau\varepsilon)^*; \tau a; (\tau\varepsilon)^* \rangle X \wedge \\ [(\tau\varepsilon)^*; \tau a; (\tau\varepsilon)^*]\, \langle (\varepsilon\tau)^*; a\tau; (\varepsilon\tau)^* \rangle X \end{array} \Big)$$

Although there are five levels of nesting in these formulas, the alternation depth is only two. Therefore, we can compute it by the algorithm given in [11] using $O(|\tilde{S}|^2 |\tilde{Act}|^2)$ iterations or $O(|\tilde{Act}|^3 |\tilde{M}||\tilde{S}|^2)$ time. Recall that each iteration can take upto $O(|\tilde{Act}||\tilde{M}|)$ time. However, there is another algorithm by H. Andersen [2] that can compute the fixpoints in $O(|Act|^2 |\tilde{S}||\tilde{M}|)$ time. The algorithm in [18] can also be adapted to compute weak bisimulation equivalence by precomputing the transitive closure of the ε relation. However, the expense of this step dominates the cost of the entire computation. Again, it is not clear that OBDDs can be used in the last two algorithms.

9 Conclusion

In this paper, we show the importance of the propositional μ-calculus by giving translations of various graph-based verification algorithms into the μ-calculus. We also present an OBDD based algorithm for μ-calculus model checking which has proved to be extremely efficient in practice. Finally, we give the best known algorithm for evaluating μ-calculus formulas. However, there is still much work to be done in each of these areas.

Although OBDDs do not reduce the worst-case complexity of the model checking problem for the μ-calculus, their use in model checking has had an enormous effect on formal verification. Before the use of OBDDs, it was only possible to verify models with at most 10^6 states [7]. By using the OBDD techniques described in this paper, in practice, it is now possible to verify examples with up to 10^{120} states and several hundred state variables [5]. However, there is no theoretical framework which explains when OBDDs will work well in practice. Our algorithm does not depend on the data structure used to represent boolean functions, so it should be possible to use any better data structures that may be discovered.

In addition to the verification problems we have considered, there are other graph theoretic problems that can be encoded in the μ-calculus. An important question is how useful these OBDD and fixpoint techniques are for problems like finding minimum spanning trees, determining graph isomorphism, etc. For example, let $E(u, v)$ be the edge relation for a directed graph and let each vertex v be a state encoded by an assignment $\vec{v}$ to the boolean variables $\vec{x} = x_1, \ldots, x_k$. The formula

$$\phi(\vec{x}) \equiv \mu R.\vec{x} \vee \langle a \rangle R$$

computes the set of states reachable from the state encoded by the assignment to $\vec{x}$, where the interpretation for the program letter a is the edge relation E. Then the graph satisfies the formula

$$[\vec{u} \rightarrow \phi(\vec{v})] \wedge [\vec{v} \rightarrow \phi(\vec{u})]$$

if and only if the two vertices u and v are in the same strongly connected component. In general, the graph is strongly connected if and only if every vertex satisfies the formula

$$\forall \vec{x}.\phi(\vec{x}).$$

Although strictly speaking this is not a μ-calculus formula according to our syntax, recall that we allow quantification over boolean variables in our translation of the μ-calculus into OBDDs.

We also discuss efficient evaluation algorithms, which exploit monotonicity properties when evaluating fixpoints. However, these algorithms remain exponential in the alternation depth. We conjecture that there is no polynomial-time algorithm for determining if a state satisfies a given formula. Consider 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. Furthermore, 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. The state in question satisfies the formula if and only if it is in the set computed by some run of the algorithm. Also note that we can negate formulas, so the complexity of determining if a state satisfies a formula is the same as the complexity of determining if a state does not satisfy the formula. Thus, the problem is in the intersection of NP and co-NP. This suggests that our conjecture will be very difficult to prove.

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] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126:3–30, 1994.

[3] 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.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 13(4):401–424, April 1994.

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[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, April 1986.

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

[9] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In Bochmann and Probst [3].

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

[13] K. G. Larsen. Efficient local correctness checking. In Bochmann and Probst [3].

[14] D. Long, A. Browne, E. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D. Dill, editor, *Proceedings of the 1995 Workshop on Computer-Aided Verification*, pages 338–350. Springer-Verlag, June 1994.

[15] A. Mader. Tableau recycling. In Bochmann and Probst [3].

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

[17] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[18] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 6 (16):973–989, 1987.

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

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

[21] G. Winskel. A note on model checking the modal ν-calculus. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming*, 1989.