Reasoning about The Past with Two-Way Automata

Moshe Y. Vardi[1*]

Rice University, Department of Computer Science, Houston, TX 77005-1892, USA
Email: vardi@cs.rice.edu
URL: http://www.cs.rice.edu/~vardi

Abstract. The μ-calculus can be viewed as essentially the "ultimate" program logic, as it expressively subsumes all propositional program logics, including dynamic logics, process logics, and temporal logics. It is known that the satisfiability problem for the μ-calculus is EXPTIME-complete. This upper bound, however, is known for a version of the logic that has only forward modalities, which express weakest preconditions, but not backward modalities, which express strongest postconditions. Our main result in this paper is an exponential time upper bound for the satisfiability problem of the μ-calculus with both forward and backward modalities. To get this result we develop a theory of two-way alternating automata on infinite trees.

1 Introduction

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators. It was introduced in [22], following earlier studies of fixpoint calculi in the theory of program correctness [7,30,35]. Over the past 15 years, the μ-calculus has been established as essentially the "ultimate" program logic, as it expressively subsumes all propositional program logics, including dynamic logics such as PDL [13], process logics such as YAPL [50], and temporal logics such as CTL* [8]. The μ-calculus has gained further prominence with the discovery that its formulas can be evaluated symbolically in a natural way [4], leading to industrial acceptance of computer-aided verification [1]. More recently, the μ-calculus has found a new application domain in the theory of *description logics* in Artificial Intelligence [14]. As a result of this prominence, the μ-calculus has been the subject of extensive research; in particular, researchers focused on the truth problem and the satisfiability problem.

In the truth problem, we are asked to verify whether a given formula holds in a given state of a given Kripke structure (which is the essence of model checking). In spite of extensive research (see [2,3,5,11,12,46,52]), the precise complexity of this problem is still open; it is known to be in NP$\cap$co-NP and PTIME-hard.

* Supported in part by NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation.

In contrast, the complexity of the satisfiability problem, where we are asked to decide if a given formula holds in some state of some Kripke structure, has been precisely identified. An exponential time lower time bound follows from the lower bound for PDL in [13], and an exponential time upper time bound was shown in [9], following a sequence of improving upper bounds in [24,40,49].

The exponential time upper bound for the μ-calculus was shown, however, only for a version of the logic that has only *forward* modalities. The formula $\langle a \rangle \varphi$ holds in a state s of a Kripke structure M when φ holds in some *a-successor* of s; in contrast, the "*backward*" formula $\langle a^- \rangle \varphi$ holds in s if φ holds in some *a-predecessor* of s. Here a^- describes the *converse* of the atomic program a. Essentially, forward modalities express weakest preconditions, while backward modalities express strongest postconditions. Backward modalities correspond to reasoning about the past. There is now a significant body of evidence of the usefulness of reasoning about the past in the context of program correctness [19,27]. For example, it is shown in [32,33] that past temporal connective can conveniently replace history variables in compositional verification. Backward modalities also have a counterpart in description logics, where they correspond to *inverse roles* [14].

The importance of backword modalities motivated the study of procedures for the satisfiability problem for logics that include them [16,25,31,39,44,45,51]. (Backward modalities do not, in general, pose any difficulty to truth-checking procedures.) The challenge in developing such decision procedures is that the interaction of backward modalities with other constructs of the logic can be quite subtle. For example, it has been observed by Halpern that backward modalities interact with the assumption of *determinism* of atomic programs in dynamic logic in such a way that the *finite-model property* is lost [51]. Similarly, backward modalities interact with the *Repeat* construct of *Repeat-PDL*, posing a great difficulty to the development of decision procedures. The first elementary decision procedure for *Repeat-Converse-PDL* was octuply exponential [41]. This was improved later to a quadruply exponential procedure [39]. Finally, combining the techniques in [9] with the techniques in [44] leads to a singly exponential procedure.

Because of the subtlety of dealing with backward modalities, the satisfiability problem for the *full* μ-calculus, which has both forward and backward modalities, is still open. Our main result in this paper is an exponential time upper bound for the problem. The approach we take is the automata-theoretic approach advocated in [9,39,51]. We first show that even though the full μ-calculus does not have the finite-model property, it does have the *tree-model property*. (As argued in [48], the tree-model property, which asserts that if a formula is satisfiable then it is satisfiable by a bounded-degree infinite tree structure, offers an explanation for the robust decidability of many propositional program logics.) This proof requires a careful analysis of well-founded regeneration sequences of least-fixpoint formulas. We then show how a formula φ can be translated to an automaton A_φ on infinite trees that accepts precisely the tree models of φ. To check whether φ is satisfiable it suffices then to solve the emptiness problem for A_φ.

Earlier papers that employed the automata-theoretic approach used nondeterministic tree automata [9,39,51]. The translation from formulas to nondeterministic automata is nontrivial; for example, the translation in [51] is exponential and consists of a sequence of successive translations. As argued in [28] and

then utilized in [2,47], it is easier to translate formulas to *alternating* automata. Alternating tree automata generalize nondeterministic tree automata by allowing multiple successor states to go down along the same branch of the tree. It is known that while the translation from branching temporal logic formulas to nondeterministic tree automata is exponential, the translation to alternating tree automata is linear [2,28]. Similarly, there is a simple translation from μ-calculus formulas to alternating tree automata [2,10]. Alternating tree automata as defined in [29], however, cannot easily handle backwards modalities, since they are *one-way* automata. To deal with backward modalities we introduce *two-way* alternating automata on infinite trees, based on an analogous notion of two-way automata on *finite trees* in [38].

It remains then to solve the emptiness problem for two-way alternating tree automata. Alternating tree automata can be viewed as infinite games [29]; this holds for both one-way and two-ways automata. It is shown in [20] that under certains conditions, which hold here, the winning player has a *memoryless* strategy in these games. We use this to show that two-way alternating tree automata can be translated to equivalent one-way nondeterministic tree automata with an exponential blowup. The emptiness problem can then be solved by using known algorithms for emptiness of nondeterministic tree automata [9,26,34]. This yields an exponential time upper bound for the emptiness problem for alternating tree automata, resulting in a bound of the same complexity for satisfiability of the full μ-calculus.

2 Preliminaries

2.1 The μ-Calculus

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [22]. A *signature* Ξ for the μ-calculus consists of a set AP of atomic propositions, a set Var of propositional variables, a set Prog of atomic programs, and a subset DProg of Prog of *deterministic* atomic programs. Deterministic atomic programs correspond to *functional roles* in description logics [15]. Determinism plays a role in the semantics, but not in the syntax. As we will see, having to deal with deterministic programs introduces an additional difficulty that we will have to overcome in order to develop a decision procedure. In the *full* μ-calculus, we associate with each atomic program a its *converse* a^-. A *program* is either an atomic program or its converse. We denote programs by α.

A formula of the full μ-calculus the signature Ξ is either:

- **true**, **false**, p or $\neg p$ for all $p \in \mathsf{AP}$;
- y for all $y \in \mathsf{Var}$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are μ-calculus formulas;
- $\langle\alpha\rangle\varphi$ or $[\alpha]\varphi$, where φ is a μ-calculus formula and α is a program;
- $\mu y.\varphi(y)$ or $\nu y.\varphi(y)$, where $y \in \mathsf{Var}$ and $\varphi(y)$ is a formula.

The only difference between the full μ-calculus and the *standard* μ-calculus is that in the full μ-calculus both atomic programs and their converse are allowed in the modalities $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$, while only atomic programs are allowed in such modalities in the standard μ-calculus. A *sentence* is a formula that contains no free propositional variables. We call μ and ν *fixpoint operators*. We say that a

formula is a *μ-formula* (*ν-formula*), if it is of the form $\mu y.\varphi(y)$ ($\nu y.\varphi(y)$). We use λ to denote a fixpoint operator μ or ν. For a λ-formula $\lambda y.\varphi(y)$, the formula $\varphi(\lambda y.\varphi(y))$ is obtained from $\varphi(y)$ by replacing each free occurrence of y with $\lambda y.\varphi(y)$. For example, if $\varphi(y)$ is $p \vee \langle\alpha\rangle y$, then $\varphi(\mu y.\varphi)$ is $p \vee \langle\alpha\rangle \mu y.(p \vee \langle\alpha\rangle y)$. We call a formula of the form $\langle\alpha\rangle\varphi$ an *existential* formula.

The semantics of the full μ-calculus is defined with respect to a *Kripke structure* $K = \langle W, R, L\rangle$ over the signature Ξ, where W is a set of points, $R : \mathsf{Prog} \to 2^{W \times W}$ assigns to each atomic program a transition relation over W such that $R(a)$ is a partial function for each $a \in \mathsf{DProg}$, and $L : \mathsf{AP} \to 2^W$ assigns to each atomic proposition a set of points. We now extend R to the converse of atomic programs. For each atomic program a, we define $R(a^-)$ to be the relational inverse of $R(a)$, i.e., $R(a^-) = \{(v,u) : (u,v) \in R(a)\}$. Note that $R(a^-)$ for $a \in \mathsf{DProg}$ need not be a function.

Given a Kripke structure $K = \langle W, R, L\rangle$ and a set $\{y_1, \ldots, y_n\}$ of variables in Var, a *valuation* $\mathcal{V} : \{y_1, \ldots, y_n\} \to 2^W$ is an assignment of subsets of W to the variables $y_1, \ldots, y_n$. For a valuation $\mathcal{V}$, a variable y, and a set $W' \subseteq W$, we denote by $\mathcal{V}[y \leftarrow W']$ the valuation obtained from $\mathcal{V}$ by assigning W' to y. A formula φ with free variables among $y_1, \ldots, y_n$ is interpreted over the structure K as a mapping φ^K from valuations to 2^W. Thus, $\varphi^K(\mathcal{V})$ denotes the set of points that satisfy φ with the valuation $\mathcal{V}$. The mapping φ^K is defined inductively as follows:

- $\mathbf{true}^K(\mathcal{V}) = W$ and $\mathbf{false}^K(\mathcal{V}) = \emptyset$;
- For $p \in \mathsf{AP}$, we have $p^K(\mathcal{V}) = L(p)$ and $(\neg p)^K(\mathcal{V}) = W \setminus L(p)$;
- For $y_i \in \mathsf{Var}$, we have $y_i^K(\mathcal{V}) = \mathcal{V}(y_i)$;
- $(\varphi_1 \wedge \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cap \varphi_2^K(\mathcal{V})$;
- $(\varphi_1 \vee \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cup \varphi_2^K(\mathcal{V})$;
- $([\alpha]\varphi)^K(\mathcal{V}) = \{w \in W : \forall w' \text{ such that } (w,w') \in R(\alpha), \text{ we have } w' \in \varphi^K(\mathcal{V})\}$,
- $(\langle\alpha\rangle\varphi)^K(\mathcal{V}) = \{w \in W : \exists w' \text{ such that } (w,w') \in R(\alpha) \text{ and } w' \in \varphi^K(\mathcal{V})\}$,
- $(\mu y.\varphi(y))^K(\mathcal{V}) = \bigcap\{W' \subseteq W : f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'\}$;
- $(\nu y.\varphi(y))^K(\mathcal{V}) = \bigcup\{W' \subseteq W : W' \subseteq f^K(\mathcal{V}[y \leftarrow W'])\}$.

Note that no valuation is required for a sentence. For a point $w \in W$ and a sentence φ, we say that φ holds at w in K, denoted $K, w \models \varphi$ iff $w \in \varphi^K$.

2.2 Alternating Tree Automata

For an introduction to the theory of automata on infinite trees see [42]. An *infinite tree* is a set $T \subseteq \mathbb{N}^+$, such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and, if the tree is *full*, then also $x \cdot c' \in T$ for all $0 < c' < c$, (Here we use $\mathbb{N}$ to denote the *positive* integers.) The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot c$ where $c \in \mathbb{N}$ are the *successors* of x. As a convention, we take $x \cdot 0 = x$ and $(x \cdot i) \cdot -1 = x$ ($\varepsilon \cdot -1$ is undefined). The *branching degree* $d(x)$ denotes the number of different successors x has. If $d(x) = k$ for all nodes x, then we say that the tree is k-ary. An *infinite path* P of T is a prefix-closed set $P \subseteq T$ such that for every $i \geq 0$, there exists a unique $x \in P$ with $|x| = i$. A *labeled tree* over an alphabet Σ is a pair (T, V) where T is a tree and $V : T \to \Sigma$.

Alternating automata on infinite trees generalize nondeterministic tree automata and were first introduced in [29]. Here we describe *two-way* alternating tree automata. For simplicity, we refer first to automata over infinite binary trees.

Consider a nondeterministic tree automaton $A = \langle \Sigma, Q, \delta, q_0, F \rangle$. The transition relation δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of pairs of states. Each such pair suggests a nondeterministic choice for the automaton's next configuration. When the automaton is in a state q and is reading a node x labeled by a letter σ, it proceeds by first choosing a pair $\langle q_1, q_2 \rangle \in \delta(q, \sigma)$ and then splitting into two copies. One copy enters the state q_1 and proceeds to the node $x \cdot 1$, and the other copy enters the state q_2 and proceeds to the node $x \cdot 2$.

Let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false**, and, as usual, $\wedge$ has precedence over $\vee$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. We can represent δ using $\mathcal{B}^+(\{1,2\} \times Q)$. For example, $\delta(q, \sigma) = \{\langle q_1, q_2 \rangle, \langle q_3, q_1 \rangle\}$ can be written as $\delta(q, \sigma) = (1, q_1) \wedge (2, q_2) \vee (1, q_3) \wedge (2, q_1)$, meaning that the automaton can chose between two possibilities. In the first, the copy that proceeds to direction 1 enters the state q_1 and the one that proceeds to direction 2 enters the state q_2. In the second, the copy that proceeds to direction 1 enters the state q_3 and the one that proceeds to direction 2 enters the state q_1.

In nondeterministic tree automata, each conjunction in δ has exactly one element associated with each direction. In alternating automata on binary trees, $\delta(q, \sigma)$ can be an arbitrary formula from $\mathcal{B}^+(\{1,2\} \times Q)$. We can have, for instance, a transition

$$\delta(q, \sigma) = (1, q_1) \wedge (1, q_2) \vee (1, q_2) \wedge (2, q_2) \wedge (2, q_3).$$

This illustrates that several copies may go to the same direction and that the automaton is not required to send copies to all the directions. In *two-way* alternating automata on binary trees, $\delta(q, \sigma)$ can be an arbitrary formula from $\mathcal{B}^+(\{-1, 0, 1, 2\} \times Q)$. We can have, for instance, a transition

$$\delta(q, \sigma) = (-1, q_1) \wedge (0, q_2) \vee (1, q_2) \wedge (2, q_2) \wedge (2, q_3),$$

where the direction 0 refers to the current node, and the direction -1 refers to the parent node. We now generalize this to k-ary trees. Let $[k] = \{-1, 0, 1, \ldots, k\}$. A *two-way alternating automaton* over infinite k-ary trees is a tuple $A = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+([k] \times Q)$ is the transition function, $q_0 \in Q$ is an initial state, and F specifies the acceptance condition.

A run of an alternating automaton A over a labeled tree $\langle T, V \rangle$ is a labeled tree $\langle T_r, r \rangle$ in which every node is labeled by an element of $T \times Q$. A node in T_r, labeled by (x, q), describes a copy of the automaton that is in the state q and reads the node x of T. Note that many nodes of T_r can correspond to the same node of T; there is no one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its successors have to satisfy the transition function. Formally, a run $\langle T_r, r \rangle$ is a Σ_r-labeled tree, where $\Sigma_r = T \times Q$ and $\langle T_r, r \rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_1, q_1), (c_1, q_1), \ldots, (c_n, q_n)\} \subseteq \{-1, 0, \ldots, k\} \times Q$, such that the following hold:

- S satisfies θ, and
- for all $1 \leq i \leq n$, we have $y \cdot i \in T_r$, $x \cdot c_i$ is defined, and $r(y \cdot i) = (x \cdot c_i, q_i)$.

Note that the automaton cannot go backwards from the root of the input tree, as we require that $x \cdot c_i$ be defined, but $\varepsilon \cdot -1$ is undefined.

A run $\langle T_r, r \rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. We consider here *parity* acceptance conditions [43]. A parity condition over a state set Q is a finite sequence $F = (G_1, G_2, \ldots, G_m)$ of subsets of Q, where $G_1 \subseteq G_2 \subseteq \ldots \subseteq G_m = Q$. Given a run $\langle T_r, r \rangle$ and an infinite path $P \subseteq T_r$, let $inf(P) \subseteq Q$ be such that $q \in inf(P)$ if and only if there are infinitely many $y \in P$ for which $r(y) \in T \times \{q\}$. That is, $inf(P)$ contains exactly all the states that appear infinitely often in P. A path P satisfies the condition F if there is an *even* i for which $inf(P) \cap G_i \neq \emptyset$ and $inf(P) \cap G_{i-1} = \emptyset$. (For *co-parity* acceptance condition we require i to be *odd*.) An automaton accepts a labeled tree if and only if there exists a run that accepts it. We denote by $\mathcal{L}(A)$ the set of all Σ-labeled trees that A accepts.

3 The Tree-Model Property

To determine the truth value of a Boolean formula it suffices to consider its subformulas. For modal formulas, one has to consider a bigger collection of formulas, the so called Fischer-Ladner closure [13]. The closure, $cl(\varphi)$, of a sentence φ is the smallest set of sentences that satisfies the following:

- $\varphi \in cl(\varphi)$.
- If $\varphi_1 \wedge \varphi_2 \in cl(\varphi)$ or $\varphi_1 \vee \varphi_2 \in cl(\varphi)$, then $\varphi_1 \in cl(\varphi)$ and $\varphi_2 \in cl(\varphi)$.
- If $\langle \alpha \rangle \psi \in cl(\varphi)$ or $[\alpha]\psi \in cl(\varphi)$, then $\psi \in cl(\varphi)$.
- If $\lambda y.\varphi(y) \in cl(\varphi)$, then $\varphi(\lambda y.\varphi(y)) \in cl(\varphi)$.

As proved in [22], for every sentence φ, the number of elements in $cl(\varphi)$ is linear in the length $\|\varphi\|$ of φ.

An *atom* $\mathbf{A}$ of φ is a set of formulas in $cl(\varphi)$ that satisfies the following properties:

- if $p \in \mathsf{AP}$, then, exclusively, either $p \in \mathbf{A}$ or $\neg p \in \mathbf{A}$,
- if $\varphi_1 \wedge \varphi_2 \in cl(\varphi)$, then $\varphi_1 \wedge \varphi_2 \in \mathbf{A}$ iff $\varphi_1 \in \mathbf{A}$ and $\varphi_2 \in \mathbf{A}$,
- if $\varphi_1 \vee \varphi_2 \in cl(\varphi)$, then $\varphi_1 \vee \varphi_2 \in \mathbf{A}$ iff $\varphi_1 \in \mathbf{A}$ or $\varphi_2 \in \mathbf{A}$,
- if $\lambda X.\psi(X) \in cl(\varphi)$, then $\lambda X.\psi(X) \in \mathbf{A}$ iff $\psi(\lambda X.\psi(X)) \in \mathbf{A}$.

Intuitively, an atom is a consistent subset of $cl(\varphi)$. The set of atoms of φ is denoted $at(\varphi)$. Clearly, the size of $at(\varphi)$ is at most exponential in the length of φ.

A *pre-model* (K, π) for φ is a pair consisting of a Kripke structure $K = \langle W, R, L \rangle$ and a labeling function $\pi : W \to at(\varphi)$ that satisfies the following properties:

- $\varphi \in \pi(u)$, for some $u \in W$,
- if $p \in \pi(u)$, then $u \in L(p)$, and if $\neg p \in \pi(u)$, then $u \notin L(p)$, for $p \in \mathsf{AP}$ and $u \in W$,
- if $\langle \alpha \rangle \psi \in \pi(u)$, for $u \in W$, then $\psi \in \pi(v)$, for some $v \in W$ such that $(u, v) \in R(\alpha)$.

– if $[\alpha]\psi \in \pi(u)$, for $u \in W$, then $\psi \in \pi(v)$, for all $v \in W$ such that $(u,v) \in R(\alpha)$.

A pre-model of φ is almost a model of φ except for fixpoint formulas that do not necessarily get the right semantics (that is, fixpoints are arbitrary rather than minimal or maximal as needed).

Fixpoint sentences "trigger" evaluation of other fixpoint formulas. For example, for the formula $\mu X.(p \vee \langle a \rangle X)$ to hold at some point u, the sentence $p \vee \langle a \rangle(\mu X.(p \vee \langle a \rangle X))$ also has to hold at u. The distinction between least and greatest fixpoint formulas is in the presence or absence of nonterminating evaluation sequences. The problem with these evaluation sequences is that they are hard to trace in the presence of disjunctions and existential modalities. Intuitively, we do not *a priori* know whether a disjunction $\varphi_1 \vee \varphi_2$ will be true because of φ_1 or because of φ_2 and we do not know which a-successor of a point u will make $\langle a \rangle \psi$ true at u. To overcome this difficulty, Streett and Emerson introduced the technical notion of *choice functions* [40].

A *choice function* ρ for a pre-model (K, π) of φ, where $K = \langle W, R, L \rangle$, is a partial function from $W \times cl(\varphi)$ to $W \cup cl(\varphi)$ such that for each $u \in W$: (a) for each disjunction $\varphi_1 \vee \varphi_2 \in \pi(u)$, we have that $\rho(u, \varphi_1 \vee \varphi_2)$ is either φ_1 or φ_2, and $\rho(u, \varphi_1 \vee \varphi_2) \in \pi(u)$, and (b) for each existential formula $\langle \alpha \rangle \psi \in \pi(u)$, we have that $\rho(u, \langle \alpha \rangle \psi)$ is some $v \in W$ such that $(u,v) \in R(\alpha)$ and $\psi \in \pi(v)$. Intuitively, a choice function identifies how a disjunctive formula or an existential formula is satisfied. An *adorned* pre-model (K, π, ρ) for φ consists of a pre-model (K, π) for φ and a choice function ρ.

We can now define formally the notion of *derivation* between occurrences of sentences in adorned pre-models. Let (K, π, ρ) be an adorned pre-model of φ, The derivation relation, denoted $\vdash$, is defined as follows:

– if $\varphi_1 \vee \varphi_2 \in \pi(u)$ then $(\varphi_1 \vee \varphi_2, u) \vdash (\rho(u, \varphi_1 \vee \varphi_2), u)$,
– if $\varphi_1 \wedge \varphi_2 \in \pi(u)$, then $(\varphi_1 \wedge \varphi_2, u) \vdash (\varphi_1, u)$ and $(\varphi_1 \wedge \varphi_2, u) \vdash (\varphi_2, u)$,
– if $\langle \alpha \rangle \psi \in \pi(u)$, then $(\langle \alpha \rangle \psi, u) \vdash (\psi, \rho(u, \langle \alpha \rangle \psi))$,
– if $\lambda X.\psi(X) \in \pi(u)$, then $(\lambda X.\psi(X), u) \vdash (\psi(\lambda X.\psi(X)), u)$.

A least-fixpoint sentence $\mu X.\psi(X)$ is said to be *regenerated* from point u to point v (u might be equal to v) in an adorned premodel (K, π, ρ) if there is a sequence $(\theta_1, u_1), \ldots, (\theta_k, u_k)$, with $k > 1$, such that $\theta_1 = \theta_k = \mu X.\psi(X)$, $u_1 = u$, $u_k = v$, $(\theta_l, u_l) \vdash (\theta_{l+1}, u_{l+1})$, for $0 < l < k$, and $\mu X.\psi(X)$ is a subsentence of each of the θ_i's. We say that (K, π, ρ) is *well-founded* if there is no fixpoint sentence $\mu X.\psi(X) \in cl(\varphi)$ and an infinite sequence $u_0, u_1, \ldots$ such that $\mu X.\psi(X)$ is regenerated from u_j to u_{j+1} for all $j \geq 0$.

The following theorem was shown for the standard μ-calculus, but the proof is insensitive to the direction of the modalities.

Theorem 1. [40] *A sentence φ of the full μ-calculus has a model K if and only if it has a well-founded adorned pre-model (K, π, ρ).*

We can now establish the tree-model property for the full μ-calculus. Note that it follows from [39] that the finite-model property does not hold for the full μ-calculus; in contrast, it does hold for the standard μ-calculus [23]. The tree-model property asserts that if a sentence is satisfiable then it is satisfiable by a bounded-degree infinite *tree structure*. A tree structure is a Kripe structure

$\langle W, R, L\rangle$ where W is a tree and for each program α if $(u, v) \in R(\alpha)$, then either v is a successor of u or u is a successor of v.

The standard way of proving the tree-model property is to take a model and straightforwardly "unravel" it (see [48]); this does not work when we have backward modalities and deterministic programs. We have to be careful to unravel in a way that simultaneously preserves determinism and satisfies backward modalities.

Theorem 2. *If a formula φ in the full μ-calculus is satisfiable, then it is satisfiable at the root of a tree structure whose branching degrees are bounded by $\|\varphi\|$.*

Proof Sketch: Let $\|\varphi\| = n$. Suppose that $K, w_0 \models \varphi$ for $K = \langle W, R, L\rangle$ and $w_0 \in W$. By Theorem 1, φ has a well-founded adorned pre-model (K, π, ρ). We will construct a well-founded adorned tree pre-model of φ, and then appeal again to Theorem 1. We define a partial mapping $\theta : \{1, \ldots, n\}^* \to W$ by induction. Let W' be the set of elements where θ is defined. The pre-model will be (K', π', ρ'), where (a) $K' = \langle W', R', L'\rangle$, (b) $L'(x) = L(\theta(x))$, (c) $\pi'(x) = \pi(\theta(x))$, (d) $\rho'(x, \varphi_1 \vee \varphi_2) = \rho(\theta(x), \varphi_1 \vee \varphi_2)$. We will define R' and the rest of ρ' inductively.

Let $\langle\alpha_1\rangle\psi_1, \ldots \langle\alpha_m\rangle\psi_m$ be an enumeration of the existential subformulas of φ. Note that $m < n$. We start by taking $\theta(\varepsilon) = w_0$. Suppose now that we have already considered every member of $\{1, \ldots, n\}^k$, and we have already considered $xi1, \ldots, xi(j-1)$, where $xi \in \{1, \ldots, n\}^k$ and $1 \leq j \leq n$. If $\langle\alpha_j\rangle\psi_j \in \pi(\theta(xi))$, then, since (K, π, ρ) is a pre-model, $\rho(\theta(xi), \langle\alpha_j\rangle\psi_j) = u$, where $u \in W$, $(\theta(xi), u) \in R(\alpha_j)$ and $\psi_j \in \pi(u)$. There are now four cases to consider: (a) if $\alpha_j = a$ and $a \in \mathsf{Prog} \setminus \mathsf{DProg}$, then define $\theta(xij) = u$, add (xi, xij) to $R'(a)$, and define $\rho'(xi, \langle a\rangle\psi_j) = xij$, (b) if $\alpha_j = a^-$, then define $\theta(xij) = u$, add (xij, xi) to $R'(a)$, and define $\rho'(xi, \langle a^-\rangle\psi_j) = xij$, (c) if $\alpha_j = a$, $a \in \mathsf{DProg}$, and $(xi, z) \notin R'(a)$ for all $z \in \{x, xi1, \ldots, xi(j-1)\}$, then define $\theta(xij) = u$, add (xi, xij) to $R'(a)$, and define $\rho'(xi, \langle a\rangle\psi_j) = xij$, and (d) if $\alpha_j = a$, $a \in \mathsf{DProg}$, and $(xi, z) \in R'(a)$ for some $z \in \{x, xi1, \ldots, xi(j-1)\}$, then $\theta(xij)$ stays undefined, and $\rho'(xi, \langle a\rangle\psi_j) = z$. It can now be shown that (K', π', ρ') is a well-founded adorned tree pre-model of φ. □

Note that the tree model constructed in the above proof is not a *full* tree; it is possible for xi to be a node in the tree without $x(i-1)$ being a node in the tree. It is technically more convenient to deal with full trees. To that end we add a new atomic proposition p_T. The intuition is that p_T is true only at nodes that belong to the tree (so nodes where p_T is false are dummy nodes). We now replace each modal subformula $\langle\alpha\rangle\psi$ by $\langle\alpha\rangle(p_T \wedge \psi)$ and each modal subformula $[\alpha]\psi$ by $[\alpha](p_T \to \psi)$. This transformation causes only a linear blow-up, It is easy to see that the if the original formula is satisfiable at the root of a tree structure whose branching degrees are bounded by $\|\varphi\|$, then the new formula φ' is satisfiable at the root of a $\|\varphi'\|$-ary tree (where p_T is true at the root). We call such formulas *uniform* formulas, and we can thus restrict attention to full trees. We can assume further that $\mathsf{DProg} = \{a_1, \ldots, a_k\}$ and the a_j-successor of a node x is the node xj. Since we need to represent the information about the transition function R, we introduce new atomic propositions to represent this information. For each atomic program a we introduce two atomic propositions: p_a holds at a node xj when $(x, xj) \in R(a)$ and p_a^- holds in xj when $(xj, x) \in R(a)$. Note that for a

deterministic program a_j, the atomic proposition p_{a_j} holds in nodes of the form xj, but not in the form xi for $i \neq j$. Also, if $p_{a_j}^-$ holds at x, then p_T cannot hold in xj, because x cannot have two a_j-successors. We call tree structures that obey these constraints *well-behaved* tree structures,

We are now ready to describe the translation from formulas to two-way alternating tree-automata.

Theorem 3. *Given a uniform formula φ of the full μ-calculus, we can construct a two-way alternating parity automaton A_φ whose number of states is $O(\|\varphi\|)$, such that $\mathcal{L}(A_\varphi)$ is exactly the set of well-behaved $\|\varphi\|$-ary tree structures satisfying φ at the root.*

Proof Sketch: The automaton is obtained by taking the intersection of two automata (intersection is trivial for alternating automata [29]). The first automaton checks that the input tree is well-behaved. Constructing this automaton is an easy exercise. The second automaton checks that φ is satisfied at the root of the input tree. Take $A_\varphi = \langle 2^{\mathsf{AP}}, cl(\varphi), \delta, \varphi, F\rangle$. We need to define the transition function δ and the acceptance condition F. Let $n = \|\varphi\|$. For all $\sigma \in 2^{\mathsf{AP}}$ and $a \in \mathsf{Prog}$ we define:

- $\delta(p,\sigma) = \mathbf{true}$ if $p \in \sigma$. • $\delta(p,\sigma) = \mathbf{false}$ if $p \notin \sigma$.
- $\delta(\neg p,\sigma) = \mathbf{true}$ if $p \notin \sigma$. • $\delta(\neg p,\sigma) = \mathbf{false}$ if $p \in \sigma$.
- $\delta(\varphi_1 \wedge \varphi_2,\sigma) = (0,\varphi_1) \wedge (0,\varphi 2)$.
- $\delta(\varphi_1 \vee \varphi_2,\sigma) = (0,\varphi_1) \vee (0,\varphi 2)$.
- $\delta(\lambda y.\varphi(y),\sigma) = (0,\varphi(\lambda y.\varphi(y)))$.
- $\delta(\langle a\rangle\psi,\sigma) = ((-1,\psi) \wedge (0,p_a^-)) \vee \bigvee_{c=1}^{n} ((c,\psi) \wedge (c,p_a))$.
- $\delta(\langle a^-\rangle\psi,\sigma) = ((-1,\psi) \wedge (0,p_a)) \vee \bigvee_{c=1}^{n} ((c,\psi) \wedge (c,p_a^-))$.
- $\delta([a]\psi,\sigma) = ((-1,\psi) \vee (0,\neg p_a^-)) \wedge \bigwedge_{c=1}^{n} ((c,\psi) \vee (c,\neg p_a))$.
- $\delta([a^-]\psi,\sigma) = ((-1,\psi) \vee (0,\neg p_a)) \wedge \bigwedge_{c=1}^{n} ((c,\psi) \vee (c,\neg p_a^-))$.

The translation here is by a straighfoward induction on the structure of φ. It simplifies the translation given in [2] (for the standard μ-calculus), since we allow the usage of ε-transitions (i.e., transitions to the direction 0).

It remains to define the parity acceptance condition F. This is done analogously to the construction in [11], which drew a tight relationship between model checking for the standard μ-calculus and one-way nondeterministic parity tree automata. □

Corollary 4. *A uniform formula φ of the full μ-calculus is satisfiable iff $\mathcal{L}(A_\varphi)$ is not empty.*

4 Emptiness of Two-Way Alternating Tree Automata

We solve the emptiness problem for two-way alternating tree automata, by reducing them to one-way nondeterministic tree automata. A run of a nondeterministic tree automaton is a tree with the same structure as the input tree but a different label set; the run tree is labeled by states. In contrast, a run of an alternating automaton is a tree whose structure can be quite different than that of the input tree. Thus, to reduce alternating automata to nondeterministic automata, we have to overcome this difficulty.

Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a two-way alternating automaton on k-ary trees. A *strategy tree* for A is a mapping $\tau : \{1, \ldots, k\}^* \to 2^{Q \times [k] \times Q}$. Thus, each label in a strategy is an edge-$[k]$-labeled directed graph on Q. Intuitively, each label is a set of transitions. For each label ζ, we define $state(\zeta) = \{u : (u, i, v) \in \zeta\}$, i.e., $state(\zeta)$ is the set of sources in the graph ζ. The strategy tree τ is *on* a k-ary input tree $(\{1, \ldots, k\}^*, V)$ if $q_0 \in state(\tau(\varepsilon))$, and for each node $x \in \{1, \ldots, k\}^*$ and each state $q \in state(\tau(x))$, the set $\{(c, q') : (q, c, q') \in \tau(x)\}$ satisfies $\delta(q, V(x))$. Thus, each label can be viewed as a strategy of satisfying the transition function.

A *path* β in a strategy tree τ is a sequence $(u_1, q_1), (u_2, q_2), \ldots$ of pairs from $\{1, \ldots, k\}^* \times Q$ such that, for all $i > 0$, there is some $c_i \in [k]$ such that $(q_i, c_i, q_{i+1}) \in \tau(u_i)$ and $u_{i+1} = u_i \cdot c_i$. Thus, β is obtained by following transitions in the strategy tree. We define $inf(\beta)$ to be the set of states in Q that occur infinitely often in β. We say that an infinite path β satisfies a parity condition $F = (G_1, G_2, \ldots)$ if there is an even i for which $inf(\beta) \cap G_i \neq \emptyset$ and $inf(\beta) \cap G_{i-1} = \emptyset$. We say that τ is *accepting* if all infinite paths in τ satisfy F.

Proposition 5. *A two-way alternating parity automaton accepts an input tree iff it has an accepting strategy tree over the input tree.*

Proof Sketch: The "if" direction is immediate. For the "only if" direction, let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$, $F = (G_1, G_2, \ldots)$, and let $(\{1, \ldots, k\}^*, V)$ be the input tree. Consider the following game between two players: the Protagonist and the Antagonist. Intuitively, the Protoganist is trying to show that A accepts the input tree, and the Antagonist is trying to challenge that. A *configuration* of the game is a pair in $\{1, \ldots, k\}^* \times Q$. The initial configuration is (ε, q_0). Consider a configuration (x, q). The Protagonist now chooses a set $\{(c_1, q_1), \ldots, (c_m, q_m)\}$ that satisfies $\delta(q, V(x))$; the Antagonist responds by choosing an element (c_i, q_i) of the set. The new configuration is then $(x \cdot c_i, q_i)$. If $x \cdot c_i$ is undefined or if $\delta(q, V(x)) = \mathbf{false}$, then the Antagonist wins immediately. Consider now an infinite play γ. Let $inf(\gamma)$ be the set of states in Q that repeat infinitely in the sequence of configurations in γ. The Protagonist wins if there is an even i for which $inf(\gamma) \cap G_i \neq \emptyset$ and $inf(\gamma) \cap G_{i-1} = \emptyset$. It is not difficult to see that the Protagonists *wins the game*, i.e., has a winning strategy against the Antagonist, iff A accepts the input tree.

The game as we described it meets the conditions in [20] (see also [43]). It follows that if the Protagonist wins then it as a *memoryless* strategy, i.e., a strategy whose moves do not depend on the history of the game, but only on the current configuration. Thus, A accepts the input tree iff it has a strategy tree over the input tree. □

We have thus succeeded in defining a notion of run for alternating automata that will have the same tree structure as the input tree. We are still facing the problem that paths in a strategy tree can go both up and down. We need to find a way to restrict attention to uni-directional paths.

Let $A = \langle \Sigma, Q, \delta, q_0, F \rangle$, $F = (G_1, G_2, \ldots, G_m)$, be a two-way alternating automaton on k-ary trees, and let $\tau : \{1, \ldots, k\}^* \to 2^{Q \times [k] \times Q}$ be a strategy tree for A. An *annotation* for A is a mapping $\eta : \{1, \ldots, k\}^* \to 2^{Q \times 2^{\{1, \ldots, m\}} \times Q}$. Thus, each label in an annotation is an edge-$2^{\{1, \ldots, m\}}$-labeled directed graph on Q. For each state $q \in Q$, let $index(q)$ be the minimal i such that $q \in G_i$. We say that η is an annotation *of* τ if some closure conditions hold for each node

$x \in \{1,\ldots,k\}^*$. Intuitively, these conditions say that η contains all relevant information about finite paths in τ. The conditions are: (a) if $(q,H_1,q') \in \eta(x)$ and $(q',H_2,q'') \in \eta(x)$, then $(q,H_1 \cup H_2,q'') \in \eta(x)$, (b) if $(q,0,q') \in \tau(x)$ then $(q,index(q'),q') \in \eta(x)$, (c) if $x = yi$, $(q,-1,q') \in \tau(x)$, $(q',H,q'') \in \eta(y)$, and $(q'',i,q''') \in \tau(x)$, then $(q,H \cup \{index(q'),index(q''')\},q''') \in \eta(x)$, (d) if $y = xi$, $(q,i,q') \in \tau(x)$, $(q',H,q'') \in \eta(y)$, and $(q'',-1,q''') \in \tau(y)$, then $(q,H \cup \{index(q'),index(q''')\},q''') \in \eta(x)$.

A *downward path* κ in η is a sequence $(u_1,q_1,t_1),(u_2,q_2,t_2),\ldots$ of triples, where each u_i is in $\{1,\ldots,k\}^*$, each q_i is in Q, and each t_i is either an element of $\tau(u_i)$ or $\eta(u_i)$, such that: (a) either t_i is (q_i,c,q_{i+1}) for some $c \in \{1,\ldots,k\}$, and $u_{i+1} = u_i \cdot c$; in this case we define $index(t_i)$ to be $index(q_{i+1})$, or (b) t_i is (q_i,H,q_{i+1}), for $H \subseteq \{1,\ldots,m\}$, and $u_{i+1} = u_i$; in this case we define $index(t_i)$ to be $\min(H)$. We consider two kinds of downward paths: (a) infinite paths $\kappa = (u_1,q_1,t_1),(u_2,q_2,t_2),\ldots$, whose index, $index(\kappa)$ is defined as the minimal j such that $index(t_i) = j$ infinitely often, or finite paths $\kappa = (u_1,q_1,t_1),\ldots,(u_s,q_s,t_s)$, where $t_s = (q_s,H_s,q_s)$ (i.e., the path ends in a loop), whose index, $index(\kappa)$ is defined as $index(t_s)$. In either case we say that κ *violates* F if $index(\kappa)$ is odd. We say that η is *accepting* if no downward path in η violates F.

Proposition 6. *A two-way alternating parity automaton accepts an input tree iff it has a strategy tree over the input tree and an accepting annotation of the strategy tree.*

Proof Sketch: Let $A = \langle \Sigma,Q,\delta,q_0,F\rangle$, $F = (G_1,G_2,\ldots,G_m)$, and let $(\{1,\ldots,k\}^*,V)$ be the input tree. Suppose first that A accepts the input tree. By Proposition 5, there is an accepting strategy tree τ over $(\{1,\ldots,k\}^*,V)$. Consider two annotations η_1 and η_2 of τ. Their *intersection* $\eta_1 \cap \eta_2$, defined by $\eta_1 \cap \eta_2(x) = \eta_1(x) \cap \eta_2(x)$ for each $x \in \{1,\ldots,k\}^*$, is also an annotation of τ. Thus, there exists a *minimal* annotation η of τ. (That is, if η' is also an annotation of τ, then η must be *contained* in η', i.e., for each $x \in \{1,\ldots,k\}^*$ we have that $\eta(x) \subseteq \eta'(x)$.) It can be shown that since all paths in τ satisfies F, no downward path in η violates F. Conversely, suppose that τ is a strategy tree over $(\{1,\ldots,k\}^*,V)$ and η is an annotation of τ such that no downward path in η violates F. Let η' be the minimal annotation of τ; η' must be contained in η. It follows that no downward path in η' violates F. From this we can show that all paths in τ satisfy F. By Proposition 5, A accepts $(\{1,\ldots,k\}^*,V)$. □

Consider now *annotated trees* $(\{1,\ldots,k\}^*,V,\tau,\eta)$, where τ is a strategy tree for A on $(\{1,\ldots,k\}^*,V)$ and η is an annotation of τ. We say that $(\{1,\ldots,k\}^*,V,\tau,\eta)$ is *accepting* if η is accepting.

Theorem 7. *Let A be a two-way alternating parity tree automaton. Then there is a nondeterministic parity tree automaton A^n such that $\mathcal{L}(A) = \mathcal{L}(A^n)$. The number of states in A^n is exponential in the number of states of A, but the size of the acceptance condition of A^n is linear in the size of the acceptance condition of A.*

Proof Sketch: Let $A = \langle \Sigma,Q,\delta,q_0,F\rangle$, $F = (G_1,\ldots,G_m)$. The nondeterministic automaton A^n is the intersection of two automata. Given an annotated tree $(\{1,\ldots,k\}^*,V,\tau,\eta)$, the first automaton, A_1^n, checks that τ is a strategy tree on $(\{1,\ldots,k\}^*,V)$ and that η is an annotation of τ. Constructing A_1^n is a

not-too-difficult exercise. The second automaton, A_2^n, checks that τ is accepting. We construct A_2^n in several steps.

Consider a downward path $\kappa = (u_1, q_1, t_1), (u_2, q_2, t_2), \ldots$. We define its *projection* to be the sequence $proj(\kappa) = (q_1, t_1), (q_2, t_2), \ldots$, which is is a sequence over the finite alphabet $Q \times \big((Q \times [k] \times Q) \cup (Q \times 2^{\{1,\ldots,m\}} \times Q)\big)$. We can construct a co-parity word automaton B that accepts projections of downward paths that violates F. The state set of B is $Q \times \{1, \ldots, m\}$. All that B does is check that (a) either t_i is (q_i, c, q_{i+1}) for some $c \in \{1, \ldots, k\}$, (b) t_i is (q_i, H, q_{i+1}) for some $H \subseteq \{1, \ldots, m\}$. In either case B also remembers $index(t_i)$. B accepts if there is some $t_i = (q_i, H_i, q_i)$ with an odd $index(t_i)$ or if the minimal index that repeats infinitely often is odd. The size of the acceptance condition of B is linear in the size of the acceptance condition of A. From B we construct another co-parity word automaton B'. This automaton reads a sequence of labels of τ or η and checks whether it contains a projection of a downward path that violates F. The state set of B' is still $Q \times \{1, \ldots, m\}$, though its alphabet is $2^{Q \times [k] \times Q} \cup 2^{Q \times 2^{\{1,\ldots,m\}} \times Q}$. The size of the acceptance condition of B' is still linear in the size of the acceptance condition of A. We now co-determinize B', i.e., determinize it and complement it in a singly-exponential construction [36,37,43] to obtain a deterministic parity word automaton B'' that rejects violating downward paths. The deterministic tree automaton B''' is obtained by simply running B'' in parallel over all branches of $(\{1, \ldots, k\}^*, V, \tau, \eta)$. Thus, its size is exponential in the size of A. The size of the acceptance condition of B''' is still linear in the size of the acceptance condition of A. Finally, A_2^n is obtained from B''' by projecting out the τ and η component of the inputs tree, so the input tree is of the form $(\{1, \ldots, k\}^*, V)$. □

We can now combine Theorem 3, Corollary 4, Theorem 7, and the tree automata emptiness algorithms in [9,26,34] (which are polynomial in the number of states, but exponential in the size of the acceptance condition) to obtain:

Theorem 8. *The satisfiability problem for the full μ-calculus is decidable in exponential time.*

5 Concluding Remarks

Over the last few years there has been a significant interest in logics with bounded number of variables (see [6,21]). One surprising result in this line of research is that while FO^3, 3-variable first-order logic, is already undecidable, the satisfiability problem for FO^2, 2-variable first-order logic, is NEXPTIME-complete [17]. This lead researchers to consider extensions of FO^2. Unfortunately, fairly modest extensions of FO^2 by fixpoint constructs quickly lead to undecidability [18]. On the other hand, the full μ-calculus can be viewed as a fragment of 2-variable fixpoint logic [48]. Our main decidability result here for the full μ-calculus pushes the "decidability envelope" further. The key difference between the full μ-calculus and the fixpoint extensions of FO^2 seems to be the tree-model property, which was offered in [48] as an explanation for the robust decidability of propositional program logics.

References

1. I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Confer-*

ence on Computer Aided Verification, volume 818 of *Lecture Notes in Computer Science*, pages 182–193, Stanford, June 1994.

2. O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Computer Aided Verification, Proc. 6th Int. Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, June 1994. Springer-Verlag, Berlin.
3. G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proc. 1996 Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1996.
4. 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.
5. R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.
6. A. Dawar, S. Lindell, and S. Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 119:160–175, 1995.
7. E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th Int'l Colloq. on Automata, Languages and Programming*, pages 169–181, 1980.
8. E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.
9. E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 368–377, White Plains, October 1988.
10. E.A. Emerson and C. Jutla. Tree automata, Mu-calculus and determinacy. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.
11. E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Conference*, volume 697, pages 385–396, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.
12. E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.
13. M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.
14. G. De Giacomo and M. Lenzerini. Concept languages with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 411–415. John Wiley and Sons, 1994.
15. G. De Giacomo and M. Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. In *Proc. 4th European Workshop on Logics in Artificial Intelligence (JELIA-94)*, number 838 in Lecture Notes In Artificial Intelligence, pages 332–346. Springer-Verlag, 1994.
16. G. De Giacomo and F. Masacci. Tableaux and algorithms for propositional dynamic logic with converse. In M. A. McRobbie and J.K. Slaney, editors, *Proc. 13th Int'l Conf. on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 613–627. Springer-Verlag, 1996.
17. E. Grädel, Ph. G. Kolaitis, and M. Y. Vardi. The decision problem for 2-variable first-order logic. *Bulletin of Symbolic Logic*, 3:53–69, 1997.
18. E. Grädel, M. Otto, and E. Rosen. Undecidability results for two-variable logics. Unpublished manuscript, 1996.
19. T.A. Henzinger, O. Kupferman, and S. Qadeer. From pre-historic to post-modern symbolic model checking. In *Computer Aided Verification, Proc. 10th Int. Conference*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
20. C.S. Jutla. Determinization and memoryless winning strategies. *Information and Computation*, 133(2):117–134, 1997.

21. Ph.G. Kolaitis and M.Y. Vardi. On the expressive power of variable-confined logics. In *Proc. 11th IEEE Symp. on Logic in Computer Science*, pages 348–359, 1996.
22. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
23. D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47(3):333–354, 1988.
24. D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 313–325. Springer-Verlag, 1984.
25. O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th IEEE Symposium on Logic in Computer Science*, pages 25–35, San Diego, June 1995.
26. O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symposium on Theory of Computing*, Dallas, 1998.
27. O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.
28. D.E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science*, pages 422–427, Edinburgh, July 1988.
29. D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54,:267–276, 1987.
30. D. Park. Finiteness is μ-ineffable. *Theoretical Computer Science*, 3:173–181, 1976.
31. S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, pages 28–37, Vancouver, August 1984.
32. A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Proc. Advanced School on Current Trends in Concurrency*, pages 510–584, Berlin, 1985. Volume 224, LNCS, Springer-Verlag.
33. A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 123–144. Springer-Verlag, 1985.
34. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages*, Austin, January 1989.
35. V.R. Pratt. A decidable μ-calculus: preliminary report. In *Proc. 22nd IEEE Symposium on Foundation of Computer Science*, pages 421–427, 1981.
36. S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.
37. S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symposium on Theory of Computing*, Victoria, May 1992.
38. G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.
39. R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.
40. R.S. Streett and E.A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, 1989.
41. S.S. Streett. *A propositional dynamic logic for reasoning about program divergence.* PhD thesis, M.Sc. Thesis, MIT, 1980.
42. W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.
43. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.
44. M.Y. Vardi. The taming of converse: Reasoning about two-way computations. In *Logic of Programs Workshop*, volume 193, pages 413–424, Brooklyn, June 1985. Lecture Notes in Computer Science, Springer-Verlag.

45. M.Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. on Principles of Programming Languages*, pages 250–259, San Diego, January 1988.
46. M.Y. Vardi. On the complexity of bounded-variable queries. In *Proc. ACM 14th Symposium on Principles of Database Systems*, June 1995.
47. M.Y. Vardi. Alternating automata – unifying truth and validity checking for temporal logics. In W. McCune, editor, *Proc. 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 191–206. Springer-Verlag, Berlin, july 1997.
48. M.Y. Vardi. What makes modal logic so robustly decidable? In *Descriptive Complexity and Finite Models*, pages 149–183. American Mathematical Society, 1997.
49. M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.
50. M.Y. Vardi and P. Wolper. Yet another process logic. In *Logics of Programs*, volume 164, pages 501–512. Lecture Notes in Computer Science, Springer-Verlag, 1984.
51. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986.
52. G. Winskel. Note on model checking the modal ν-calculus. *Theoretical Computer Science*, 83:157–167, 1991.