An Automata-Theoretic Approach to Branching-Time Model Checking

ORNA KUPFERMAN

Hebrew University, Jerusalem, Israel

MOSHE Y. VARDI

Rice University, Houston, Texas

AND

PIERRE WOLPER

Université de Liège, Liège Sart-Tilman, Belgium

Abstract. Translating linear temporal logic formulas to automata has proven to be an effective approach for implementing linear-time model-checking, and for obtaining many extensions and improvements to this verification method. On the other hand, for branching temporal logic, automata-theoretic techniques have long been thought to introduce an exponential penalty, making them essentially useless for model-checking. Recently, Bernholtz and Grumberg [1993] have shown that this exponential penalty can be avoided, though they did not match the linear complexity of non-automata-theoretic algorithms. In this paper, we show that *alternating tree automata* are the key to a comprehensive automata-theoretic framework for branching temporal logics. Not only can they be used to obtain optimal decision procedures, as was shown by Muller et al., but, as we show here, they also make it possible to derive optimal model-checking algorithms. Moreover, the simple combinatorial structure that emerges from the automata-theoretic approach opens up new possibilities for the implementation of branching-time model checking, and has enabled us to derive improved space complexity bounds for this long-standing problem.

A preliminary version of this work was presented at the 1994 Conference on Computer-Aided Verification.

M. Y. Vardi was supported in part by National Science Foundation (NSF) grants CCR 96-28400 and CCR 97-00061 and by a grant from the Intel Corporation.

P. Wolper was supported in part by ESPRIT LTR action REACT.

Authors' addresses: O. Kupferman, Institute of Computer Science, Hebrew University, Jerusalem 91904, Israel, e-mail: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~orna; M. Y. Vardi, Department of Computer Science, Rice University, Houston, TX 77251-1892, e-mail: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi; P. Wolper, Université de Liège, Institut Montefiore, B28; B-4000 Liège Sart-Tilman, Belgium, e-mail: pw@montefiore.ulg.ac.be; URL: http://www.montefiore.ulg.ac.be/~pw.

Journal of the ACM, Vol. 47, No. 2, March 2000, pp. 312–360.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata (e.g., finite, push-down, resource-bounded)*; F.4.1. [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*temporal logic*

General Terms: Algorithms. Verification.

1. *Introduction*

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent programs [1981]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [Clarke et al. 1986; Lichtenstein and Pneuli 1985; Queille and Sifakis 1981]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state programs, as well as from the great ease of use of fully algorithmic methods. Finite-state programs can be modeled by transition systems where each state has a bounded description, and hence can be characterized by a fixed number of Boolean atomic propositions. This means that a finite-state program can be viewed as a finite *propositional Kripke structure* and that its properties can be specified using *propositional* temporal logic. Thus, to verify the correctness of the program with respect to a desired behavior, one only has to check that the program, modeled as a finite Kripke structure, satisfies (is a model of) the propositional temporal logic formula that specifies that behavior. Hence, the name *model checking* for the verification methods derived from this viewpoint. Surveys can be found in Clarke et al. [1993] and Wolper [1989].

There are two types of temporal logics: linear and branching [Lamport 1980]. In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. For linear temporal logics, a close and fruitful connection with the theory of automata over infinite words has been developed [Vardi and Wolper 1986a; 1994]. The basic idea is to associate with each linear temporal logic formula a finite automaton over infinite words that accepts exactly all the computations that satisfy the formula. This enables the reduction of linear temporal logic problems, such as satisfiability and model-checking, to known automata-theoretic problems, yielding clean and asymptotically optimal algorithms. Furthermore, these reductions are very helpful for implementing temporal-logic based verification methods, and are naturally combined with techniques such as *on-the-fly* verification [Vardi and Wolper 1986a; Jard and Jeron 1989; Courcoubetis et al. 1992] that help coping with the "state-explosion" problem.

For branching temporal logics, the automata-theoretic counterpart are automata over infinite trees [Rabin 1969; Vardi and Wolper 1986b]. By reducing satisfiability to the nonemptiness problem for these automata, optimal decision procedures have been obtained for various branching temporal logics.[1] Unfortunately, the automata-theoretic approach does not seem to be applicable to branching-time model checking. Indeed, model checking can be done in linear

[1] See, for example, Emerson [1985], Emerson and Jutla [1988], Emerson and Sistla [1984], Street and Emerson [1984], and Vardi and Wolper [1986b].

running time for CTL [Clarke et al. 1986; Queille and Sifakis 1981] and for the alternation-free fragment of the μ-calculus [Cleaveland 1993], and is in NP $\cap$ co-NP for the general μ-calculus [Clarke et al. 1986; Queille and Sifakis 1981] whereas there is an exponential blow-up involved in going from formulas in these logics to automata. Similarly, while model checking for the full branching time logic CTL* is PSPACE-complete, going from CTL* formulas to automata involves a doubly exponential blow up [Emerson and Sistla 1984]. Thus, using the construction of a tree automaton as a step in a model-checking algorithm seems a nonstarter, which can only yield algorithms that are far from optimal. (Indeed, the proof in Emerson et al. [1993] avoids the construction of tree automata that correspond to μ-calculus formulas.)

A different automata-theoretic approach to branching-time model checking, based on the concepts of *amorphous automata* and *simultaneous trees*, was suggested by Bernholtz and Grumberg [1993]. Amorphous automata have a flexible transition relation that can adapt to trees with varying branching degree. Simultaneous trees are trees in which each subtree is duplicated twice as the two leftmost successors of its root. Simultaneous trees thus enable the automaton to visit different nodes of the same path simultaneously. Bernholtz and Grumberg showed that CTL model checking is *linearly* reducible to the acceptance of a simultaneous tree by an amorphous automaton and that the latter problem can be solved in quadratic running time.

While this constitutes a meaningful first step towards applying automata-theoretic techniques to branching-time model checking, it is not quite satisfactory. First, unlike the situation with linear temporal logic, different automata are required to solve model checking and satisfiability and thus, we do not get a uniform automata-theoretic treatment for the two problems. Second, and more crucial, the complexity of the resulting algorithm is quadratic in both the size of the specification and the size of the program, which makes this algorithm impractical; after all, most of the current research in this area is attempting to develop methods to cope with *linear* complexity.

In this paper, we argue that *alternating tree automata* are the key to a comprehensive and satisfactory automata-theoretic framework for branching temporal logics. Alternating tree automata generalize the standard notion of nondeterministic tree automata by allowing several 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 [Muller et al. 1988; Emerson and Jutla 1991]. In fact, Emerson [1996] stated that "μ-calculus formulas are simply alternating tree automata." Muller et al. [1988] showed that this explains the exponential decidability of satisfiability for various branching temporal logics. We show here that this also explains the efficiency of model checking for those logics. The crucial observation is that for model checking, one does not need to solve the nonemptiness problem of tree automata, but rather the *1-letter* nonemptiness problem of *word* automata. This problem (testing the nonemptiness of an alternating word automaton that is defined over a singleton alphabet) is substantially simpler. Thus, alternating automata provide a unifying and optimal framework for both the satisfiability and model-checking problems for branching temporal logic.

We first show how our automata-theoretic approach unifies previously known results about model checking for branching temporal logics. The alternating automata used by Muller et al. [1988] are of a restricted type called *weak alternating automata*. To obtain an exponential decision procedure for the satisfiability of CTL and related branching temporal logics, Muller et al. [1986] used the fact that the nonemptiness problem for these automata is in exponential time. (In fact, as we show here, it is EXPTIME-complete.) We prove that the 1-letter nonemptiness of weak alternating word automata is decidable in linear running time, which yields an automata-based model checking algorithm of linear running time for CTL. We present a linear translation from alternation-free μ-calculus formulas to weak alternating automata. This implies, using the same technique, that model checking for this logic can be done in linear running time. For the general μ-calculus, it follows from the results in Emerson and Jutla [1988] that μ-calculus formulas can be linearly translated to alternating *Rabin* automata. We prove here that the 1-letter nonemptiness of alternating Rabin word automata is in NP, which entails that model checking of μ-calculus formulas is in NP $\cap$ co-NP.

As the algorithms obtained by our approach match known complexity bounds for branching temporal logics [Clarke et al. 1986; Cleaveland 1993; Emerson et al. 1993], what are the advantages offered by our approach? The first advantage is that it immediately broadens the scope of efficient model checking to other, and more expressive, branching temporal logics. For example, the dynamic logic considered in Muller et al. [1988] allows, in the spirit of Wolper [1983], nondeterministic tree automata as operators. Since this logic has a linear translation to weak alternating automata, it follows directly from our results that it also has a linear model-checking algorithm.

The second advantage comes from the fact that our approach combines the Kripke structure and the formula into a single automaton before checking this automaton for nonemptiness. This facilitates the use of a number of implementation heuristics. For instance, the automaton combining the Kripke structure and the formula can be computed *on-the-fly* and limited to its reachable states. This avoids exploring the parts of the Kripke structure that are irrelevant for the formula to be checked, and hence addresses the issue raised in the work on *local* model checking [Stirling and Walker 1991; Vergauwen and Lewis 1993], while preserving optimal complexity and ease of implementation. The above advantage of our approach is reflected in the performance of its implementation, as described in Visser [1998] and Visser and Barringer [1999].

The third advantage of the automata-theoretic approach is that it offers new and significant insights into the *space complexity* of branching-time model checking. It comes from the observation that the alternating automata that are obtained from CTL formulas have a special structure: they have *limited alternation*. We define a new type of alternating automata, *hesitant alternating automata*, that have this special structure. A careful analysis of the 1-letter nonemptiness problem for hesitant alternating word automata yields a top-down model-checking algorithm for CTL that uses space that is linear in the length of the formula and only poly-logarithmic in the size of the Kripke structure. We also present a translation of CTL* formulas to hesitant alternating automata and hence obtain a space-efficient model-checking algorithm for this logic too. This is very significant since it implies that for concurrent programs, model checking for

CTL and CTL* can be done in space polynomial in the size of the program description, rather than requiring space of the order of the exponentially larger expansion of the program, as is the case with standard bottom-up model-checking algorithms. On the other hand, the weak alternating automata that are obtained for alternation-free μ-calculus formulas do not have this special structure. Accordingly, we show that for this logic, as well as for the general μ-calculus, model checking for concurrent programs is EXPTIME-complete.

As discussed in Section 7, the automata-theoretic approach to branching-time model checking described here has contributed to several other results concerning the specification and verification of reactive systems. These results include contributions to the verification and synthesis of open systems, timed systems, and distributed systems (possibly with fairness constraints), as well as techniques for incorporating modularity and partial-order methods into branching-time verification.

2. *Preliminaries*

2.1. Temporal Logics and m-Calculi

2.1.1. *The Temporal Logics CTL* and CTL.* The logic CTL* combines both branching-time and linear-time operators [Emerson and Halpern 1986]. A path quantifier, either A ("for all paths") or E ("for some path"), can prefix an assertion composed of an arbitrary combination of the linear-time operators X ("next time"), and U ("until"). A *positive normal form* CTL* formula is a CTL* formula in which negations are applied only to atomic propositions. It can be obtained by pushing negations inward as far as possible, using De Morgan's laws and dualities of quantifiers and temporal connectives. For technical convenience, we use the linear-time operator $\tilde{U}$ as a dual of the U operator, and write all CTL* formulas in a positive normal form. There are two types of formulas in CTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. Formally, let AP be a set of atomic proposition names. A CTL* state formula is either:

—**true, false**, p, or $\neg p$, for all $p \in AP$;

—$\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL* state formulas;

—$A\psi$ or $E\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either:

—A CTL* state formula;

—$\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, or $\psi_1 \tilde{U} \psi_2$, where ψ_1 and ψ_2 are CTL* path formulas.

CTL* is the set of state formulas generated by the above rules.

The logic CTL is a restricted subset of CTL* in which the temporal operators must be immediately preceded by a path quantifier. Formally, it is the subset of CTL* obtained by restricting the path formulas to be $X\varphi_1$, $\varphi_1 U \varphi_2$, or $\varphi_1 \tilde{U} \varphi_2$, where φ_1 and φ_2 are CTL state formulas.

We use the following abbreviations in writing formulas:

—$F\psi$ = $\mathbf{true} U \psi$ ("eventually").

—$G\psi$ = $\mathbf{false} \tilde{U} \psi$ ("always").

We say that a CTL formula φ is an *U-formula* if it is of the form $A\varphi_1 U\varphi_2$ or $E\varphi_1 U\varphi_2$. The subformula φ_2 is then called the *eventuality* of φ. Similarly, φ is a *$\tilde{U}$-formula* if it is of the form $A\varphi_1\tilde{U}\varphi_2$ or $E\varphi_1\tilde{U}\varphi_2$. The *closure* $cl(\varphi)$ of a CTL* (CTL) formula φ is the set of all CTL* (CTL) state subformulas of φ (including φ, but excluding **true** and **false**). We define the *size* $\|\varphi\|$ of φ as the number of elements in $cl(\varphi)$. Note that, even though the number of elements in the closure of a formula can be logarithmic in the length of the formula if there are multiple occurrences of identical subformulas, our definition of size is legitimate since it corresponds to the number of nodes in a reduced DAG representation of the formula.

The semantics of CTL* is defined with respect to a *Kripke structure* $K = \langle AP, W, R, w^0, L\rangle$, where AP is a set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w'\rangle \in R$), w^0 is an initial state, and L: $W \rightarrow 2^{AP}$ maps each state to the set of atomic propositions true in that state. A *path* in K is an infinite sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, $\langle w_i, w_{i+1}\rangle \in R$. We denote the suffix $w_i, w_{i+1}, \ldots$ of π by π^i. We define the size $\|K\|$ of K as $|W| + |R|$.

The notation $K, w \models \varphi$ indicates that a CTL* state formula φ holds at the state w of the Kripke structure K. Similarly, $K, \pi \models \psi$ indicates that a CTL* path formula ψ holds on a path π of the Kripke structure K. When K is clear from the context, we write $w \models \varphi$ and $\pi \models \psi$. Also, $K \models \varphi$ if and only if $K, w^0 \models \varphi$.

The relation $\models$ is inductively defined as follows:

—For all w, we have $w \models$ **true** and $w \not\models$ **false**.

—$w \models p$ for $p \in AP$ iff $p \in L(w)$.

—$w \models \neg p$ for $p \in AP$ iff $p \notin L(w)$.

—$w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$.

—$w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.

—$w \models A\psi$ iff for every path $\pi = w_0, w_1, \ldots,$ with $w_0 = w$, we have $\pi \models \psi$.

—$w \models E\psi$ iff there exists a path $\pi = w_0, w_1, \ldots,$ with $w_0 = w$, such that $\pi \models \psi$.

—$\pi \models \varphi$ for a state formula φ, iff $w_0 \models \varphi$ where $\pi = w_0, w_1, \ldots$

—$\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.

—$\pi \models \psi_1 \vee \psi_2$ iff $\pi \models \psi_1$ or $\pi \models \psi_2$.

—$\pi \models X\psi$ iff $\pi^1 \models \psi$.

—$\pi \models \psi_1 U\psi_2$ iff there exists $i \geq 0$ such that $\pi^i \models \psi_2$ and for all $0 \leq j < i$, we have $\pi^j \models \psi_1$.

—$\pi \models \psi_1\tilde{U}\psi_2$ iff for all $i \geq 0$ such that $\pi^i \not\models \psi_2$, there exists $0 \leq j < i$ such that $\pi^j \models \psi_1$.

Note that $\pi \models \psi_1\tilde{U}\psi_2$ if and only if $\pi \not\models (\neg\,\psi_1)U(\neg\,\psi_2)$. That is, a path π satisfies $\psi_1\tilde{U}\psi_2$ if ψ_2 holds everywhere along π (thus, the U does not reach its eventuality), or if the first occurrence of $\neg\,\psi_2$ is strictly preceded by an occurrence of ψ_1 (thus, $\neg\,\psi_1$ is falsified before the eventuality is reached). Another way to understand the $\tilde{U}$ operator is to interpret $\psi_1\tilde{U}\psi_2$ by "as long as ψ_1 is false, ψ_2 must be true".

2.1.2. *The Propositional μ-Calculus*. The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Kozen 1983]. Specifically, we consider a μ-calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operators EX and AX, as well as least (μ) and greatest (ν) fixpoint operators. We assume that μ-calculus formulas are written in positive normal form (negation only applied to atomic propositions constants and variables). Formally, given a set AP of atomic proposition constants and a set APV of atomic proposition variables, a μ-calculus formula is either:

—**true, false**, p or $\neg p$ for all $p \in AP$;

—y for all $y \in APV$;

—$\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are μ-calculus formulas;

—$AX\varphi$ or $EX\varphi$, where φ is a μ-calculus formula;

—$\mu y.f(y)$ or $\nu y.f(y)$, where $y \in APV$ and $f(y)$ is a μ-calculus formula containing y as a free variable.

A *sentence* is a formula that contains no free atomic proposition variables. We call AX and EX *next modalities*, and we call μ and ν *fixpoint modalities*. We say that a μ-calculus formula is a *μ-formula* (*ν-formula*), if it is of the form $\mu y.f(y)$ ($\nu y.f(y)$). We use λ to denote a fixpoint modality μ or ν. For a λ-formula $\lambda y.f(y)$, the formula $f(\lambda y.f(y))$ is obtained from $f(y)$ by replacing each free occurrence of y with $\lambda y.f(y)$.

The closure, $cl(\varphi)$, of a μ-calculus sentence φ is the smallest set of μ-calculus 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 $AX\varphi \in cl(\varphi)$ or $EX\varphi \in cl(\varphi)$, then $\varphi \in cl(\varphi)$.

—If $\mu y.f(y) \in cl(\varphi)$, then $f(\mu y.f(y)) \in cl(\varphi)$.

—If $\nu y.f(y) \in cl(\varphi)$, then $f(\nu y.f(y)) \in cl(\varphi)$.

For example, for $\varphi = \mu y.(q \vee (p \wedge EXy))$, $cl(\varphi) = \{\varphi, q \vee (p \wedge EX\varphi), q, p \wedge EX\varphi, p, EX\varphi\}$. It follows from a result of Kozen [1983] that for every μ-calculus formula φ, the number of elements in $cl(\varphi)$ is linear with respect to a reduced DAG representation of φ. Accordingly, as with CTL*, we define size $\|\varphi\|$ of φ as the number of elements in $cl(\varphi)$.

Given a Kripke structure $K = \langle AP, W, R, w^0, L\rangle$, and a set $\{y_1, \ldots, y_n\}$ of free variables, a *valuation* $\mathcal{V}$: $\{y_1, \ldots, y_n\} \rightarrow 2^W$ 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 $\{y_1, \ldots, y_n\}$ 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 states 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 $p \in AP$, we have $p^K(\mathcal{V}) = \{w \in W : p \in L(w)\}$ and $(\neg p)^K(\mathcal{V}) = \{w \in W : p \notin L(w)\}$;

—For $y_i \in APV$, we have $y_i^K(\mathcal{V}) = \mathcal{V}(y_i)$;
—$(\varphi_1 \wedge \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cap \varphi_2^K(\mathcal{V})$;
—$(\varphi_1 \vee \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cup \varphi_2^K(\mathcal{V})$;
—$(AX\varphi)^K(\mathcal{V}) = \{w \in W : \forall w'$ such that $\langle w, w' \rangle \in R$, we have $w' \in \varphi^K(\mathcal{V})\}$;
—$(EX\varphi)^K(\mathcal{V}) = \{w \in W : \exists w'$ such that $\langle w, w' \rangle \in R$ and $w' \in \varphi^K(\mathcal{V})\}$;
—$(\mu y.f(y))^K(\mathcal{V}) = \bigcap\{W' \subseteq W : f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'\}$;
—$(\nu y.f(y))^K(\mathcal{V}) = \bigcup\{W' \subseteq W : W' \subseteq f^K(\mathcal{V}[y \leftarrow W'])\}$.

Note that no valuation is required for a sentence. For a state $w \in W$ and a sentence φ, we say that $w \models \varphi$ iff $w \in \varphi^K$. For example, the μ-calculus formula $\mu y.(q \vee (p \wedge EXy))$ is equivalent to the CTL formula $EpUq$.

A μ-calculus formula is *alternation free* if, for all $y \in APV$, there are respectively no occurrences of ν (μ) on any syntactic path from an occurrence of μy (νy) to an occurrence of y. For example, the formula $\mu x.(p \vee \mu y.(x \vee EXy))$ is alternation free and the formula $\nu x.\mu y.((p \wedge x) \vee EXy)$ is not alternation free. The *alternation-free μ-calculus* is a subset of μ-calculus containing only alternation-free formulas.

A μ-calculus formula is *guarded* if for all $y \in APV$, all the occurrences of y that are in a scope of a fixpoint modality λ are also in a scope of a next modality which is itself in the scope of λ. Thus, a μ-calculus sentence is *guarded* if for all $y \in APV$, all the occurrences of y are in the scope of a next modality. For example, the formula $\mu y.(p \vee EXy)$ is guarded and the formula $EX\mu y.(p \vee y)$ is not guarded. We assume that all μ-calculus formulas are guarded. By the theorem below, stated without a proof in Banieqbal and Barringer [1987], this can be done without loss of generality.

THEOREM 2.1. *Given a μ-calculus formula, we can construct, in linear time, an equivalent guarded formula.*

PROOF. We first define a function

$$new: \mu\text{-calculus formulas} \times \{\mu, \nu\} \times APV \rightarrow \mu\text{-calculus formulas}.$$

The formula $new(\varphi, \mu, y)$ is obtained from φ by replacing with **false** every occurrence of y that is not in a scope of a modality. Similarly, $new(\varphi, \nu, y)$ is obtained from φ by replacing with **true** every occurrence of y that is not in a scope of a modality. Formally, we define $new(\varphi, \lambda, y)$ by induction on the structure of φ as follows:

—$new(y, \mu, y) =$ **false**.—$new(y, \nu, y) =$ **true**.
—$new(\varphi_1 \wedge \varphi_2, \lambda, y) = new(\varphi_1, \lambda, y) \wedge new(\varphi_2, \lambda, y)$.
—$new(\varphi_1 \vee \varphi_2, \lambda, y) = new(\varphi_1, \lambda, y) \vee new(\varphi_2, \lambda, y)$.
—For all φ that differ from y, $\varphi_1 \wedge \varphi_2$, or $\varphi_1 \vee \varphi_2$, we have $new(\varphi, \lambda, y) = \varphi$.

For example, $new(y \vee p \vee EXy, \mu, y) = p \vee EXy$. We now prove that for all λ, y, and $f(y)$, we have

$$\lambda y.f(y) = \lambda y.new(f, \lambda, y)(y).$$

We consider here the case where $\lambda = \mu$. The proof for the case $\lambda = \nu$ is symmetric. By the semantics of μ-calculus, for a Kripke structure $K = \langle AP, W,$

R, w^0, $L\rangle$ and every $w \in W$ and valuation $\mathcal{V}$ for the free variables (except y) in $f(y)$, we have that $w \in \mu y.f(y)^K(\mathcal{V})$ if and only if $w \in \bigcap \{W' \subseteq W : f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'\}$. For every $W' \subseteq W$, we prove that

$$f^K(\mathcal{V}[y \leftarrow W']) \subseteq W' \text{ if and only if } new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'.$$

Assume first that $f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. As for every valuation $\mathcal{V}'$ we have $new(f, \mu, y)^K(\mathcal{V}') \subseteq f^K(\mathcal{V}')$, clearly $new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. For the second direction, we assume that f is given in a disjunctive normal form. We prove that if $new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$, then for every w, if there exists a disjunct g in f such that $w \in g^K(\mathcal{V}[y \leftarrow W'])$, then $w \in W'$. It follows that $f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. Let g be such that $w \in g^K(\mathcal{V}[y \leftarrow W'])$. We consider two cases.

(1) If $g = new(g, \mu, y)$, then $w \in new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W'])$, and since $new(f, \mu, y)^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$, we are done.

(2) Otherwise, there is a conjunct y in g and it must be that $g^K(\mathcal{V}[y \leftarrow W']) \subseteq W'$. Hence, we are also done.

At first glance, it might appear that to transform a μ-calculus formula into an equivalent guarded formula, it is sufficient to replace all subformulas $\lambda y.f(y)$ by $\lambda y.new(f, \lambda, y)(y)$. However, this transformation only ensures that the occurrences of y are in the scope of *some* modality, not necessarily of a next modality. Fortunately, one can easily work around this problem as follows. First notice that for the syntactically innermost $\lambda y.f(y)$, we have that $\lambda y.new(f, \lambda, y)(y)$ puts all occurrences of y in the scope of a next modality. So, we start the transformation with the innermost λ-formulas. Now, consider a $\lambda z.g(z)$ on the next outer syntactic level. In $\lambda z.new(g, \lambda, z)(z)$, the variable z can appear within an inner $\lambda y.new(f, \lambda, y)(y)$ without being in the scope of a next modality. To avoid this, instead of replacing the inner $\lambda y.f(y)$ by $\lambda y.new(f, \lambda, y)(y)$, we replace it by

$$new(f, \lambda, y)(\lambda y.new(f, \lambda, y)(y)), \tag{1}$$

which is semantically equivalent. In $new(f, \lambda, y)$, all the occurrences of y are in the scope of a next modality. Therefore, in (1), all occurrences of variables (e.g., z) that are in the scope of some modality are in the scope of a next modality. Thus, after this transformation, replacing $\lambda z.g(z)$ by $\lambda z.new(g, \lambda, z)(z)$ ensures that all occurrences of z are in the scope of a next modality, and replacing $\lambda z.g(z)$ by $new(g, \lambda, z)(\lambda z.new(g, \lambda, z)(z))$ allows the same transformation to be also used at the next outer syntactic level. Proceeding likewise to the outermost syntactic level we obtain an equivalent guarded formula. Finally, notice that for the outermost $\lambda z.g(z)$, replacing it by $\lambda z.new(g, \lambda, z)(z)$ is sufficient since there is no outer fixpoint modality. In Figure 1, we describe the translation procedure *guard*(ψ) formally. The procedure operates on formulas represented as reduced DAGs. It uses the function *order*, which, given a μ-calculus formula ψ, returns an ordered list of the μ and ν subformulas of ψ that preserves the subformula order.

For example, translation of the formula $\psi = \mu y.(p \vee \nu z.(y \vee (z \wedge AXz)))$ proceeds as follows:

(1) $order(\psi) = \langle \nu z.(y \vee (z \wedge AXz)), \psi \rangle$.

```
procedure guard($\psi$);
let $\langle \lambda_1 y_1.\varphi_1, \lambda_2 y_2.\varphi_2, \ldots, \lambda_n y_n.\varphi_n \rangle = order(\psi)$;
for $i = 1 \ldots n-1$ do
  let $\psi := \psi$ with $\lambda_i y_i.\varphi_i$ replaced by $new(\varphi_i, \lambda_i, y_i)(\lambda_i y_i.new(\varphi_i, \lambda_i, y_i)(y_i))$
od;
let $\psi := \psi$ with $\lambda_n y_n.\varphi_n$ replaced by $\lambda_n y_n.new(\varphi_n, \lambda_n, y_n)(y_n))$;
return $\psi$.
```

FIG. 1. Translating a formula to an equivalent guarded formula.

(2) $new(y \vee (z \wedge AXz), \nu, z)(z) = y \vee AXz$.

(3) Therefore, $new(y \vee (z \wedge AXz), \nu, z)(\nu z.new(y \vee (z \wedge AXz), \nu, z)(z)) = y \vee AX(\nu z.y \vee AXz)$.

(4) $new(p \vee y \vee AX(\nu z.y \vee AXz), \mu, y)(y) = p \vee AX(\nu z.y \vee AXz)$.

(5) Therefore, $\mu y.new(p \vee y \vee AX(\nu z.y \vee AXz), \mu, y)(y) = \mu y.p \vee AX(\nu z.y \vee AXz)$.

We now consider the size, $\|guard(\psi)\|$, of the guarded formula obtained from ψ. First, note that for all φ, λ, and y, we have $\|new(\varphi, \lambda, y)(y)\| \leq \|\varphi(y)\|$. Therefore, the only potential blow up in $guard(\psi)$ originates from the replacement of subformulas $\lambda y.\varphi(y)$ by $\varphi'(\lambda y.\varphi'(y))$, for φ' with $\|\varphi'\| \leq \|\varphi\|$. However, since by definition $\varphi'(\lambda y.\varphi'(y)) \in cl(\lambda y.\varphi'(y))$, this does not increase the size of the resulting formula. □

2.2. ALTERNATING TREE AUTOMATA. A *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 for all $0 \leq c' < c$, $x \cdot c' \in T$. 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. The number of successors of x is called the *degree* of x and is denoted by $d(x)$. A node is a *leaf* if it has no successors. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \pi$. Given an alphabet Σ, a Σ-*labeled tree* is a pair $\langle T, V \rangle$ where T is a tree and $V: T \rightarrow \Sigma$ maps each node of T to a letter in Σ. Note that an infinite word in Σ^ω can be viewed as a Σ-labeled tree in which the degree of all nodes is 1. Of special interest to us are Σ-labeled trees in which $\Sigma = 2^{AP}$ for some set AP of atomic propositions. We call such Σ-labeled trees *computation trees*. A computation tree $\langle T, V \rangle$ with $\Sigma = 2^{AP}$ can be viewed as the Kripke structure $K_{\langle T, V \rangle} = \langle AP, T, R_T, \epsilon, V \rangle$, where $R_T(x, x')$ iff x' is a successor of x in T. We sometimes refer to satisfaction of temporal logic formulas in a computation tree, meaning their satisfaction in this Kripke structure. Given a set $\mathcal{D} \subset \mathbb{N}$, a $\mathcal{D}$-tree is a computation tree in which all the nodes have degree in $\mathcal{D}$.

Automata over infinite trees (tree automata) run over Σ-labeled trees that have no leaves [Thomas 1990]. *Alternating automata* generalize nondeterministic tree automata and were first introduced in Muller and Schupp [1987] (see Slutzki [1985] for alternating automata on finite trees). For simplicity, we refer first to automata over binary trees (i.e., when $T = \{0, 1\}^*$). Consider a nondeterministic tree automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, δ is a transition function, $q_0 \in Q$ is an initial state, and F specifies the acceptance condition (a condition that defines a subset of Q^ω; we define several types of acceptance conditions below). The transition relation δ: $Q \times \Sigma \rightarrow 2^{S^2}$ 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 as it reads 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 $x \cdot 0$ (the left successor of x), and the other copy enters the state q_2 and proceeds to the node $x \cdot 1$ (the right successor of x).

For a given set X, 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 $X \backslash Y$ makes θ true. We can represent the transition relation δ of a nondeterministic automaton on binary trees using $\mathcal{B}^+(\{0, 1\} \times Q)$. For example, $\delta(q, \sigma) = \{\langle q_1, q_2 \rangle, \langle q_3, q_1 \rangle\}$ can be written as $\delta(q, \sigma) = (0, q_1) \wedge (1, q_2) \vee (0, q_3) \wedge (1, q_1)$, meaning that the automaton can choose between two possibilities. In the first, the copy that proceeds to direction 0 enters the state q_1 and the one that proceeds to direction 1 enters the state q_2. In the second, the copy that proceeds to direction 0 enters the state q_3 and the one that proceeds to direction 1 enters the state q_1.

In nondeterministic tree automata, each conjunction in δ has exactly one element associated with each direction. In alternating automata over binary trees, $\delta(q, \sigma)$ can be an arbitrary formula from $\mathcal{B}^+(\{0, 1\} \times Q)$. We can have, for instance, a transition

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

The above transition illustrates that several copies may go to the same direction and that the automaton is not required to send copies to all the directions. Formally, a finite alternating automaton over infinite binary trees is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ as in nondeterministic automata, only that the transition function is now δ: $Q \times \Sigma \rightarrow \mathcal{B}^+(\{0, 1\} \times Q)$.

We now generalize alternating automata to trees where nodes can have varying degrees. As we consider finite automata (and thus, in particular, automata with a finite transition function), our generalization is restricted to trees for which the set of all possible degrees is finite and known in advance. Explicitly, we require that each automaton has a finite set $\mathcal{D} \subset \mathbb{N}$ of possible degrees, specified in its definition. Then, the transition function is δ: $Q \times \Sigma \times \mathcal{D} \rightarrow \mathcal{B}^+(\mathbb{N} \times Q)$ with the requirement that for every $k \in \mathcal{D}$, we have $\delta(q, \sigma, k) \in \mathcal{B}^+(\{0, \ldots, k - 1\} \times Q)$. Thus, q, σ and k are all arguments of the transition function. When the automaton is in a state q as it reads a node that is labeled by a letter σ and has k successors, it applies the transition $\delta(q, \sigma, k)$. For each $q \in Q$ and $\sigma \in \Sigma$, we denote $\bigvee_{k \in \mathcal{D}} \delta(q, \sigma, k)$ by $\delta(q, \sigma)$. We define the *size* $\|\mathcal{A}\|$ of an automaton $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, q_0, F \rangle$ as $|\mathcal{D}| + |Q| + |F| + \|\delta\|$, where $|\mathcal{D}|$, $|Q|$, and $|F|$ are the respective cardinalities of the sets $\mathcal{D}$, Q, and F, and where $\|\delta\|$ is the sum of the lengths of the nonidentically false formulas that appear as $\delta(q, \sigma, k)$ for some $q \in Q$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$ (note that the restriction to nonidentically false formulas is to avoid an unnecessary $|Q| \cdot |\Sigma| \cdot |\mathcal{D}|$ minimal size for δ).

A run of an alternating automaton $\mathcal{A}$ over a tree $\langle T, V\rangle$ is a tree $\langle T_r, r\rangle$ in which the root is labeled by q_0 and every other node is labeled by an element of $\mathbb{N}^* \times Q$. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton over $\langle T, V\rangle$ there is a 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 $\Sigma_r = \mathbb{N}^* \times Q$ and $\langle T_r, r\rangle$ satisfies the following:

(1) $r(\epsilon) = (\epsilon, q_0)$.

(2) Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x), d(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_n, q_n)\} \subseteq \{0, \ldots, d(x) - 1\} \times Q$, such that the following hold:

—S satisfies θ, and

—for all $0 \leq i \leq n$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, q_i)$.

For example, if $\langle T, V\rangle$ is a binary tree with $V(\epsilon) = a$ and $\delta(q_0, a, 2) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then, at level 1, $\langle T_r, r\rangle$ includes a node labeled $(0, q_1)$ or a node labeled $(0, q_2)$, and includes a node labeled $(0, q_3)$ or a node labeled $(1, q_2)$. Note that if, for some y, the transition function δ has the value **true**, then y need not have successors. Also, δ can never have the value **false** in a run.

A run $\langle T_r, r\rangle$ is accepting if all its infinite paths satisfy the acceptance condition. We consider here Büchi, Rabin, Streett, and parity acceptance conditions. Given a run $\langle T_r, r\rangle$ and an infinite path $\pi \subseteq T_r$, let $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $V_r(y) \in \mathbb{N}^* \times \{q\}$. That is, $inf(\pi)$ contains exactly all the states that appear infinitely often in π. The four acceptance conditions are defined as follows:

—A path π satisfies a *Büchi* acceptance condition $F \subseteq Q$ if and only if $inf(\pi) \cap F \neq \emptyset$.

—A path π satisfies a *Rabin* acceptance condition $F = \{\langle G_1, B_1\rangle, \ldots, \langle G_m, B_m\rangle\}$, where for $1 \leq i \leq m$, $G_i \subseteq Q$ and $B_i \subseteq Q$, if and only if there exists a pair $\langle G_i, B_i\rangle \in F$ for which $inf(\pi) \cap G_i \neq \emptyset$ and $inf(\pi) \cap B_i = \emptyset$.

—A path π satisfies a *Streett* acceptance condition $F = \{\langle G_1, B_1\rangle, \ldots, \langle G_m, B_m\rangle\}$, where for $1 \leq i \leq m$, $G_i \subseteq Q$ and $B_i \subseteq Q$, if and only if for all pairs $\langle G_i, B_i\rangle \in F$, either $inf(\pi) \cap G_i = \emptyset$ or $inf(\pi) \cap B_i \neq \emptyset$.

—A path π satisfies a *parity* acceptance condition $F = \{F_1, F_2, \ldots, F_k\}$ with $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_k$ iff the minimal index i for which $inf(\pi) \cap F_i \neq \emptyset$ is even. The number k of sets in F is called the *index* of the automaton.

An automaton accepts a tree if and only if there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts. Note that if $\mathcal{D}$ is a singleton, then $\mathcal{A}$ runs over trees with a fixed branching degree. We then say that $\mathcal{A}$ is a *fixed arity* tree automaton. In particular, note that an alternating automaton over infinite words is simply an alternating automaton over infinite trees with $\mathcal{D} = \{1\}$. Formally, we define an alternating automaton over infinite words as $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ where $\delta: Q \times \Sigma \rightarrow \mathcal{B}^+(Q)$.

Muller et al. [1986] introduce *weak alternating automata* (WAAs). In a WAA, we have a Büchi acceptance condition $F \subseteq Q$ and there exists a partition of Q into disjoint sets, $Q_1, \ldots, Q_m$, such that for each set Q_i, either $Q_i \subseteq F$, in which case Q_i is an *accepting set*, or $Q_i \cap F = \emptyset$, in which case Q_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma, k)$, for some $\sigma \in \Sigma$ and $k \in \mathcal{D}$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of a WAA ultimately gets "trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set. Indeed, a run visits infinitely many states in F if and only if it gets trapped in an accepting set. We sometimes refer to the *type* of an automaton, meaning its acceptance condition, and its being weak or not weak.

We call the partition of Q into sets the *weakness partition* and we call the partial order over the sets of the weakness partition the *weakness order*. Often (in particular, in all the cases we consider in this work) a WAA is given together with its weakness partition and order. Otherwise, as we claim below, these can be induced by the partition of the graph of the WAA into *maximal strongly connected components* (MSCCs). Formally, given $\mathcal{A}$, let $G_{\mathcal{A}}$ be a directed graph induced by $\mathcal{A}$; that is, the vertices of $G_{\mathcal{A}}$ are states in $\mathcal{A}$ and there is an edge from vertex q to vertex q' iff there is a transition in $\mathcal{A}$ from the state q that involves the state q'. Let $C_1, \ldots, C_n$ be a partition of $G_{\mathcal{A}}$ to maximal strongly connected components. That is, for every C_i and for every two vertices q and q' in C_i, there is a path from q to q' and from q' to q, and for every vertex q'', the set $C_i \cup \{q''\}$ no longer satisfies this condition. Since the partition to the MSCCs is maximal, there is a partial order $\leq$ between them so that $C_i \leq C_j$ iff C_i is reachable from C_j.

THEOREM 2.2. *Given a WAA $\mathcal{A}$, the partition of its states to maximal strongly connected components is a weakness partition with a weakness order $\leq$.*

PROOF. Let $C_1, \ldots, C_n$ be the MSCCs of $G_{\mathcal{A}}$. We show that each C_i is either contained in F or disjoint from F. Since $\mathcal{A}$ is weak, there is weakness partition $Q_1, \ldots, Q_m$ for $\mathcal{A}$. We claim that the partition to the C_i's refines the partition to the Q_i's, in the sense that each set Q_i is a union of maximal strongly connected components. Indeed, since there is a path between every two states in a strongly connected component, all the states of a component must belong to the same set Q_i. Now, since each Q_i is either contained in F or disjoint from F, the same holds for all the maximal strongly connected components, and we are done. □

It follows, by Tarjan [1972], that when a weakness partition and order are not given, they can be found in linear running time.

Remark 2.3. Since the modalities of conventional temporal logics, such as CTL* and the μ-calculus, do not distinguish between the various successors of a node (that is, they impose requirements either on all the successors of the node or on some successor), the alternating tree automata that one gets by translating formulas to automata are of a special structure, in which whenever the automaton reads a node x and a state s is sent to direction c, the state s is sent to all the directions $c' \in d(x)$, in either a disjunctive or conjunctive manner. Formally, following the notations in Graedel and Walukiewicz [1999] the formulas in $\mathcal{B}^+(\{0, \ldots, k-1\} \times Q)$ that appear in transitions $\delta(q, \sigma, k)$ of such alternating tree automata are members of $\mathcal{B}^+(\{\Box, \Diamond\} \times Q)$, where $\Box s$ stands for $\bigvee_{c \in \{0, \ldots, k-1\}}(c, s)$ and $\Diamond s$ stands for $\bigvee_{c \in \{0, \ldots, k-1\}}(c, s)$. We say that an alternating tree automaton is *symmetric* if it has the special structure described above. As detailed in Section 4, all the alternating tree automata we use in this work are symmetric, and thus can also be described using transitions in $\mathcal{B}^+(\{\Box, \Diamond\} \times Q)$. □

3. *Alternating Automata and Model Checking*

In this section, we introduce an automata-theoretic approach to model checking for branching temporal logic. The model-checking problem for a branching temporal logic is as follows: Given a Kripke structure K and a branching temporal formula ψ, determine whether $K \models \psi$. Recall that for linear temporal logic, each Kripke structure may correspond to infinitely many computations. Model checking is thus reduced to checking inclusion between the set of computations allowed by the Kripke structure and the language of an automaton describing the formula [Vardi and Wolper 1986a]. For branching temporal logic, each Kripke structure corresponds to a single nondeterministic computation. On that account, model checking is reduced to checking the membership of this computation in the language of the automaton describing the formula [Wolper 1989]. We show here that alternating automata are the suitable framework for automata-based model-checking algorithms. Alternation is used to reduce the size of the automaton from exponential in the length of ψ to linear in the length of ψ.

A Kripke structure $K = \langle AP, W, R, w^0, L\rangle$ can be viewed as a tree $\langle T_K, V_K\rangle$ that corresponds to the unwinding of K from w^0. Formally, for every node w, let $d(w)$ denote the degree of w (i.e., the number of successors that w has, and note that for all w we have $d(w) \geq 1$), and let $succ_R(w) = \langle w_0, \ldots, w_{d(w)-1}\rangle$ be an ordered list of w's R-successors (we assume that the nodes of W are ordered). We define T_K and V_K inductively as follows:

(1) $V_K(\epsilon) = w^0$.

(2) For $y \in T_K$ with $succ_R(V_K(y)) = \langle w_0, \ldots, w_m\rangle$ and for $0 \leq i \leq m$, we have $y \cdot i \in T_K$ and $V_K(y \cdot i) = w_i$.

We will sometimes view $\langle T_K, V_K\rangle$ as a computation tree over 2^{AP}, taking the label of a node to be $L(V_K(x))$ instead of $V_K(x)$. Which interpretation is intended will be clear from the context.

Let ψ be a branching temporal formula and let $\mathcal{D} \subset \mathbb{N}$ be the set of degrees of a Kripke structure K. Suppose that $\mathcal{A}_{\mathcal{D},\psi}$ is an alternating automaton that accepts exactly all the $\mathcal{D}$-trees that satisfy ψ. Consider a product of K and $\mathcal{A}_{\mathcal{D},\psi}$; that is, an automaton that accepts the language $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi}) \cap \{\langle T_K, V_K\rangle\}$. The language of this product either contains the single tree $\langle T_K, V_K\rangle$, in which case $K \models \psi$, or is empty, in which case $K \not\models \psi$. This discussion suggests the following automata-based model-checking algorithm. Given a branching temporal formula ψ and a Kripke structure K with degrees in $\mathcal{D}$, proceed as follows:

(1) Construct the alternating automaton $\mathcal{A}_{\mathfrak{D},\psi}$.

(2) Construct an alternating automaton $\mathcal{A}_{K,\psi} = K \times \mathcal{A}_{\mathfrak{D},\psi}$ by taking the product of K and $\mathcal{A}_{\mathfrak{D},\psi}$. This automaton simulates a run of $\mathcal{A}_{\mathfrak{D},\psi}$ over $\langle T_K, V_K \rangle$.

(3) Output "Yes" if $\mathcal{L}(\mathcal{A}_{K,\psi}) \neq \emptyset$, and "No", otherwise.

The type of $\mathcal{A}_{\mathfrak{D},\psi}$ and, consequently, the type of $\mathcal{A}_{K,\psi}$ as well as the complexity of its nonemptiness test, depend on the logic in which ψ is specified. The crucial point in our approach is that the automaton $\mathcal{A}_{K,\psi}$ can be defined as a word automaton over a 1-letter alphabet; this reduces the complexity of the nonemptiness test. In the remainder of this section, we discuss the 1-letter nonemptiness problem and present the product automaton $\mathcal{A}_{K,\psi}$.

3.1. THE 1-LETTER NONEMPTINESS PROBLEM. The nonemptiness problem for nondeterministic word automata is reducible to the 1-letter nonemptiness problem for them; instead of checking the nonemptiness of an automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, one can check the nonemptiness of the automaton $\mathcal{A}' = \langle \{a\}, Q, \delta', q_0, F \rangle$, where for all $q \in Q$, we have $\delta'(q, a) = \bigcup_{\sigma \in \Sigma} \delta(q, \sigma)$. It is easy to see that if $\mathcal{A}$ accepts some word, then $\mathcal{A}'$ accepts a^{ω}. Also, as each transition of $\mathcal{A}'$ originates from a transition of $\mathcal{A}$, it is not hard to see that if $\mathcal{A}'$ accepts a^{ω}, then there exists a word that is accepted by $\mathcal{A}$. One way to view this is that $\mathcal{A}'$ first guesses an input word and then proceeds like $\mathcal{A}$ over this word.

This reduction, however, is not valid for alternating word automata: if $\mathcal{A}'$ accepts a^{ω}, it is still not guaranteed that $\mathcal{A}$ accepts some word! Indeed, a necessary condition for the validity of the reduction is that the points of the run of $\mathcal{A}'$ that correspond to the same suffix guess the same Σ-labeling for this suffix, but nothing enforces this. This problem does not occur when $\mathcal{A}$ is defined over a singleton alphabet, because there is only one possible word. (Note however that this simplification does not hold for *finite words* since the length of the word is then a distinguishing factor.)

The theorem below relates the complexity of the nonemptiness problem for various classes of automata on infinite words.

A formula in $\mathcal{B}^+(X)$ is *simple* if it either atomic, **true, false**, or has the form $x * y$, where $* \in \{\bigwedge, \bigvee\}$ and $x, y \in X$. An alternating automaton is simple if all its transitions are simple. Thus, in each transition, a simple alternating automaton splits into two copies in either a universal or an existential (when $*$ is $\bigvee$) mode. Note that states of a 1-letter simple alternating automaton corresponds to nodes in an AND/OR graph.

THEOREM 3.1. *For automata of a given type* (*i.e.*, *Büchi*, *Rabin*, *Street*, *weak or not*, . . .), *the following three problems are intereducible in linear time and logarithmic space*.

(1) 1-*letter nonemptiness of alternating word automata*.

(2) 1-*letter nonemptiness of simple alternating word automata*.

(3) *Nonemptiness of nondeterministic fixed-arity tree automata*.

PROOF. We prove the theorem by reducing problem (1) to problem (2), reducing problem (2) to problem (3), and reducing problem (3) to problem (1). Before we get to the proof, we define the two functions *deflate*: $\mathcal{B}^+(\mathbb{N} \times Q) \rightarrow$

$\mathcal{B}^+(Q)$ and *inflate*: $\mathcal{B}^+(Q) \to \mathcal{B}^+(\mathbb{N} \times Q)$. For a formula $\theta \in \mathcal{B}^+(\mathbb{N} \times Q)$, the formula *deflate*$(\theta)$ is the formula obtained from θ by replacing each atom (c, q) in θ by the atom q. The function *inflate* is defined with respect to an enumeration $q_0, q_1, \ldots, q_n$ of the states in Q. For a formula $\theta \in \mathcal{B}^+(Q)$, the formula *inflate*$(\theta)$ is the formula in $\mathcal{B}^+(\{0, \ldots n\} \times Q)$ obtained from θ by replacing each atom q_c in θ by the atom (c, q_c). For example,

—*deflate*$(((0, q_0) \wedge (0, q_2)) \vee ((0, q_2) \wedge (1, q_2) \wedge (1, q_1))) = (q_0 \wedge q_2) \vee (q_2 \wedge q_2 \wedge q_1)$.

—*inflate*$((q_0 \wedge q_2) \vee (q_2 \wedge q_2 \wedge q_1)) = ((0, q_0) \wedge (2, q_2)) \vee ((2, q_2) \wedge (2, q_2) \wedge (1, q_1))$.

We start by reducing problem (1) to problem (2). Given an alternating word automaton $\mathcal{A} = \langle \{a\}, Q, \delta, q_0, F\rangle$, we define an alternating word automaton $\mathcal{A}' = \langle \{a\}, Q', \delta', q_0, F'\rangle$, such that for all $q \in Q'$, the formula $\delta'(q, a)$ is simple, and $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. The idea is that since all the suffixes of a^ω are the same, we can regard transitions of $\mathcal{A}'$ as ϵ-transitions and replace a transition in δ with several simple transitions in δ'. This requires an extension of the state set of $\mathcal{A}$ with at most the number of subformulas of transitions in δ. We assume, without loss of generality, that for all $q \in Q$, the parse tree of the formula $\delta(q, a)$ is binary; that is, $\delta(q, a)$ is either **true, false**, or has the form $\theta_1 * \theta_2$ ($*$ denotes $\wedge$ or $\vee$). We define Q' inductively as follows:

—For every $q \in Q$, we have $q \in Q'$.

—For every $q \in Q$ with $\delta(q, a) = \theta_1 * \theta_2$, we have $\theta_1 \in Q'$ and $\theta_2 \in Q'$.

—For every $\theta_1 * \theta_2 \in Q'$, we have $\theta_1 \in Q'$ and $\theta_2 \in Q'$.

Thus, a state in Q' is either $q \in Q$, or $\theta_1 * \theta_2 \in \mathcal{B}^+(Q)$. Moreover, Q' contains all the formulas in $\mathcal{B}^+(Q)$ that are strict subformulas of transitions in δ. We define δ' as follows:

—$\delta'(q, a) = \delta(q, a)$. That is, if $\delta(q, a)$ is **true** or **false**, so is $\delta'(q, a)$. Otherwise, $\delta(q, a)$ is of the form $\theta_1 * \theta_2$, in which case the subformulas θ_1 and θ_2 are regarded in $\delta'(q, a)$ as states.

—$\delta'(\theta_1 * \theta_2, a) = \theta_1 * \theta_2$.

It is easy to see that all the transitions in δ' are simple.

Consider the alphabets $\Sigma = \mathbb{N}^* \times Q$ and $\Sigma' = \mathbb{N}^* \times Q'$, and consider an infinite word b' over Σ'. The (possibly finite) word $b'_{|\Sigma}$ is obtained from b' by restricting it to its letters in Σ. We say that a word b' over Σ' *corresponds* to a word b over Σ iff $b'_{|\Sigma} = b$. Consider a run $\langle T', r'\rangle$ of $\mathcal{A}'$ (recall that one can regard a run of an alternating word automaton as a run of an alternating tree automaton with $\mathcal{D} = \{1\}$). For every $F \subseteq Q$ and for every path $\pi' \subseteq T'$, we have that π' visits infinitely many states in F iff so does $\pi'_{|\Sigma}$. Accordingly, as all our acceptance conditions only refer to visiting subsets of Q infinitely often, we define $F' = F$.

Every run $\langle T, r\rangle$ of $\mathcal{A}$ corresponds to a run $\langle T', r'\rangle$ of $\mathcal{A}'$, in the sense that for each path $\pi \subseteq T$ there exists a path $\pi' \subseteq T'$ such that $r'(\pi')$ corresponds to $r(\pi)$, and vice versa. Indeed, the run $\langle T', r'\rangle$ proceeds exactly as $\langle T, r\rangle$, only in “smaller steps”. Similarly, each run $\langle T', r'\rangle$ of $\mathcal{A}'$ corresponds to a run $\langle T, r\rangle$ of $\mathcal{A}$. Here, $\langle T, r\rangle$ proceeds in “larger steps”. By the definition of δ', it is guaranteed that, when necessary, $\mathcal{A}$ can cluster several transitions of $\mathcal{A}'$ into a single transition. Since $F' = F$, we thus have that $\langle T, r\rangle$ is accepting iff $\langle T', r'\rangle$ is accepting. Thus, $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$.

We prove that $\mathcal{A}$ and $\mathcal{A}'$ are of the same type. Clearly, they agree on the type of their acceptance conditions. We show that if $\mathcal{A}$ is weak then so is $\mathcal{A}'$. Let $\{Q_1, Q_2, \ldots, Q_n\}$ be the partition of Q into sets such that $Q_1 \leq \cdots \leq Q_n$ is an extension of the partial order to a total order. We define a partition $\{Q'_1, Q'_2, \ldots, Q'_n\}$ of Q' as follows. A state $q' \in Q'$ is a member of Q'_i iff one of the following holds:

(1) $q' \in Q_i$, or
(2) $i = \min\{j : q'$ is a subformula of $\delta(q, a)$ for $q \in Q_j\}$.

That is, a state that corresponds to a state of $\mathcal{A}$ remains in its set there. A state associated with a subformula θ joins the least Q_i from which θ is reachable. It is easy to see that $\{Q'_1, Q'_2, \ldots, Q'_n\}$ is a valid partition, with the ordering $Q'_1 \leq \cdots \leq Q'_n$.

We now reduce problem (2) to problem (3). Let $\mathcal{A} = \langle\{a\}, Q, \delta, q_0, F\rangle$ be a simple alternating word automaton. Consider the alternating tree automaton $\mathcal{A}' = \langle\{a\}, \{n\}, Q, \delta', q_0, F\rangle$, where $n = |Q|$ and for all $q \in Q$ we have that $\delta'(q, a, n) = \mathit{inflate}(\delta(q, a))$. The fact that the transitions in δ are simple guarantees that so are the transitions in δ'. Also, the definition of *inflate* guarantees that for all $q \in Q$ and all atoms (i, q_j) in $\delta'(q, a, n)$, we have that $i = j$. Thus, if $\delta'(q, a, n)$ is a conjunction that sends two copies of the automaton to the same direction, then these copies enter the same state and they are equivalent to a single copy sent to this direction. Hence, $\mathcal{A}'$ can be viewed as a nondeterministic tree automaton. Formally, let $\mathcal{A}'' = \langle\{a\}, \{n\}, Q'', \delta'', q_0, F''\rangle$ be a nondeterministic tree automaton, where

—$n = |Q|$,
—$Q'' = Q \cup \{q_{acc}\}$,
—$\delta'' : Q'' \times \{a\} \times \{n\} \to 2^{Q''^{n}}$ is defined as follows:

—If $\delta'(q, a, n) = \mathbf{true}$, then $\delta''(q, a, n) = \{\langle q_{acc}, q_{acc}, \ldots, q_{acc}\rangle\}$.
—If $\delta'(q, a, n) = \mathbf{false}$, then $\delta''(q, a, n) = \emptyset$.
—If $\delta'(q, a, n) = (c_1, q_{c_1}) \wedge (c_2, q_{c_2})$, then $\delta''(q, a, n) = \{\langle s_0, s_1, \ldots, s_{n-1}\rangle\}$, where $s_{c_1} = q_{c_1}$, $s_{c_2} = q_{c_2}$, and $s_i = q_{acc}$ for $i \notin \{c_1, c_2\}$.
—If $\delta'(q, a, n) = (c_1, q_{c_1}) \vee (c_2, q_{c_2})$, then $\delta''(q, a, n) = \{\langle s^1_0, s^1_1, \ldots, s^1_{n-1}\rangle, \langle s^2_0, s^2_1, \ldots, s^2_{n-1}\rangle\}$, where $s^1_{c_1} = q_{c_1}$, $s^2_{c_2} = q_{c_2}$, and $s^j_i = q_{acc}$ for $(j, i) \notin \{(1, c_1), (2, c_2)\}$.
—$\delta''(q_{acc}, a, n) = \{\langle q_{acc}, q_{acc}, \ldots, q_{acc}\rangle\}$.
—F'' extends F by making q_{acc} an accepting sink. Thus, for example, if $\mathcal{A}'$ is a Büchi automaton, then $F'' = F \cup \{q_{acc}\}$.

Clearly, $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}'')$. We show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Assume first that $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Then, there exists an accepting run $\langle T, r\rangle$ of $\mathcal{A}$ over a^ω. It is easy to see that the tree $\langle T, r'\rangle$, in which $r'(\epsilon) = r(\epsilon)$ and for all $y \cdot c \in T$ with $r'(y) = (x, q)$ and $r(y \cdot c) = (0^{|x|+1}, q_i)$, we have $r'(y \cdot c) = (x \cdot i, q_i)$, is an accepting run of $\mathcal{A}'$ over the a-labeled $|Q|$-ary tree. Assume now that $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Then, there exists an accepting run $\langle T, r'\rangle$ of $\mathcal{A}'$ over the a-labeled $|Q|$-ary tree.

It is easy to see that the tree $\langle T, r \rangle$, in which for all $y \in T$ with $r'(y) = (x, q)$ we have $r(y) = (0^{|x|}, q)$, is an accepting run of $\mathcal{A}$ over a^ω. It is also easy to see that the type of $\mathcal{A}$ is preserved.

It is left to reduce problem (3) to problem (1). The nonemptiness problem for nondeterministic tree automata is reducible to their 1-letter nonemptiness problem [Rabin 1969]. In addition, the reduction preserves the type of the automaton. Hence, as nondeterministic tree automata are a special case of alternating tree automata, we reduce the problem of 1-letter nonemptiness for alternating fixed-arity tree automata to problem (1).

Consider a 1-letter alternating tree automaton $\mathcal{A}$ over n-ary trees. Since the 1-letter n-ary tree is homogeneous (that is, all its subtrees are identical), then it is not important to which direction $\mathcal{A}$ sends new copies in each of its transitions. Thus, we can assume, without loss of generality, that for every atom (i, q_j) in the transitions of $\mathcal{A}$ we have that $i = j$. Given an alternating tree automaton $\mathcal{A} = \langle \{a\}, \{n\}, Q, \delta, q_0, F \rangle$ with this property, consider the alternating word automaton $\mathcal{A}' = \langle \{a\}, Q, \delta', q_0, F \rangle$, where for all $q \in Q$ we have $\delta'(q, a) = \mathit{deflate}(\delta(q, a, n))$. We show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Intuitively, since the 1-letter tree that could be accepted is unique, its branching structure is not important.

Assume first that $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Then, there exists an accepting run $\langle T, r \rangle$ of $\mathcal{A}$ over the n-ary a-labeled tree. It is easy to see that the tree $\langle T, r' \rangle$, in which for all $y \in T$ with $r(y) = (x, q)$ we have $r'(y) = (0^{|x|}, q)$, is an accepting run of $\mathcal{A}'$ over a^ω. Assume now that $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Then, there exists an accepting run $\langle T, r' \rangle$ of $\mathcal{A}'$ over a^ω. Consider the tree $\langle T, r \rangle$ in which $r(\epsilon) = r'(\epsilon)$ and for all $y \cdot c \in T$ with $r(y) = (x, q)$ and $r'(y \cdot c) = (0^{|x|+1}, q_i)$, we have $r(y \cdot c) = (x \cdot i, q_i)$. As we assumed that all atoms (i, q_j) in δ have $i = j$, it is easy to see that $\langle T, r \rangle$ is an accepting run of $\mathcal{A}$. Clearly, the reduction preserves the type of $\mathcal{A}$. □

Later in the paper, we will study the complexity of the 1-letter nonemptiness problem.

3.2. The Product Automaton. In this section, we present the core step of our approach. Given the alternating tree automaton $\mathcal{A}_{\mathcal{D},\psi}$ and a Kripke structure K, we define their product $\mathcal{A}_{K,\psi}$ as a 1-letter alternating word automaton. Thus, taking the product with K gives us two things. First, we move from a tree automaton to a word automaton. Second, we move from an automaton over an alphabet 2^{AP} to a 1-letter automaton. Obviously, the nonemptiness problem for tree automata can not, in general, be reduced to the nonemptiness problem of word automata. Also, as discussed above, the nonemptiness problem for alternating word automata cannot, in general, be reduced to the 1-letter nonemptiness problem. It is taking the product with K that makes both reductions valid here. Since each state in $\mathcal{A}_{K,\psi}$ is associated with a state w of K, then each state has the exact information as to which subtree of $\langle T_K, V_K \rangle$ it is responsible for (i.e., which subtree it would have run over if $\mathcal{A}_{K,\psi}$ had not been a 1-letter word automaton). The branching structure of $\langle T_K, V_K \rangle$ and its 2^{AP}-labeling are thus embodied in the states of $\mathcal{A}_{K,\psi}$. In particular, it is guaranteed that all the copies of the product automaton that start in a certain state, say one associated with w, follow the same labeling: the one that corresponds to computations of K that start in w.

Let $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, Q_\psi, \delta_\psi, q_0, F_\psi \rangle$ be an alternating tree automaton that accepts exactly all the $\mathcal{D}$-trees that satisfy ψ (the details of the construction of

$\mathcal{A}_{\mathfrak{D},\psi}$ depend on the logic in which ψ is specified; we consider some examples in the next section) and let $K = \langle AP, W, R, w^0, L\rangle$ be a Kripke structure with degrees in $\mathfrak{D}$. The product automaton of $\mathcal{A}_{\mathfrak{D},\psi}$ and K is an alternating word automaton $\mathcal{A}_{K,\psi} = \langle\{a\}, W \times Q_\psi, \delta, \langle w^0, q_0\rangle, F\rangle$ where δ and F are defined as follows:

—Let $q \in Q_\psi$, $w \in W$, $succ_R(w) = \langle w_0, \ldots, w_{d(w)-1}\rangle$, and $\delta_\psi(q, L(w), d(w)) = \theta$. Then $\delta(\langle w, q\rangle, a) = \theta'$, where θ' is obtained from θ by replacing each atom (c, q') in θ by the atom $\langle w_c, q'\rangle$.

—The acceptance condition F is defined according to the acceptance condition F_ψ of $\mathcal{A}_{\mathfrak{D},\psi}$. If $F_\psi \subseteq Q_\psi$ is a Büchi condition, then $F = W \times F_\psi$ is also a Büchi condition. If $F_\psi = \{\langle G_1, B_1\rangle, \ldots, \langle G_m, B_m\rangle\}$ is a Rabin (or Streett) condition, then $F = \{\langle W \times G_1, W \times B_1\rangle, \ldots, \langle W \times G_m, W \times B_m\rangle\}$ is also a Rabin (or Streett) condition.

It is easy to see that $\mathcal{A}_{K,\psi}$ is of the same type as $\mathcal{A}_{\mathfrak{D},\psi}$. In particular, if $\mathcal{A}_{\mathfrak{D},\psi}$ is a WAA (with a partition $\{Q_1, Q_2, \ldots, Q_n\}$), then so is $\mathcal{A}_{K,\psi}$ (with a partition $\{W \times Q_1, W \times Q_2, \ldots, W \times Q_n\}$).

PROPOSITION 3.2

(1) $\|\mathcal{A}_{K,\psi}\| = O(\|K\| \cdot \|\mathcal{A}_{\mathfrak{D},\psi}\|)$.

(2) *$\mathcal{L}(\mathcal{A}_{K,\psi})$ is nonempty if and only if $K \models \psi$.*

PROOF. (1) follows easily from the definition of $\mathcal{A}_{K,\psi}$. Indeed, $|W \times Q_\psi| = |W| * |Q_\psi|$, $\|\delta\| = |W| * \|\delta_\psi\|$, and $|F| = |W| * |F_\psi|$.

To prove (2), we show that $\mathcal{L}(\mathcal{A}_{K,\psi})$ is nonempty if and only if $\mathcal{A}_{\mathfrak{D},\psi}$ accepts $\langle T_K, V_K\rangle$. Since $\mathcal{A}_{\mathfrak{D},\psi}$ accepts exactly all the $\mathfrak{D}$-trees that satisfy ψ, and since all the degrees of T_K are in $\mathfrak{D}$, the later holds if and only if $K \models \psi$. Given an accepting run of $\mathcal{A}_{\mathfrak{D},\psi}$ over $\langle T_K, V_K\rangle$, we construct an accepting run of $\mathcal{A}_{K,\psi}$. Also, given an accepting run of $\mathcal{A}_{K,\psi}$, we construct an accepting run of $\mathcal{A}_{\mathfrak{D},\psi}$ over $\langle T_K, V_K\rangle$.

Assume first that $\mathcal{A}_{\mathfrak{D},\psi}$ accepts $\langle T_K, V_K\rangle$. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathfrak{D},\psi}$ over $\langle T_K, V_K\rangle$. Recall that T_r is labeled with $\mathbb{N}^* \times Q_\psi$. A node $y \in T_r$ with $r(y) = (x, q)$ corresponds to a copy of $\mathcal{A}_{\mathfrak{D},\psi}$ that is in the state q and reads the tree obtained by unwinding K from $V_K(x)$. Consider the tree $\langle T_r, r'\rangle$ where T_r is labeled with $0^* \times W \times Q_\psi$ and for every $y \in T_r$ with $r(y) = (x, q)$, we have $r'(y) = (0^{|x|}, V_K(x), q)$. We show that $\langle T_r, r'\rangle$ is an accepting run of $\mathcal{A}_{K,\psi}$. In fact, since $F = W \times F_\psi$, we only need to show that $\langle T_r, r'\rangle$ is a run of $\mathcal{A}_{K,\psi}$; acceptance follows from the fact that $\langle T_r, r\rangle$ is accepting. Intuitively, $\langle T_r, r'\rangle$ is a “legal” run, since the W-component in r' always agrees with V_K. This agreement is the only additional requirement of δ with respect to δ_ψ. Consider a node $y \in T_r$ with $r(y) = (x, q)$, $V_K(x) = w$, and $succ_R(w) = \langle w_0, \ldots, w_{k-1}\rangle$. Let $\delta_\psi(q, w, k) = \theta$. Since $\langle T_r, r\rangle$ is a run of $\mathcal{A}_{\mathfrak{D},\psi}$, there exists a set $\{(c_0, q_0), (c_1, q_1), \ldots, (c_n, q_n)\}$ satisfying θ, such that the successors of y in T_r are $y \cdot i$, for $1 \leq i \leq n$, each labeled with $(x \cdot c_i, q_i)$. In $\langle T_r, r'\rangle$, by its definition, $r'(y) = (0^{|x|}, w, q)$ and the successors of y are $y \cdot i$, each labeled with $(0^{|x+1|}, w_{c_i}, q_i)$. Let $\delta(q, a) = \theta'$. By the definition of δ, the set $\{(w_{c_0}, q_0), (w_{c_1}, q_1), \ldots, (w_{c_n}, q_n)\}$ satisfies θ'. Thus, $\langle T_r, r'\rangle$ is a run of $\mathcal{A}_{K,\psi}$.

Assume now that $\mathcal{A}_{K,\psi}$ accepts a^ω. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{K,\psi}$. Recall that T_r is labeled with $0^* \times W \times Q_\psi$. Consider the tree $\langle T_r, r'\rangle$

labeled with $\mathbb{N}^* \times Q_\psi$, where $r'(\epsilon) = (\epsilon, q_0)$ and for every $y \cdot c \in T_r$ with $r'(y) \in \{x\} \times Q_\psi$ and $r(y \cdot c) = (0^{|x+1|}, w, q)$, we have $r'(y \cdot c) = (x \cdot i, q)$, where i is such that $V_K(x \cdot i) = w$. As in the previous direction, it is easy to see that $\langle T_r, r' \rangle$ is an accepting run of $\mathcal{A}_{\mathfrak{D},\psi}$ over $\langle T_K, V_K \rangle$. □

Proposition 3.2 can be viewed as an automata-theoretic generalization of Theorem 4.1 in Emerson et al. [1993].

In conclusion, given an alternating automaton $\mathcal{A}_{\mathfrak{D},\psi}$ such that $\mathcal{A}_{\mathfrak{D},\psi}$ accepts exactly all the $\mathfrak{D}$-trees that satisfy ψ, model checking of a Kripke structure K with degrees in $\mathfrak{D}$ with respect to ψ is reducible to checking the 1-letter nonemptiness of a word automaton of the same type as $\mathcal{A}_{\mathfrak{D},\psi}$ and of size $O(\|K\| * \|\mathcal{A}_{\mathfrak{D},\psi}\|)$. Similar approaches, where *Boolean graphs* and *games* are used for model checking and for bisimulation checking are presented in Anderson [1992], Larsen [1992], and Andersen and Vergauwen [1995] (Boolean graphs) and Stirling [1996] (games). In the following sections, we show how this approach can be used to derive, in a uniform way, known complexity bounds for model checking of several branching temporal logics, as well as to obtain new space complexity bounds.

4. *Primary Applications*

In the previous section, we presented an automata-based method for model checking of branching temporal logics. The efficiency of our method depends on the efficiency of the translation of branching temporal logic formulas to automata as well as the efficiency of the 1-letter nonemptiness test for them. In this section, we present an application of the method with respect to CTL, the alternation-free μ-calculus, and the μ-calculus.

4.1. MODEL CHECKING FOR CTL AND ALTERNATION-FREE M-CALCULUS. Vardi and Wolper showed how to solve the satisfiability problem for CTL via an exponential translation of CTL formulas to Büchi automata over infinite trees [Vardi and Wolper, 1986b].[2] Muller et al. [1988] provided a simpler proof, via a linear translation of branching dynamic logic formulas to WAAs. We exploit here the ideas of Muller et al. [1988] by demonstrating a linear translation from CTL formulas to WAAs. The idea is simple: each state in the automaton for ψ corresponds to a subformula of ψ. The transitions of the automaton then follows the semantics of CTL. For example, the automaton for the formula EFp has a single state q. When the automaton in state q reads a node in which p holds, its task is completed, and the transition function is **true**. When the automaton is state q reads a node in which p does not hold, it sends a copy in state q to one of the successors of the node. The acceptance condition of the automaton is $\emptyset$, guaranteeing that eventually the automaton reads a node in which p holds.

THEOREM 4.1. *Given a CTL formula ψ and a set $\mathfrak{D} \subset \mathbb{N}$, we can construct in linear running time a WAA $\mathcal{A}_{\mathfrak{D},\psi} = \langle 2^{AP}, \mathfrak{D}, cl(\psi), \delta, \psi, F \rangle$ such that $\mathcal{L}(\mathcal{A}_{\mathfrak{D},\psi})$ is exactly the set of $\mathfrak{D}$-trees satisfying ψ.*

PROOF. The set F of accepting states consists of all the $\tilde{U}$-formulas in $cl(\psi)$. It remains to define the transition function δ. For all $\sigma \in 2^{AP}$ and $k \in \mathfrak{D}$, we define:

[2] The translation described in Vardi and Wolper [1986b] handles PDL formulas, but can be easily adjusted to CTL.

—$\delta(p, \sigma, k) = \textbf{true}$ if $p \in \sigma$.

—$\delta(\mathrm{p}, \sigma, \mathrm{k}) = \textbf{false}$ if $p \notin \sigma$.

—$\delta(\neg p, \sigma, k) = \textbf{true}$ if $p \notin \sigma$.

—$\delta(\neg \mathrm{p}, \sigma, \mathrm{k}) = \textbf{false}$ if $p \in \sigma$.

—$\delta(\varphi_1 \wedge \varphi_2, \sigma, k) = \delta(\varphi_1, \sigma, k) \wedge \delta(\varphi_2, \sigma, k)$.

—$\delta(\varphi_1 \vee \varphi_2, \sigma, k) = \delta(\varphi_1, \sigma, k) \vee \delta(\varphi_2, \sigma, k)$.

—$\delta(AX\varphi, \sigma, k) = \bigwedge_{c=0}^{k-1}(c, \varphi)$.

—$\delta(EX\varphi, \sigma, k) = \bigvee_{c=0}^{k-1}(c, \varphi)$.

—$\delta(A\varphi_1 U \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \vee (\delta(\varphi_1, \sigma, k) \wedge \bigwedge_{c=0}^{k-1}(c, A\varphi_1 U \varphi_2))$.

—$\delta(E\varphi_1 U \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \vee (\delta(\varphi_1, \sigma, k) \wedge \bigvee_{c=0}^{k-1}(c, E\varphi_1 U \varphi_2))$.

—$\delta(A\varphi_1 \tilde{U} \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \wedge (\delta(\varphi_1, \sigma, k) \vee \bigwedge_{c=0}^{k-1}(c, A\varphi_1 \tilde{U} \varphi_2))$.

—$\delta(E\varphi_1 \tilde{U} \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \wedge (\delta(\varphi_1, \sigma, k) \vee \bigvee_{c=0}^{k-1}(c, E\varphi_1 \tilde{U} \varphi_2))$.

The weakness partition and order of $\mathcal{A}_{\mathcal{D},\psi}$ are defined as follows: Each formula $\varphi \in cl(\psi)$ constitutes a (singleton) set $\{\varphi\}$ in the partition. The partial order is then defined by $\{\varphi_1\} \leq \{\varphi_2\}$ iff $\varphi_1 \in cl(\varphi_2)$. Since each transition of the automaton from a state φ leads to states associated with formulas in $cl(\varphi)$, the weakness conditions hold. In particular, each set is either contained in F or disjoint from F.

We now prove the correctness of our construction, namely, that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ contains exactly all the $\mathcal{D}$-trees that satisfy ψ. We first prove that $\mathcal{A}_{\mathcal{D},\psi}$ is sound. That is, given an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ over a tree $\langle T_K, V_K\rangle$, we prove that for every $y \in T_r$ such that $r(y) = (x, \varphi)$, we have $V_K(x) \models \varphi$. Thus, in particular, $V_K(\epsilon) \models \psi$. The proof proceeds by induction on the structure of φ. The case where φ is an atomic proposition is immediate and the cases where φ is $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $AX\varphi_1$, or $EX\varphi_1$ follow easily, by the induction hypothesis, from the definition of δ. Less immediate are the cases where φ is an U-formula or a $\tilde{U}$-formula. Consider first the case where φ is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$. As $\langle T_r, r\rangle$ is an accepting run, it visits the state φ only finitely often. Since $\mathcal{A}_{\mathcal{D},\psi}$ keeps inheriting φ as long as φ_2 is not satisfied, then it is guaranteed, by the definition of δ and the induction hypothesis, that along all paths or some path, as required in φ, φ_2 does eventually holds and φ_1 holds until then. Consider now the case where φ is of the form $A\varphi_1 \tilde{U} \varphi_2$ or $E\varphi_1 \tilde{U} \varphi_2$. Here, it is guaranteed, by the definition of δ and the induction hypothesis, that φ_2 holds either always or until both φ_2 and φ_1 hold.

We now prove that $\mathcal{A}_{\mathcal{D},\psi}$ is complete. That is, given a $\mathcal{D}$-tree $\langle T_K, V_K\rangle$ such that $\langle T_K, V_K\rangle \models \psi$, we prove that $\mathcal{A}_{\mathcal{D},\psi}$ accepts $\langle T_K, V_K\rangle$. In fact, we show that there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_K, V_K\rangle$. We define $\langle T_r, r\rangle$ as follows: The run starts at the initial state; thus $\epsilon \in T_r$ and $r(\epsilon) = (\epsilon, \psi)$. The run proceeds maintaining the invariant that for all $y \in T_r$ with $r(y) = (x, \varphi)$, we have $V_K(x) \models \varphi$. Since $\langle T_K, V_K\rangle \models \psi$, the invariant holds for $y = \epsilon$. Also, by the semantic of CTL and the definition of δ, the run can always proceed such that all the successors $y \cdot c$ of a node y that satisfies the invariant have $r(y \cdot c) = (x', \varphi')$ with $V_K(x') \models \varphi'$. Finally, the run always try to satisfy eventualities of U-formulas. Thus, whenever φ is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$ and $V_K(x) \models \varphi_2$, it proceeds according to $\delta(\varphi_2, V_K(x), d(x))$. It is easy to see that all the paths in such $\langle T_r, r\rangle$ are either finite or reach a state associated with a $\tilde{U}$-formula and stay there thereafter. Thus, $\langle T_r, r\rangle$ is accepting. □

By the *sufficient degree property* [Emerson and Sistla 1984], a CTL formula ψ is satisfiable if and only if it is satisfied in an $\{n\}$-tree, where n is the number of occurrences of the path quantifier E in ψ. Hence, satisfiability of ψ can be reduced to the nonemptiness of $\mathcal{A}_{\{n\},\psi}$. As the nonemptiness problem for WAAs is in exponential time [Muller et al. 1986], the above described WAAs provide also an exponential-time satisfiability procedure for CTL.

Example 4.2. Consider the CTL formula $\psi = A(\mathbf{true}U(A\mathbf{false}\tilde{U}p))$. For every $\mathcal{D} \subset \mathbb{N}$, the WAA associated with ψ is $\mathcal{A}_{\mathcal{D},\psi} = \langle\{\{p\}, \emptyset\}, \mathcal{D}, \{\psi, A\mathbf{false}\tilde{U}p\}, \delta, \psi, \{A\mathbf{false}\tilde{U}p\}\rangle$, where δ is described in the following table (we restrict $\mathcal{A}_{\mathcal{D},\psi}$ to the reachable states).

state q	$\delta(q, \{p\}, k)$	$\delta(q, \emptyset, k)$
ψ	$\bigwedge_{c=0}^{k-1}(c, A\mathrm{false}\tilde{U}p) \vee \bigwedge_{c=0}^{k-1}(c, \psi)$	$\bigwedge_{c=0}^{k-1}(c, \psi)$
$A\mathrm{false}\tilde{U}p$	$\bigwedge_{c=0}^{k-1}(c, A\mathrm{false}\tilde{U}p)$	**false**

In the state ψ, if p holds in the present, then $\mathcal{A}_{\mathcal{D},\psi}$ may either guess that $A\mathbf{false}\tilde{U}p$, the eventuality of ψ, is satisfied in the present, or proceed with $\bigwedge_{c=0}^{k-1}(c, \psi)$, which means that the requirement for fulfilling the eventuality of ψ is postponed to the future. The crucial point is that since $\psi \notin F$, infinite postponing is impossible. In the state $A\mathbf{false}\tilde{U}p$, $\mathcal{A}_{\mathcal{D},\psi}$ expects a tree in which p is always true in all paths. Then, it keeps visiting $A\mathbf{false}\tilde{U}p$ forever. Since $A\mathbf{false}\tilde{U}p \in F$, this is permitted.

Example 4.3. Consider the CTL formula $\psi = A((EX\neg p)Ub)$. For every $\mathcal{D} \subset \mathbb{N}$, the WAA associated with ψ is $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{\{p,b\}}, \mathcal{D}, \{\psi, \neg p\}, \delta, \psi, \emptyset\rangle$, where δ is described in the following table (we restrict $\mathcal{A}_{\mathcal{D},\psi}$ to its reachable states).

state q	$\delta(q, \{p, b\}, k)$	$\delta(q, \{p\}, k)$	$\delta(q, \{b\}, k)$	$\delta(q, \emptyset, k)$
ψ	**true**	$\bigvee_{c=0}^{k-1}(c, \neg p) \wedge \bigwedge_{c=0}^{k-1}(c, \psi)$	**true**	$\bigvee_{c=0}^{k-1}(c, \neg p) \wedge \bigwedge_{c=0}^{k-1}(c, \psi)$
$\neg p$	**false**	**false**	**true**	**true**

In the state ψ, if b does not hold on the present, then $\mathcal{A}_{\mathcal{D},\psi}$ requires both $EX\neg p$ to be satisfied in the present (i.e., $\neg p$ to be satisfied in some successor), and ψ to be satisfied by all the successors. As $\psi \notin F$, $\mathcal{A}_{\mathcal{D},\psi}$ should eventually reach a node that satisfies b.

We now present a similar translation for the alternation-free μ-calculus.

THEOREM 4.4. *Given an alternation-free guarded μ-calculus formula ψ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct in linear running time a WAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), \delta, \psi, F\rangle$, such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

PROOF. For atomic propositions constants and for formulas of the forms $\varphi_1 \wedge \varphi_2$, $\varphi_1 \wedge \varphi_2$, $AX\varphi$, or $EX\varphi$, the transition function δ is equal to the one described for CTL. For μ and ν formulas, and for all $\sigma \in \Sigma$ and $k \in \mathcal{D}$, we define:

—$\delta(\mu y.f(y), \sigma, k) = \delta(f(\mu y.f(y)), \sigma, k)$.

—$\delta(\nu y.f(y), \sigma, k) = \delta(f(\nu y.f(y)), \sigma, k)$.

Note that since ψ is guarded, it is guaranteed that we have no circularity in the definition of δ. In order to define F, we define an equivalence relation $\mathcal{R}$ over $cl(\varphi)$ where

$$\varphi_1 \mathcal{R} \varphi_2 \text{ iff } \varphi_1 \in cl(\varphi_2) \text{ and } \varphi_2 \in cl(\varphi_1).$$

Since ψ is alternation free, it is guaranteed that an equivalence class of $\mathcal{R}$ cannot contain both a ν-formula and a μ-formula. A state $\varphi \in cl(\psi)$ belongs to F if and only if it belongs to an equivalence class that contains a ν-formula.

The weakness partition and order of $\mathcal{A}_{\mathcal{D},\psi}$ are induced by $\mathcal{R}$ as follows: Each equivalence class of $\mathcal{R}$ constitutes a set Q_i. We denote each set Q_i by $[\varphi]$, for some $\varphi \in Q_i$. The partial order is defined by $[\varphi_1] \leq [\varphi_2]$ iff $\varphi_1 \in cl(\varphi_2)$. As in CTL, since each transition of the automaton from a state φ leads to states associated with formulas in $cl(\varphi)$, the weakness conditions hold. In particular, each set is either contained in F or disjoint from F.

The correctness proof of the construction is similar to the one for CTL. Here, the definition of F guarantees that an accepting run cannot get trapped in a set with a μ-formula, and, on the other hand, it is allowed to stay forever in a set with a ν-formula. We describe here the soundness and completeness for the formula $\mu z.f(z)$. Recall that $\mu z.f(z)$ is equivalent to the formula $f(\mu z.f(z))$. Thus, proceeding according to $f(\mu z.f(z))$ is consistent with the semantics of μ-calculus. Assume that $\langle T_r, r\rangle$ is an accepting run over a tree $\langle T_K, V_K\rangle$. We prove that for every $\varphi \in cl(\psi)$ and for every $y \in T_K$ such that $r(y) = (x, \varphi)$, we have $V_K(x) \models \varphi$. As in CTL, the proof proceeds by induction on the structure of φ. Let $\varphi = \mu z.f(z)$. Since r is an accepting run, it visits the set $[\varphi]$ only finitely often. The only possibility of r to escape from the set $[\varphi]$ is to proceed to states ξ that appear in $\delta(f(\varphi), \sigma, k))$ and for which $\varphi \notin cl(\xi)$. For such ξ, we can employ the induction hypothesis, which implies, by the semantics of μ-calculus and the definition of δ, that $V_K(x) \models \varphi$. For the completeness, we show that if $\langle T_K, V_K\rangle \models \psi$, then there exists an accepting run of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_K, V_K\rangle$. As in CTL, we can define a run $\langle T_r, r\rangle$ such that for all $y \in T_r$ with $r(y) = (x, \varphi)$, we have $V_K(x) \models \varphi$. Consider a node $y \in T_r$ with $r(y) = (x, \mu z.f(z))$. Let $\varphi = \mu z.f(z)$. If $V_K(x)$ satisfies φ by satisfying a subformula ξ of $f(\mu z.f(z))$ for which $\varphi \notin cl(\xi)$, we say that y is an escape node. By the semantics of μ-calculus and the definition of δ, all the nodes $y \in T_r$ with $r(y) = (x, \mu z.f(z))$ eventually reach an escape node. Hence, the run $\langle T_r, r\rangle$ can proceed to states associated with the corresponding subformulas ξ and avoids $[\varphi]$-cycles. □

Example 4.5. Consider the formula $\psi = \mu y.(p \vee EXAXy)$. For every $\mathcal{D} \subset \mathbb{N}$, the WAA associated with ψ and $\mathcal{D}$ is $\mathcal{A}_{\mathcal{D},\psi} = \langle\{\{\mathrm{p}\}, \emptyset\}, \mathcal{D}, \{\psi, \mathrm{AX}\psi\}, \delta, \psi, \emptyset\rangle$, where δ is described below (we restrict $\mathcal{A}_{\mathcal{D},\psi}$ to its reachable states).

By definition, $\delta(\mu y.(p \vee EXAXy), \sigma, k) = \delta(p \vee EXAX\mu y.(p \vee EXAXy), \sigma, k)$. Hence, we have:

state q	$\delta(q, \{p\}, k)$	$\delta(q, \emptyset, k)$
ψ	**true**	$\bigvee_{c=0}^{k-1}(c, AX\psi)$
$AX\psi$	$\bigwedge_{c=0}^{k-1}(c, \psi)$	$\bigwedge_{c=0}^{k-1}(c, \psi)$

In the state ψ, if p does not hold in the present, $AX\psi$ should be satisfied in some successor. Since the state set of $\mathcal{A}_{\mathcal{D},\psi}$ constitutes a single rejecting set, p should eventually hold.

We now turn to study the complexity of the nonemptiness problem for WAAs. We first consider the general nonemptiness problem for them.

THEOREM 4.6. *The nonemptiness problem for weak alternating automata is EXPTIME-complete.*

PROOF. Membership in EXPTIME is proved in Muller et al. [1986]. Hardness in EXPTIME follows from reduction of satisfiability of CTL, proved to be EXPTIME-hard in Fischer and Ladner [1979]. □

Thus, the general nonemptiness problem for WAAs, the one required for solving the satisfiability problem, cannot be solved efficiently. For model checking, we do not have to solve the general nonemptiness problem. Taking the product with the Kripke structure, we get a 1-letter WAA over words. As we prove below, the nonemptiness problem for these automata can be solved in linear time. We note that the general nonemptiness problem for WAA over words is PSPACE-complete [Miyano and Hayashi 1984]. Thus, the transition to a 1-letter automaton is essential. Also, as follows from Theorem 3.1, the best upper-bound known for 1-letter nonemptiness of Büchi alternating word automata is quadratic [Vardi and Wolper 1986b]. Thus, the weakness of the automaton is also essential.

THEOREM 4.7. *The* 1*-letter nonemptiness problem for weak alternating word automata is decidable in linear running time.*

PROOF. Following Theorem 3.1, we prove that the 1-letter nonemptiness problem for simple weak alternating word automata is decidable in linear running time. We present an algorithm with linear running time for checking the nonemptiness of the language of a simple weak alternating word automaton $\mathcal{A} = \langle \{a\}, Q, \delta, Q^0, \alpha \rangle$.

The algorithm labels the states of $\mathcal{A}$ with either 'T', standing for **true**, or 'F', standing for **false**. The intuition is that states $q \in Q$ for which the language of $\mathcal{A}^q$ (i.e., the language of $\mathcal{A}$ with q as the initial state) is nonempty are labeled with 'T' and states q for which the language of $\mathcal{A}^q$ is empty are labeled with 'F'. The language of $\mathcal{A}$ is thus nonempty if and only if the initial state q_0 is labeled with 'T'.

As $\mathcal{A}$ is weak, there exists a partition of Q into disjoint sets Q_i such that there exists a partial order $\leq$ on the collection of the Q_i's and such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, a)$, we have that $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. In addition, each set Q_i is classified as accepting, if $Q_i \subseteq \alpha$, or rejecting, if $Q_i \cap \alpha = \emptyset$. Following Theorem 2.2, if the partition of Q is not given, one can find such a partition in linear time. The algorithm works in phases and proceeds up the partial order. We regard **true** and **false** as states with a self loop. The state **true** constitutes an accepting set and the state **false** constitutes a rejecting set, both minimal in the partial order. Let $Q_1 \leq \cdots \leq Q_n$ be an extension of the partial order to a total order. In each phase i, the algorithm handles states from the minimal set Q_i that still has not been labeled.

States that belong to the set Q_1 are labeled according to the classification of Q_1. Thus, they are labeled with 'T' if Q_1 is an accepting set and they are labeled with 'F' if it is a rejecting set. Once a state $q \in Q_i$ is labeled with 'T' or 'F', transition functions in which q occurs are simplified accordingly; that is, a conjunction with a conjunct 'F' is simplified to 'F' and a disjunction with a disjunct 'T' is simplified to 'T'. Consequently, a transition function $\delta(q', a)$ for some q' (not necessarily from Q_i) can be simplified to **true** or **false**. The state q' is then labeled, and simplification propagates further.

Since the algorithm proceeds up the total order, when it reaches a state $q \in Q_i$ that is still not labeled, it is guaranteed that all the states in all Q_j for which $Q_j < Q_i$, have already been labeled. Hence, all the states that occur in $\delta(q, a)$ have the same status as q. That is, they belong to Q_i and are still not labeled. The algorithm then labels q and all the states in $\delta(q, a)$ according to the classification of Q_i. They are labeled 'T' if Q_i is accepting and are labeled 'F' otherwise.

Correct operation of the algorithm can be understood as follows: It is guaranteed that once the automaton visits a state that belongs to Q_1, it visits only states from Q_1 thereafter. Similarly, when the automaton visits a state q whose labeling cannot be decided according to labeling of states in lower sets, this state leads to a cycle or belongs to a cycle of states of the same status, so the labeling of states according to the classification of the set to which they belong.

Formally, we prove that for all $1 \leq i \leq n$, all the states in Q_i are labeled correctly. The proof proceeds by induction on i. The case $i = 1$ is immediate. Assume that we have already labeled correctly all the states in all Q_j with $j < i$ and let $q \in Q_i$. We consider the case where Q_i is an accepting set. The proof is symmetric for the case where Q_i is a rejecting set. We distinguish between three possibilities of labeling q:

(1) The state q is labeled 'T' before the phase i. Then, the value of $\delta(q, a)$, simplified according to the labeling already done, is **true**. Therefore, there exists a run of $\mathcal{A}^q$ in which every copy created in the first step (i.e., every copy that is created in order to satisfy $\delta(q, a)$) reaches a state q' for which, by the induction hypothesis, the language of $\mathcal{A}^{q'}$ is not empty. Hence, the language of $\mathcal{A}^q$ is also not empty.

(2) The state q is labeled 'F' before the phase i. The correctness proof is symmetric to the one of the previous case: The value of $\delta(q, a)$, simplified according to the labeling already done, is **false**. Therefore, every run of $\mathcal{A}^q$ has at least one copy created in the first step and reaches a state q' for which, by the induction hypothesis, the language of $\mathcal{A}^{q'}$ is empty. Hence, the language of $\mathcal{A}^q$ is also empty.

(3) The state q is labeled 'T' during the phase i. Then, it must be the case that the simplification of $\delta(q, a)$ contains states of Q_i. Moreover, it contains only states of Q_i and they all have not been labeled before the phase i. Thus, there exists a run of $\mathcal{A}^q$ in which every copy created in the first step either

reaches a state q' for which the language of $\mathscr{A}^{q'}$ is not empty, or stays forever in Q_i. Hence, the language of $\mathscr{A}^q$ is not empty.

Note that these are indeed the only possibilities of labeling q: a state in an accepting set Q_i cannot be labeled after the phase i and it cannot be labeled with 'F' during the phase i since we are dealing with an accepting Q_i.

Using an AND/OR graph, as suggested in Beeri and Bernstein [1979], Beeri [1980], and Dowling and Gallier [1984], the algorithm can be implemented in linear running time. The graph, G, induced by the transition function, maintains the labeling and the propagation of labeling performed during the algorithm execution. In more details, each node of G corresponds to a state $q \in Q$. For q with $\delta(q, a) = q_1 * q_2$, the node q is a $*$-node with two successors, q_1 and q_2. For $q = \{$**true, false**$\}$, the node q is a sink-node. Since $\mathscr{A}$ is simple, these are the only possible forms of transitions and hence the only possible nodes. Each $*$-node q is labeled by a triple $\langle *, \textit{last,ptrs} \rangle$, where *last* is a Boolean flag and *ptrs* is a list of pointers that point to nodes that have q as a successor. The Boolean flag *last* is true iff one of the two successors of q has already been labeled with 'T' or 'F', its labeling has been propagated to q, but did not suffice to label q as well. In the beginning, *last* = **false** for all the nodes. Each sink-node q is labeled by a set *ptrs* of pointers that point to nodes that have q as a successor. It is easy to see that the size of G is linear in the size of δ, and it can be constructed in linear running time.

In addition, the algorithm maintains an integer i that contains the current phase and two stacks S_T and S_F. The stacks contain nodes that were labeled with 'T' and 'F', yet still have not propagated their labeling further. In the beginning, $i = 1$, the stack S_T contains the sink-node **true** (if exists), and the stack S_F contains the sink-node **false** (if exists).

Using G, the algorithm proceeds as follows: Whenever a node in the graph is labeled with 'T' ('F'), the node is pushed into S_{T} (S_{F}). As long as S_{T} or S_{F} are not empty, some node q is popped from either stacks, and labeling is propagated to every node q' that has q as a successor (as detected by *ptrs*). In the case $q \in S_{\mathrm{T}}$, the node q' is labeled as follows:

—If q' is a $\bigvee$-node, then label q' with 'T'.

—If q' is a $\bigwedge$-node with *last* = **true**, then label q' with 'T'.

—If q' is a $\bigwedge$-node with *last* = **false**, then change *last* to **true**.

In the case $q \in S_{\mathrm{F}}$, the node q' is labeled as follows:

—If q' is a $\bigwedge$-node, then label q' with 'F'.

—If q' is a $\bigvee$-node with *last* = **true**, then label q' with 'F'.

—If q' is a $\bigvee$-node with *last* = **false**, then change *last* to **true**.

When both S_{T} and S_{F} are empty, nodes that correspond to the states of the current Q_i are labeled according to the classification of Q_i and i is increased. Since each node of G is pushed into a stack only once, and since handling of a node that is popped from a stack involves a constant number of operations to each of the nodes that have it as a successor, the entire complexity is linear in the size of G; hence, linear in the size of δ. □

Theorems 4.1, 4.4, and 4.7, together with Proposition 3.2, yield model-checking algorithms for CTL and for the alternation-free μ-calculus with running time $O(\|K\|^2 \cdot \|\psi\|)$. Indeed, the size of the automaton $\mathcal{A}_{\mathcal{D},\psi}$ is linear in both $|\mathcal{D}|$ and $\|\psi\|$, the tightest bound for $|\mathcal{D}|$ is the number of states in K, and thus the size of $\mathcal{A}_{K,\psi}$, as guaranteed from Proposition 3.2, is quadratic in $\|K\|$ and linear in $\|\psi\|$. Nevertheless, a closer look at $\mathcal{A}_{K,\psi}$, shows that its size is only linear in both $\|K\|$ and $\|\psi\|$, and hence, so is the running time of our algorithm. To see this, consider the automaton $\mathcal{A}_{\mathcal{D},\psi}$. The $|\mathcal{D}|$ factor in the size of $\mathcal{A}_{\mathcal{D},\psi}$ comes from the need to specify the transition function for all the branching degrees in $\mathcal{D}$. This is not the case in $\mathcal{A}_{K,\psi}$. There, every state is associated with a state in K. Therefore, every state is associated with a single branching degree. Hence, when we define $\mathcal{A}_{K,\psi}$, we need to specify for each of its states $\langle w, \varphi \rangle$ only the transition induced by $\delta(\varphi, L(w), d(w))$ in $\mathcal{A}_{\mathcal{D},\psi}$. It follows that the size of the transitions in $\mathcal{A}_{K,\psi}$ is bounded by $|R| \cdot \|\psi\|$, implying its linear size.

The algorithm used in the proof of Theorem 4.7 is clearly reminiscent of the bottom-up labeling that takes place in the standard algorithms for CTL and alternation-free μ-calculus model checking [Clarke et al. 1986; Cleaveland 1993]. Thus, the automata-theoretic approach seems to capture the combinatorial essence of branching-time model checking.

4.2. MODEL CHECKING FOR THE M-CALCULUS. The intimate connection between the μ-calculus and alternating automata has been noted in Jutla [1990], Emerson and Jutla [1991], Bhat and Cleaveland [1996b], and Emerson [1996]. We show here that our automata-theoretic approach provides a clean proof that model checking for the μ-calculus is in NP $\cap$ co-NP. The key steps in the proof are showing that μ-calculus formulas can be efficiently translated to alternating parity automata, and that the 1-letter nonemptiness problem for alternating Rabin word automata is in NP.

THEOREM 4.8. *Given a μ-calculus formula ψ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct, in linear running-time, an alternating parity automaton $\mathcal{A}_{\mathcal{D},\psi}$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

PROOF. To build an alternating automaton from a μ-calculus formula, one is naturally tempted to use the same transition relation as for the alternation-free μ-calculus. The only problem with this is that it does not allow the necessary acceptance conditions to be defined. Indeed, if one looks carefully at the transition relation obtained for a CTL or an alternation-free μ-calculus formula ψ, not all elements of $cl(\psi)$ are actually used as states. Specifically, a Boolean combination can appear as a state without its constituents also appearing. This makes it impossible to express an acceptance condition involving a formula appearing only as a constituent of a Boolean state. In the case of CTL, this was of no consequence since the acceptance condition involves exclusively $\tilde{U}$ formulas and these do by construction appear as states. Similarly, for the alternation-free μ calculus, the problem was worked around by using the absence of alternation to define equivalence classes of formulas in such a way that each class contains at least one formula that is a state of the automaton. For the full μ-calculus, such short cuts are not possible and we thus need to avoid Boolean combinations as states (except for the initial state) and force states to be of the form p, $\neg p$, $AX\varphi$, $EX\varphi$, $\mu y.f(y)$, or $\nu y.f(y)$. This is the purpose of the function *split* introduced below.

For a μ-calculus formula ψ, we define the function *split*: $\mathcal{B}^+(\mathbb{N} \times cl(\psi)) \rightarrow \mathcal{B}^+(\mathbb{N} \times cl(\psi))$ as follows:

—*split*(**true**) = **true** and *split*(**false**) = **false**.

—$split(\theta_1 \wedge \theta_2) = split(\theta_1) \wedge split(\theta_2)$.

—$split(\theta_1 \vee \theta_2) = split(\theta_1) \vee split(\theta_2)$.

—For φ of the form p, $\neg p$, $EX\varphi'$, $AX\varphi'$, $\mu y.f(y)$, or $\nu y.f(y)$, we have $split((c, \varphi)) = (c, \varphi)$.

—$split((c, \varphi_1 \wedge \varphi_2)) = split((c, \varphi_1)) \wedge split((c, \varphi_2))$.

—$split((c, \varphi_1 \vee \varphi_2)) = split((c, \varphi_1)) \vee split((c, \varphi_2))$.

For example, $split((0, \varphi_1 \wedge \varphi_2) \wedge (1, EX(\varphi_3 \wedge \varphi_4))) = (0, \varphi_1) \wedge (0, \varphi_2) \wedge (1, EX(\varphi_3 \wedge \varphi_4))$. Note that $split(\theta)$ contains no atoms of the form $(c, \varphi_1 \wedge \varphi_2)$ or $(c, \varphi_1 \vee \varphi_2)$.

For a μ-calculus sentence ψ and a subformula $\varphi = \lambda y.f(y)$ of ψ, we define the *alternation level* of φ in ψ, denoted $al_\psi(\varphi)$, as follows [Bhat and Cleaveland 1996a]:

—If φ is a sentence, then $al_\psi(\varphi) = 1$.

—Otherwise, let $\xi = \lambda' x.g(x)$ be the innermost μ or ν subformula of ψ that has φ as a strict subformula. Then, if x is free in φ and $\lambda' \neq \lambda$, we have $al_\psi(\varphi) = al_\psi(\xi) + 1$. Otherwise, we have $al_\psi(\varphi) = al_\psi(\xi)$.

Intuitively, the alternation level of φ in ψ is the number of alternating fixed-point operators we have to "wrap φ with" in order to reach a subsentence of ψ.

Given ψ and $\mathcal{D}$, we define the parity automaton $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), \delta, \psi, F \rangle$, where

—The transition function δ is exactly as in the automata for the alternation-free μ-calculus, except for splitting the right hand side of the transitions. For all $\sigma \in 2^{AP}$ and $k \in \mathcal{D}$, we define:

—$\delta(p, \sigma, k) =$ **true** if $p \in \sigma$.

—$\delta(\mathrm{p}, \sigma, \mathrm{k}) =$ **false** if $p \notin \sigma$.

—$\delta(\neg p, \sigma, k) =$ **true** if $p \notin \sigma$.

—$\delta(\neg \mathrm{p}, \sigma, \mathrm{k}) =$ **false** if $p \in \sigma$.

—$\delta(\varphi_1 \wedge \varphi_2, \sigma, k) = split(\delta(\varphi_1, \sigma, k) \wedge \delta(\varphi_2, \sigma, k))$.

—$\delta(\varphi_1 \vee \varphi_2, \sigma, k) = split(\delta(\varphi_1, \sigma, k) \vee \delta(\varphi_2, \sigma, k))$.

—$\delta(AX\varphi, \sigma, k) = split(\bigwedge_{c=0}^{k-1}(c, \varphi))$.

—$\delta(EX\varphi, \sigma, k) = split(\bigvee_{c=0}^{k-1}(c, \varphi))$.

—$\delta(\mu y.f(y), \sigma, k) = split(\delta(f(\mu y.f(y)), \sigma, k))$.

—$\delta(\nu y.f(y), \sigma, k) = split(\delta(f(\nu y.f(y)), \sigma, k))$.

—Our transition relation is very similar to the one suggested in Emerson and Jutla [1991]. Splitting the formulas in the right-hand side avoids the epsilon-transitions implicitly assumed there, and explicitly given in the proof rules in

Bhat and Cleaveland [1996a]. Indeed, the split guarantees that when the automaton is tracing a fixed-point formula φ, it keeps visiting the state φ itself.

—We first define the acceptance condition in terms of a Rabin acceptance condition, as in Bhat and Cleaveland [1996a] (which dualizes the Streett condition in Emerson and Jutla [1991]). Let d be the maximal alternation level of subformulas of ψ. Denote by G_i the set of all the ν-formulas in $cl(\psi)$ of alternation level i. Denote by B_i the set of all μ-formulas in $cl(\psi)$ of alternation depth less than or equal to i. The Rabin condition is $F = \bigcup_{i \in 1 \ldots d} \{\langle G_i, B_i \rangle\}$. That is, if the automaton gets stuck in a cycle, it must visit some ν-formula infinitely often and can visit μ-formulas of smaller alternation levels only finitely often. Now, let $F_0 = \emptyset$, and for every $1 \leq i \leq d$, let $F_{2i-1} = F_{2i-2} \cup B_i$ and $F_{2i} = F_{2i-1} \cup G_i$. It is easy to see that $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_{2d}$ and that the parity condition $\{F_1, F_2, \ldots, F_{2d}\}$ is equivalent to the Rabin condition F. □

THEOREM 4.9. *The* 1-*letter nonemptiness problem for alternating parity word automata is decidable in nondeterministic polynomial running time*.

PROOF. According to Theorem 3.1, the 1-letter nonemptiness problem for alternating parity word automata is of the same complexity as the nonemptiness problem for nondeterministic parity tree automata. By Emerson [1985] and Vardi and Stockmeyer [1985], the nonemptiness problem for nondeterministic Rabin tree automata, which generalizes parity tree automata, is in NP. □

Combining Theorems 4.8 and 4.9, Proposition 3.2, and the observation in Emerson et al. [1993] that checking for satisfaction of a formula ψ and a formula $\neg\psi$ has the same complexity, we get that the model-checking problem for the μ-calculus is in NP $\cap$ co-NP. Also, since the nonemptiness problem for a parity tree automaton with n states and index k can be solved in time $O(n^k)$ [Emerson et al. 1993; Kupferman and Vardi 1998c], the construction in Theorem 4.8 also implies an $O((\|K\| \cdot \|\psi\|)^{\|\psi\|})$ algorithm for the model-checking problem of μ-calculus.

5. *The Space Complexity of Model Checking*

Lichtenstein and Pnueli [1985] argued that when analyzing the complexity of model checking, a distinction should be made between complexity in the size of the input structure and complexity in the size of the input formula; it is the complexity in the size of the structure that is typically the computational bottleneck.[3] The Kripke structures to which model-checking is applied are often obtained by constructing the reachability graph of *concurrent programs*, and can thus be very large. So, even linear complexity, in terms of the input structure, can be excessive, especially as far as space is concerned. The question is then whether it is possible to perform model-checking without ever holding the whole structure to be checked in memory at any one time. For linear temporal formulas, the answer has long been known to be positive [Vardi and Wolper 1986a]. Indeed, this problem reduces to checking the emptiness of a Büchi automaton over words, which is NLOGSPACE-complete. Thus, if the Büchi automaton whose

[3] For a similar distinction in database query evaluation, see Vardi [1982].

emptiness has to be checked is obtained as the product of the components of a concurrent program (as is usually the case), the space required is polynomial in the size of these components rather than of the order of the exponentially larger Büchi automaton. Pragmatically, this is very significant and is, to some extent, exploited in the "on the fly" approaches to model checking and in related memory saving techniques [Courcoubetis et al. 1992; Mihail and Papadimitriou 1994].

Is the same true of branching-time model-checking? The answer to this question was long thought to be negative. Indeed, the bottom-up nature of the known model-checking algorithms seemed to imply that storing the whole structure was required. Using our automata-theoretic approach to branching-time model-checking, we are able to show that this is not necessarily so. In this section, we introduce a new type of alternating automata, called *hesitant alternating automata* (HAAs), for which the 1-letter nonemptiness problem can be solved with a very efficient use of space. We show that formulas of CTL and CTL* can be translated to HAAs and that the model-checking problem for these logics can be solved in space polynomial in $m \log n$, where m is the length of the formula and n is the size of the Kripke structure. Hence, the model-checking problem for concurrent programs for these logics is in PSPACE. We also show that this bound is tight. We claim that the ability to translate formulas to HAAs is of a great importance when space complexity of model checking is considered. For example, formulas of the alternation-free μ-calculus cannot be translated to HAAs and the model-checking problem for concurrent programs for this logic is EXPTIME-complete.

5.1. HESITANT ALTERNATING AUTOMATA. Consider the product automaton $\mathcal{A}_{K,\psi} = K \times \mathcal{A}_{\mathfrak{D},\psi}$ for a Kripke structure K and a CTL formula ψ. The states of $\mathcal{A}_{K,\psi}$ are elements of $W \times cl(\psi)$ and they are partitioned into sets Q_i according to their second component (two states are in the same Q_i if and only if their second components are identical). Thus, the number of Q_i's is bounded by the size of $cl(\psi)$ and is independent of the size of the Kripke structure. If we examine the Q_i's closely, we notice that they all fall into one of the following three categories:

(1) Sets from which all transitions lead exclusively to states in lower Q_i's. These are the Q_i's corresponding to all elements of $cl(\psi)$ except U-formulas and $\tilde{U}$-formulas.
(2) Sets Q_i such that, for all $q \in Q_i$, the transition $\delta(q, a, k)$ only contains *disjunctively related* elements of Q_i (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of Q_i in each disjunct). These are the Q_i's corresponding to the $E\varphi_1 U \varphi_2$ and $E\varphi_1 \tilde{U} \varphi_2$ elements of $cl(\psi)$.
(3) Sets Q_i such that, for all $q \in Q_i$, the transition $\delta(q, a, k)$ only contains *conjunctively related* elements of Q_i (i.e., if the transition is rewritten in conjunctive normal form, there is at most one element of Q_i in each conjunct). These are the Q_i's corresponding to the $A\varphi_1 U \varphi_2$ and $A\varphi_1 \tilde{U} \varphi_2$ elements of $cl(\psi)$.

This means that it is only when moving from one Q_i to the next, we can move from a state that is conjunctively related to states in its set to a state that is disjunctively related to states in its set, or vice-versa. In other words, when a copy of the automaton visits a state in some set Q_i which is associated with an EU-formula of an $E\tilde{U}$-formula, then as long as it stays in this set, it proceeds in an "existential mode"; namely, it imposes only existential requirements on its successors in Q_i. Similarly, when a copy of the automaton visits a state in some set Q_i which is associated with an AU-formula or an $A\tilde{U}$-formula, then as long as it stays in this set, it proceeds in a "universal mode". Thus, whenever a copy alternates modes, it must be that it moves from one Q_i to a lower one.

The above observation is captured in the restricted structure of hesitant alternating automata (HAAs), and is the key to our space-efficient model-checking procedure for CTL and CTL*. An HAA is an alternating automaton $\mathcal{A} = \langle \Sigma, \mathfrak{D}, Q, \delta, q_0, F \rangle$, where $F = \langle G, B \rangle$ with $G \subseteq Q$ and $B \subseteq Q$. That is, the acceptance condition of HAAs consists of a pair of sets of states. As in WAAs, there exists a partition of Q into disjoint sets and a partial order $\leq$ such that transitions from a state in Q_i lead to states in either the same Q_i or a lower one. In addition, each set Q_i is classified as either *transient, existential*, or *universal*, such that for each set Q_i and for all $q \in Q_i$, $\sigma \in \Sigma$, and $k \in \mathfrak{D}$, the following hold:

(1) If Q_i is a transient set, then $\delta(q, \sigma, k)$ contains no elements of Q_i.
(2) If Q_i is an existential set, then $\delta(q, \sigma, k)$ only contains disjunctively related elements of Q_i.
(3) If Q_i is a universal set, then $\delta(q, \sigma, k)$ only contains conjunctively related elements of Q_i.

It follows that every infinite path π of a run r gets trapped within some existential or universal set Q_i. The path then satisfies an acceptance condition $\langle G, B \rangle$ if and only if either Q_i is an existential set and $inf(\pi) \cap G \neq \emptyset$, or Q_i is a universal set and $inf(\pi) \cap B = \emptyset$. Note that the acceptance condition of HAAs combines the Rabin and the Streett acceptance conditions: Existential sets refer to a Rabin condition $\{\langle G, \emptyset \rangle\}$ and universal sets refer to a Street condition $\{\langle B, \emptyset \rangle\}$. Note also that while the transition function of HAAs is more restricted than the one of WAAs, their acceptance condition is more expressive. We will need the stronger acceptance condition to handle CTL* formulas. We call the partition of Q into sets the *hesitation partition* and we call the partial order over the sets the *hesitation order*. The length of the longest descending chain in the hesitation order is defined as the *depth* of the HAA. As with WAA (see Theorem 2.2), it is easy to see that the partition of the graph induced by an HAA to maximal strongly connected components refines any hesitation partition and can therefore serve as such partition. The reachability order on the MSCCs then induces the hesitation order and the depth of the HAA. Since the reachability problem for a directed graph is in NLOGSPACE, it follows that when a hesitation partition is not given, it is possible to refer to the partition into MSCCs and check, in NLOGSPACE, whether two vertices belong to the same set in the partition. In the HAA considered here, however, the hesitation partition and order are always known.

5.2. THE SPACE COMPLEXITY OF CTL AND CTL* MODEL CHECKING. In Theorem 4.1, we presented a translation of CTL formulas to WAAs. We have already shown that the resulting WAAs have the restricted structure of HAAs.

By showing that their acceptance condition can be put in the required form, we can establish Theorem 5.1 below.

THEOREM 5.1. *Given a CTL formula ψ and a set $\mathcal{D} \subset \mathbb{N}$, we can construct an HAA $\mathcal{A}_{\mathcal{D},\psi}$ of size $O(|\mathcal{D}| * \|\psi\|)$ and of depth $O(\|\psi\|)$ such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.*

PROOF. As observed above, each set in the WAA that correspond to CTL formulas is either transient, existential, or universal. Thus, we only have to define a suitable acceptance condition. In the WAA, we allow a path to get trapped in a set that corresponds to $\tilde{U}$-formulas. Accordingly, in HAA, we allow a path to get trapped in an existential set only if it corresponds to a $\tilde{U}$-formula and we allow a path to get trapped in a universal set only if it does not correspond to a U-formula. This is done with the acceptance condition $\langle G, B \rangle$ where G is the set of all $E\tilde{U}$-formulas in $cl(\psi)$ and B is the set of all AU-formulas in $cl(\psi)$. Since each set in the HAA corresponds to a single formula in $cl(\psi)$, the depth of the HAA is at most $\|\psi\|$. □

We now present a translation of CTL* formulas to HAAs. Weak alternating automata define exactly the set of weakly definable languages [Rabin 1970; Muller et al. 1986]. The logic CTL* can define languages that are not weakly definable. For example, the set of trees that satisfy the CTL* formula $AFGp$ is not weakly definable [Rabin 1970]. Therefore, a stronger acceptance condition is required for automata corresponding to formulas of CTL*. As we shall see later, the stronger acceptance condition does not raise the complexity of the 1-letter nonemptiness problem. We first show that complementation is easy for HAAs. For two HAA $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet Σ, we say that $\mathcal{A}_1$ is the *complement* of $\mathcal{A}_2$ iff $\mathcal{L}(\mathcal{A}_1)$ includes exactly all the Σ-labeled trees that are not in $\mathcal{L}(\mathcal{A}_2)$.

Given a transition function δ, let $\tilde{\delta}$ denote the dual function of δ. That is, for every q, σ, and k, with $\delta(q, \sigma, k) = \theta$, let $\tilde{\delta}(q, \sigma, k) = \tilde{\theta}$, where $\tilde{\theta}$ is obtained from θ by switching $\bigvee$ and $\bigwedge$ and by switching **true** and **false**. If, for example, $\theta = p \bigvee (\textbf{true} \bigwedge q)$, then $\tilde{\theta} = p \bigwedge (\textbf{false} \bigvee q)$,

LEMMA 5.2. *Given an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \langle G, B \rangle \rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_0, \langle B, G \rangle \rangle$ is an HAA that complements $\mathcal{A}$.*

PROOF. It is easy to see that $\tilde{\mathcal{A}}$ is an HAA. Indeed, the partition of Q into sets and the partial order over them hold also with respect to $\tilde{\mathcal{A}}$. In particular, a set that is existential in $\mathcal{A}$ is universal in $\tilde{\mathcal{A}}$ and vice versa. Consider an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, for some acceptance condition F, and consider the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_0, \tilde{F} \rangle$, where $\tilde{F}$ is such that for every path $\pi \in Q^{\omega}$, we have that π satisfies F in a run of $\mathcal{A}$ iff π does not satisfy $\tilde{F}$ in a run of $\tilde{\mathcal{A}}$. Muller and Schupp [1987] prove that $\tilde{\mathcal{A}}$ complements $\mathcal{A}$. We prove here that when $\mathcal{A}$ is a HAA with $F = \langle G, B \rangle$, then $\tilde{F} = \langle B, G \rangle$ satisfies the required property.

Consider a path π in a run of $\mathcal{A}$. By the definition of acceptance of HAA, we have that π satisfies an acceptance condition $\langle G, B \rangle$ iff either π gets trapped in an existential set with $inf(\pi) \cap G \neq \emptyset$, or π gets trapped in a universal set with $inf(\pi) \cap B = \emptyset$. Since π always gets trapped in either an existential or a universal set, it follows that π does not satisfy the acceptance condition $\langle G, B \rangle$

iff either π gets trapped in an existential set with $inf(\pi) \cap G = \emptyset$, or π gets trapped in a universal set with $inf(\pi) \cap B \neq \emptyset$. Dualizing δ, existential sets become universal and vice versa. Hence, π satisfies the acceptance condition $\langle G, B \rangle$ in a run of $\mathcal{A}$ iff it does not satisfy the acceptance condition $\langle B, G \rangle$ in a run of $\tilde{\mathcal{A}}$. □

For an HAA $\mathcal{A}$, we say that $\tilde{\mathcal{A}}$ is the *dual* HAA of $\mathcal{A}$.

THEOREM 5.3. *Given a CTL* formula* ψ *and a set* $\mathcal{D} \subset \mathbb{N}$, *we can construct an HAA* $\mathcal{A}_{\mathcal{D},\psi}$ *of size* $O(|\mathcal{D}| * 2^{\|\psi\|})$ *and of depth* $O(\|\psi\|)$ *such that* $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ *is exactly the set of* $\mathcal{D}$*-trees satisfying* ψ.

PROOF. Before defining $\mathcal{A}_{\mathcal{D},\psi}$ we need the following definitions and notations. For two CTL* formulas θ and φ, we say that θ is *maximal* in φ, if and only if θ is a strict state subformula of φ and there exists no state formula "between them", namely, there exists no strict subformula ξ of φ such that θ is a strict subformula of ξ. We denote by $max(\varphi)$ the set of all formulas maximal in φ. For example, $max(A((Xp)U(EXq))) = \{p, EXq\}$.

We construct $\mathcal{A}_{\mathcal{D},\psi}$ by induction on the structure of ψ. For technical convenience, we describe the definition for binary trees. The extension to any $\mathcal{D} \subset \mathbb{N}$ is straightforward. With each formula $\varphi \in cl(\psi)$, we associate an HAA $\mathcal{A}_\varphi$ composed from HAAs associated with formulas maximal in φ. We assume that the state sets of composed HAAs are disjoint (otherwise, we rename states) and that for all the HAAs we have $\Sigma = 2^{AP}$ (i.e., an HAA associated with a subformula that does not involve all of AP is extended in a straightforward way). For φ with $max(\varphi) = \{\varphi_1, \ldots, \varphi_n\}$ and for all $1 \leq i \leq n$, let $\mathcal{A}_{\varphi_i} = \langle \Sigma, Q^i, \delta^i, q_0^i, \langle G^i, B^i \rangle \rangle$ be the HAA associated with φ_i and let $\tilde{\mathcal{A}}_{\varphi_i} = \langle \Sigma, \tilde{Q}^i, \tilde{\delta}^i, \tilde{q}_0^i, \langle \tilde{G}^i, \tilde{B}^i \rangle \rangle$ be its dual HAA. We define $\mathcal{A}_\varphi$ as follows:

—If $\varphi = p$ or $\varphi = \neg p$ for some $p \in AP$, then $\mathcal{A}_\varphi$ is a one-state HAA.

—If $\varphi = \varphi_1 \wedge \varphi_2$, then $\mathcal{A}_\varphi = \langle \Sigma, Q^1 \cup Q^2 \cup \{q_0\}, \delta, q_0, \langle G^1 \cup G^2, B^1 \cup B^2 \rangle \rangle$, where q_0 is a new state and δ is defined as follows. For states in Q^1 and Q^2, the transition function δ agrees with δ^1 and δ^2. For the state q_0 and for all $\sigma \in \Sigma$, we have $\delta(q_0, \sigma) = \delta(q_0^1, \sigma) \wedge \delta(q_0^2, \sigma)$. Thus, in the state q_0, $\mathcal{A}_\varphi$ sends all the copies sent by both $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$. The singleton $\{q_0\}$ constitutes a transient set, with the ordering $\{q_0\} > Q_i$ for all the sets Q_i in Q^1 and Q^2. The construction for $\varphi = \varphi_1 \vee \varphi_2$ is similar, with $\delta(q_0, \sigma) = \delta(q_0^1, \sigma) \vee \delta(q_0^2, \sigma)$.

—If $\varphi = E\xi$, where ξ is a CTL* path formula, we first build an HAA $\mathcal{A}'_\varphi$ over the alphabet $\Sigma' = 2^{max(\varphi)}$. That is, $\mathcal{A}'_\varphi$ regards the formulas maximal in φ as atomic propositions. Let $\mathcal{U}_\xi = \langle \Sigma', Q, M, q_0, F \rangle$ be a nondeterministic Büchi automaton on infinite words such that $\mathcal{U}_\xi$ accepts exactly all the word models of ξ [Vardi and Wolper 1994], where the maximal subformulas are regarded as atomic propositions. Then, $\mathcal{A}'_\varphi = \langle \Sigma', Q, \delta', q_0, \langle F, \emptyset \rangle \rangle$ extends $\mathcal{U}_\xi$ to trees by simulating it along a single branch. That is, for all $q \in Q$ and $\sigma' \in \Sigma'$, we have

$$\delta'(q, \sigma') = \bigvee_{q_i \in M(q,\sigma')} (0, q_i) \vee (1, q_i).$$

If $M(q, \sigma') = \emptyset$, then $\delta'(q, \sigma') = \mathbf{false}$. Note that Q constitutes a single existential set. The HAA $\mathcal{A}'_\varphi$ accepts exactly all the Σ'-labeled tree models of φ.

We now adjust $\mathcal{A}'_\varphi$ to the alphabet Σ. The resulted automaton is $\mathcal{A}_\varphi$. Intuitively, $\mathcal{A}_\varphi$ starts additional copies of the HAAs associated with formulas in $max(\varphi)$. These copies guarantee that whenever $\mathcal{A}'_\varphi$ assumes that a formula in $max(\varphi)$ holds, then it indeed holds, and that whenever $\mathcal{A}'_\varphi$ assumes that a formula does not hold, then the negation of the formula holds. Formally, $\mathcal{A}_\varphi = \langle \Sigma, Q \cup \bigcup_i(Q^i \cup \tilde{Q}^i), \delta, q_0, \langle F \cup \bigcup_i(G^i \cup \tilde{G}^i), \bigcup_i(B^i \cup \tilde{B}^i)\rangle\rangle$, where δ is defined as follows. For states in $\bigcup_i(Q^i \cup \tilde{Q}^i)$, the transition function δ agrees with the corresponding δ^i and $\tilde{\delta}^i$. For $q \in Q$ and for all $\sigma \in \Sigma$, we have

$$\delta(q, \sigma) = \bigvee_{\sigma' \in \Sigma'} (\delta'(q, \sigma') \wedge (\bigwedge_{\varphi_i \in \sigma'} \delta^i(q_0^i, \sigma)) \wedge (\bigwedge_{\varphi_i \notin \sigma'} \tilde{\delta}^i(\tilde{q}_0^i, \sigma))).$$

Each conjunction in δ corresponds to a label $\sigma' \in \Sigma'$. Some copies of $\mathcal{A}_\varphi$ (these originated from $\delta'(q, \sigma')$) proceed as $\mathcal{A}'_\varphi$ when it reads σ'. Other copies guarantee that σ' indeed holds in the current node. The set Q constitutes an existential set, with the ordering $Q > Q'$ for all the sets Q' in $\bigcup_i(\{Q^i\} \cup \{\tilde{Q}^i\})$.

—If $\varphi = A\xi$, we construct and dualize the HAA of $E\neg\xi$.

We prove the correctness of the construction by induction on the structure of φ. The proof is immediate for the case φ is of the form p, $\neg p$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, or $A\xi$. We consider here the case where $\varphi = E\xi$. If a tree $\langle T_K, V_K\rangle$ satisfies φ, then there exists a path π in it such that $\pi \models \xi$. Thus, there exists an accepting run r of $\mathcal{U}_\xi$ on a word that agrees with π on the formulas in $max(\varphi)$. It is easy to see that a run of $\mathcal{A}_\varphi$ that proceeds on π according to r can accept $\langle T_K, V_K\rangle$. Indeed, by the definition of $\mathcal{A}'_\varphi$, the copies that proceeds according to δ' satisfy the acceptance condition. In addition, by the adjustment of $\mathcal{A}'_\varphi$ to the alphabet 2^{AP} and by the induction hypothesis, copies that take care of the maximal formulas can fulfill the acceptance condition. Now, if a run r of $\mathcal{A}_\varphi$ accepts a tree $\langle T_K, V_K\rangle$, then there must be a path π in this tree such that $\mathcal{A}_\varphi$ proceeds according to an accepting run of $\mathcal{U}_\xi$ on a word that agrees with π on the formulas in $max(\varphi)$. Thus, $\pi \models \xi$ and $\langle T_K, V_K\rangle$ satisfies φ.

We now consider the size of $\mathcal{A}_\psi$. For every φ, we prove, by induction on the structure of φ, that the size of $\mathcal{A}_\varphi$ is exponential in $\|\varphi\|$.

—Clearly, for $\varphi = p$ or $\varphi = \neg p$ for some $p \in AP$, the size of $\mathcal{A}_\varphi$ is constant.

—For $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, we have $\|\mathcal{A}_\varphi\| = O(\|\mathcal{A}_{\varphi_1}\| + \|\mathcal{A}_{\varphi_2}\|)$. By the induction hypothesis, $\|\mathcal{A}_{\varphi_1}\|$ is exponential in $\|\varphi_1\|$ and $\|\mathcal{A}_{\varphi_2}\|$ is exponential in $\|\varphi_2\|$. Thus, $\|\mathcal{A}_\varphi\|$ is surely exponential in $\|\varphi\|$.

—For $\varphi = E\xi$, we know, by Vardi and Wolper [1994], that the size of the word automaton $\mathcal{U}_\xi$ is exponential in $\|\xi\|$. Therefore, $\mathcal{A}'_\varphi$ is exponential in $\|\varphi\|$. Also, $|\Sigma'|$ is exponential in $|max(\varphi)|$ and, by the induction hypothesis, for all $\varphi_i \in max(\varphi)$, the size of $\mathcal{A}_{\varphi_i}$ is exponential in $\|\varphi_i\|$. Therefore, $\mathcal{A}_\varphi$ is also exponential in $\|\varphi\|$.

—For $\varphi = A\xi$, we know, by the above, that $\|\mathscr{A}_{E\neg\xi}\|$ is exponential in $\|\varphi\|$. Since complementing an HAA does not change its size, the result for φ follows.

Finally, since each subformula of ψ induces exactly one set, the depth of $\mathscr{A}_\psi$ is linear in $\|\psi\|$. □

We note that the same construction holds also for ECTL* [Vardi and Wolper 1984]. In ECTL*, formulas are constructed from Boolean connectives and automata connectives. As the construction above handles a formula $E\xi$ by translating the path formula ξ into an automaton, allowing automaton operators in the path formulas causes no difficulty. We handle these automaton operators as we handle the word automaton $\mathscr{U}_\xi$ constructed for a CTL* path formula ξ.

Example 5.4. Consider the CTL* formula $\psi = AGF(p \vee AXp)$. We describe the construction of $\mathscr{A}_{\mathscr{D},\psi}$, for $\mathscr{D} \subset \mathbb{N}$, step by step. Since ψ is of the form $A\xi$, we need to construct and dualize the HAA of $EFG((\neg p) \wedge EX\neg p)$. We start with the HAAs $\mathscr{A}_{\mathscr{D},\varphi}$ and $\tilde{\mathscr{A}}_{\mathscr{D},\varphi}$ for $\varphi = (\neg p) \wedge EX\neg p$.

$\mathscr{A}_{\mathscr{D},\varphi} = \langle\{\{p\}, \emptyset\}, \mathscr{D}, \{q_2, q_3\}, \delta, q_2, \langle\emptyset, \emptyset\rangle\rangle$, with

state q	$\delta(q,\{p\},k)$	$\delta(q,\emptyset,k)$
q_2	**false**	$\bigvee_{c=0}^{k-1}(c, q_3)$
q_3	**false**	**true**

$\tilde{\mathscr{A}}_{\mathscr{D},\varphi} = \langle\{\{p\}, \emptyset\}, \mathscr{D}, \{\tilde{q}_2, \tilde{q}_3\}, \tilde{\delta}, \tilde{q}_2, \langle\emptyset, \emptyset\rangle\rangle$, with

state q	$\tilde{\delta}(q,\{p\},k)$	$\tilde{\delta}(q,\emptyset,k)$
$\tilde{q}_2$	**true**	$\bigwedge_{c=0}^{k-1}(c, \tilde{q}_3)$
$\tilde{q}_3$	**true**	**false**

Starting with a sequential Büchi automaton $\mathscr{U}_\xi$ for $\xi = FG\varphi$, we construct $\mathscr{A}'_{\mathscr{D},E\xi}$.

$\mathscr{U}_\xi = \langle\{\{\varphi\}, \emptyset\}, \{q_0, q_1\}, M, q_0, \{q_1\}\rangle$, with

—$M(q_0, \{\varphi\}) = \{q_0, q_1\}$—$M(q_0, \emptyset) = \{q_0\}$
—$M(q_1, \{\varphi\}) = \{q_1\}$—$M(q_1, \emptyset) = \emptyset$

Hence, $\mathscr{A}'_{\mathscr{D},E\xi} = \langle\{\{\varphi\}, \emptyset\}, \mathscr{D}, \{q_0, q_1\}, \delta', q_0, \langle\{q_1\}, \emptyset\rangle\rangle$, with

state q	$\delta'(q,\{\varphi\},k)$	$\delta'(q,\emptyset,k)$
q_0	$\bigvee_{c=0}^{k-1}((c,q_0) \vee (c,q_1))$	$\bigvee_{c=0}^{k-1}(c, q_0)$
q_1	$\bigvee_{c=0}^{k-1}(c, q_1)$	**false**

We are now ready to compose the automata into an automaton over the alphabet $\{\{p\}, \emptyset\}$.

$\mathscr{A}_{\mathscr{D},E\xi} = \langle\{\{p\}, \emptyset\}, \mathscr{D}, \{q_0, q_1, q_3, \tilde{q}_3\}, \delta, q_0, \langle\{q_1\}, \emptyset\rangle\rangle$, with (note that we simplify the transitions, replacing **true** $\wedge\ \theta$ or **false** $\vee\ \theta$ by θ, replacing **true** $\vee\ \theta$ by **true**, and replacing **false** $\wedge\ \theta$ by **false**).

state q	$\delta(q, \{p\}, k)$	$\delta(q, \emptyset, k)$
q_0	$\bigvee_{c=0}^{k-1}(c, q_0)$	$\left(\bigvee_{c=0}^{k-1}((c, q_0) \vee (c, q_1)) \wedge \bigvee_{c=0}^{k-1}(c, q_3)\right) \vee \left(\bigvee_{c=0}^{k-1}(c, q_0) \wedge \bigwedge_{c=0}^{k-1}(c, \tilde{q_3})\right)$
q_1	**false**	$\bigvee_{c=0}^{k-1}(c, q_1) \wedge \bigvee_{c=0}^{k-1}(c, q_3)$
q_3	**false**	**true**
$\tilde{q_3}$	**true**	**false**

Consider $\delta(q_0, \emptyset, k)$. The first conjunction corresponds to the case where $\mathcal{A}'_{\mathcal{D},E\xi}$ guesses that φ holds in the present. Then, $\mathcal{A}_{\mathcal{D},E\xi}$ proceeds with $\delta'(q_0, \{\varphi\}, k)$ conjuncted with $\delta(q_2, \emptyset, k)$. The later guarantees that φ indeed holds in the present. The second conjunction corresponds to the case where φ does not hold in the present. Then, $\mathcal{A}_{\mathcal{D},E\xi}$ proceeds with $\delta'(q_0, \emptyset, k)$ conjuncted with $\delta(\tilde{q}_2, \emptyset, k)$.

We obtain $\mathcal{A}_{\mathcal{D},\psi}$ by dualizing $\mathcal{A}_{\mathcal{D},E\xi}$. Hence, $\mathcal{A}_{\mathcal{D},\psi} = \langle\{\{p\}, \emptyset\}, \mathcal{D}, \{\tilde{q}_0, \tilde{q}_1, \tilde{q}_3, q_3\}, \tilde{\delta}, \tilde{q}_0, \langle\emptyset, \{\tilde{q}_1\}\rangle\rangle$, with

state q	$\tilde{\delta}(q, \{p\}, k)$	$\tilde{\delta}(q, \emptyset, k)$
$\tilde{q_0}$	$\bigwedge_{c=0}^{k-1}(c, \tilde{q_0})$	$\left(\bigwedge_{c=0}^{k-1}((c, \tilde{q_0}) \wedge (c, \tilde{q_1})) \vee \bigwedge_{c=0}^{k-1}(c, \tilde{q_3})\right) \wedge \left(\bigwedge_{c=0}^{k-1}(c, \tilde{q_0}) \vee \bigvee_{c=0}^{k-1}(c, q_3)\right)$
$\tilde{q_1}$	**true**	$\bigwedge_{c=0}^{k-1}(c, \tilde{q_1}) \vee \bigwedge_{c=0}^{k-1}(c, \tilde{q_3})$
$\tilde{q_3}$	**true**	**false**
q_3	**false**	**true**

Consider the state $\tilde{q}_1$. A copy of $\mathcal{A}_{\mathcal{D},\psi}$ that visits $\tilde{q}_1$ keeps creating new copies of $\mathcal{A}_{\mathcal{D},\psi}$, all visiting $\tilde{q}_1$. Spreading a copy that visits $\tilde{q}_1$ stops only when it reaches a node that satisfies p or AXp. Since $\tilde{q}_1 \in B$, all the copies should eventually reach such a node. Hence, by sending a copy that visits the state $\tilde{q}_1$ to a node x, $\mathcal{A}_{\mathcal{D},\psi}$ guarantees that all the paths in the subtree rooted x eventually reach a node satisfying $p \bigvee AXp$. Hence, in the state $\tilde{q}_0$, unless $\mathcal{A}_{\mathcal{D},\psi}$ gets convinced that $p \bigvee AXp$ holds in the present, it sends copies that visit $\tilde{q}_1$ to all the successors. In addition, it always send copies visiting $\tilde{q}_0$ to all the successors.

Before we get to the space complexity of the 1-letter nonemptiness problem for HAAs over words, let us consider the time complexity of this problem.

THEOREM 5.5. *The* 1-*letter nonemptiness problem for hesitant alternating word automata is decidable in linear running time.*

PROOF. We present an algorithm with a linear running time for checking the nonemptiness of the language of an hesitant alternating word automaton $\mathcal{A} = \langle\{a\}, Q, \delta, q_0, \langle G, B\rangle\rangle$. The algorithm is very similar to the one described in the proof of Theorem 4.7. The only change is that here we do not have accepting and rejecting sets. Rather, being trapped in an existential set, the automaton should be able to visit states from G infinitely often. Similarly, being trapped in a universal set, the automaton should be able to visit states from B only finitely often. As in Theorem 4.7, the algorithm proceeds up a total order of the Q_i's, labeling states with 'T' and 'F', guaranteeing that when it reaches a set Q_i, all the states in all sets Q_j's for which $Q_j < Q_i$, have already been labeled. For a transient set Q_i, the above implies that the states in it have already been labeled too. For existential and universal sets, the algorithm proceeds as follows:

—In an existential set Q_i, all the transitions from states in Q_i only contain disjunctively related elements of Q_i. So, when the algorithm reaches Q_i, all its simplified transitions are disjunctions and induce an OR-graph that has states in Q_i as nodes. The algorithm partitions the graph into maximal strongly connected components Q_i^j. Since the partition is maximal, there exists a partial order over these components ($Q_i^j \leq Q_i^k$ if and only if there exists a transition from Q_i^k to Q_i^j) and there exists an extension of it into a total order $Q_i^1 \leq \cdots \leq Q_i^{m_i}$. The algorithm proceeds up this total order. When it reaches a component Q_i^j for which $Q_i^j \cap G \neq \emptyset$, then all the states in Q_i^j are labeled 'T'. Otherwise ($Q_i^j \cap G = \emptyset$), they are labeled 'F'. In both cases, the labeling is propagated. Indeed, a copy of $\mathcal{A}$ can stay in Q_i and visit states in G infinitely often if and only if it reaches a maximal strongly connected component Q_i^j, with $Q_i^j \cap G \neq \emptyset$.

—For a universal set, the transitions induce an AND-graph. Accordingly, when the algorithm reaches a maximal strongly connected component Q_i^j for which $Q_i^j \cap B \neq \emptyset$, all the states in Q_i^j are labeled 'F' (and the labeling is propagated). Otherwise, they are labeled 'T'. Here, a copy of $\mathcal{A}$ can stay in Q_i and visits states in B only finitely often if and only if it reaches a component Q_i^j, with $Q_i^j \cap B = \emptyset$.

Consider the total order $Q_1^1 \leq Q_1^2 \leq \cdots \leq Q_1^{m_1} \leq Q_2^1 \leq \cdots \leq Q_n^{m_n}$. We prove that for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$, all the states in Q_i^j are labeled correctly. The proof proceeds by induction on (i, j) (with the ordering $(i_1, j_1) < (i_2, j_2)$ iff $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$):

—The set Q_1 cannot be a transient set. Consider the case where it is an existential set. Then, for every $q \in Q_1^1$, it must be that $\delta(q, a)$ is a disjunction of states in Q_1^1. The algorithm labels all the states in Q_1^1 with 'T' if and only if $Q_1^1 \cap G \neq \emptyset$. Assume first that $Q_1^1 \cap G \neq \emptyset$. Then, as Q_1^1 is a strongly connected component, there exists, for every $q \in Q_1^1$, a state $q' \in G$ such that q' is reachable from q and from itself. Hence, there exists a run of $\mathcal{A}^q$ that visits (in its single branch) the state q' infinitely often. This run is accepting an thus the language of $\mathcal{A}^q$ is not empty. Assume now that $Q_1^1 \cap G = \emptyset$. Then, for every $q \in Q_1^1$, no run of $\mathcal{A}^q$ can visit, even once, a state in G. Thus, no run of $\mathcal{A}^q$ is an accepting run. The proof is analogous for the case in which Q_1 is a universal set.

—Assume that we have already labeled correctly all the states in all Q_k^l with $(k, l) < (i, j)$, and let $q \in Q_i^j$. If q is labeled before the phase (i, j), then, by the same consideration we used for WAAs, it is labeled correctly. Consider the case that q is labeled during the phase (i, j) and let Q_i be a universal set. Then, the simplification of $\delta(q, a)$ is a conjunction of states in Q_i^j and q it is labeled 'F' if and only if $Q_i^j \cap B \neq \emptyset$. Assume first that $Q_i^j \cap B \neq \emptyset$. Then, as Q_i^j is a strongly connected component, there exists, for every $q \in Q_i^j$, a state $q' \in B$ such that q' is reachable from q and from itself. Hence, in every run of $\mathcal{A}^q$, there is a copy that visits q' infinitely often. Thus, no run of $\mathcal{A}^q$ is an accepting run. Assume now that $Q_i^j \cap B = \emptyset$. Then, for every $q \in Q_i^j$, no copy of $\mathcal{A}^q$ can visit, even once, a state in B. So, every copy either stays in Q_i^j without visiting B, or reaches a state q' for which the language of $\mathcal{A}^{q'}$ is not empty. Thus, all the copies of $\mathcal{A}^q$ are accepting and the language of $\mathcal{A}^q$ is not empty. The proof is analogous for the case in which Q_i is an existential set.

Since partitioning each graph into maximal strongly connected components can be done in linear running time [Tarjan 1972], the overall running time remains linear, as with WAAs. The data structure used for a linear-time implementation is an AND/OR graph similar to the one described for WAAs. Since when testing WAAs for nonemptiness, we were dealing with simple WAAs, taking the states of the WAA as nodes induced an AND/OR graph in a straightforward way. Here, however, $\mathcal{A}$ is not simple. Let $\mathcal{A}'$ be the simple alternating word automaton obtained from $\mathcal{A}$ by the simplification described in Theorem 3.1, and let $Q'_1 \leq \cdots \leq Q'_n$ be the order on its sets (described there too). The automaton $\mathcal{A}'$ is no longer an HAA. Still, taking its states as the nodes of the AND/OR graph is adequate: the flow of the labeling in the graph guarantees that when the algorithm reaches an existential (universal) set, all the nodes of this set that are still not labeled induce an OR-graph (AND-graph) as required. □

So, the 1-letter nonemptiness problem for HAAs over words can be solved in linear running time. This, together with Theorem 5.3 and Proposition 3.2 (with the observation that the size of $\mathfrak{D}$ does not influence the size of $\mathcal{A}_{K,\psi}$), provides us with a model-checking procedure for CTL* with running time $O(\|K\|) \times 2^{O(\|\psi\|)}$). Note that the algorithm used there is essentially a bottom-up labeling of the transition graph of the automaton. We now show that by using a top-down exploration of this transition graph, we can get a space efficient 1-letter nonemptiness algorithm for HAAs.

THEOREM 5.6. *The* 1-*letter nonemptiness problem for hesitant alternating word automata of size n and depth m, can be solved in space $O(m \log^2 n)$.*

PROOF. Consider a HAA $\mathcal{A} = \langle \{a\}, Q, \delta, q_0, \langle G, B \rangle \rangle$. The algorithm described below assumes that the hesitation partition and order for $\mathcal{A}$ are given. Otherwise, it proceeds with the partition of Q into maximal strongly connected components. As discussed above, such a partition is a hesitation partition, it induces a hesitation order, and (as we elaborate further below) it is possible to check in space $O(\log^2 n)$ whether two given states in Q belong to the same set in the partition.

The property of HAAs we use is that, from a state in Q_i, it is possible to search for another reachable state in the same Q_i using space $O(\log^2 n)$. For a transient Q_i, there are no such states. For universal and existential Q_i, the exact notion of reachability we use is the transitive closure of the following notion of *immediate reachability*. Consider a set Q_i and assume that we have a Boolean value for all states in sets lower than Q_i. Then, a state $q' \in Q$ is immediately reachable from a state q, if it appears in the transition from q when this transition has been simplified using the known Boolean values for states in lower Q_i's. Note that the simplified transition is always a disjunction for a state of an existential Q_i, and a conjunction for a state of a universal Q_i. As we describe below, the algorithm does not perform simplification of the transitions, but rather computes recursively the values of states in lower Q_i's.

We call a state q' of Q_i *provably true* if, when the procedure is applied to the successors of q' that are not in Q_i, and the Boolean expression for the transition from q' is simplified, it is identically true. States that are *provably false* are defined analogously.

The following recursive procedure labels the states of the automaton with 'T' (accepts) or 'F' (does not accept).

(1) Start at the initial state.

(2) At a transient state q, evaluate the transition from q by recursively applying the procedure to the successor states of q. Label the state with the Boolean value that is obtained for the transition.

(3) At a state q of an existential Q_i, proceed as follows:

  (3.1) Search for a reachable state q' of the same Q_i that is provably true (note that this requires applying the procedure recursively to all states from lower Q_i's that are touched by the search). If such a state q' is found, label q with 'T'.

  (3.2) If no such state exists, search for a state $q' \in Q_i \cap G$ that is reachable from q and from itself. If such a state is found, label q with 'T'.

  (3.3) If none of the first two cases apply, label q with 'F'.

(4) At a state q of a universal Q_i, proceed as follows:

  (4.1) Search for a reachable state q' of the same Q_i that is provably false. If such a state q' is found, label q with 'F'.

  (4.2) If no such state exists, search for a state $q' \in Q_i \cap B$ that is reachable from q and from itself. If such a state is found, label q with 'F'.

  (4.3) If none of the first two cases apply, label q with 'T'.

With every state q we can associate a finite integer, $rank(q)$, corresponding to the depth of the recursion required in order to label q. It is easy to prove, by induction on $rank(q)$, that q is labeled correctly.

We now consider the complexity of our algorithm. We show that for each state $q \in Q_i$, if we already have Boolean values for all the states in sets lower than Q_i, then evaluating the Boolean value of q can be done deterministically in space $O(\log^2 n)$. In practice, we do not keep the Boolean values of the states in sets lower than Q_i. Instead, whenever we need such a value we evaluate it recursively. Clearly, the depth of the recursion is bounded by the number of sets lower than Q_i. Hence, evaluating the Boolean value of the initial state can be done in space $O(m \log^2 n)$.

We start with q that belongs to a transient set. There, we have to evaluate the transition from q. It is known that the problem of evaluating Boolean expressions is in LOGSPACE [Lynch 1977]. Here, we evaluate expressions over Q and the length of each expression is linear in n. Hence, evaluating the Boolean value of a transient state in Q_i, assuming we have Boolean values for all states in sets lower than Q_i, can be done deterministically in space $O(\log n)$.

Consider now a state $q \in Q_i$ for an existential set Q_i. For each state $q' \in Q_i$, we have that q' is provably true iff the transition from q' evaluates to **true** when we assign **false** to all the states in Q_i (and evaluates, using a recursion, states in lower sets). Checking the latter is as simple as evaluating a transition from a transient state. Thus, determining whether a state in Q_i is provably true assuming we have Boolean values for all states in sets lower than Q_i, can be done deterministically using space $O(\log n)$.

Furthermore, for each pair of states q' and q'' in an existential Q_i, we have that q'' is immediately reachable from q' iff the following two conditions hold. First, the transition from q' evaluates to **true** when we assign **true** to q'' and assign **false** to all the other states in Q_i (and evaluate recursively states from lower sets). Second, the transition from q' evaluates to **false** when we assign **false** to all the states in Q_i (and evaluate recursively states from lower sets). Again, checking this is as simple as evaluating a transition from a transient state. Thus, determining whether a given state in Q_i is immediately reachable from another given state in Q_i, assuming we have Boolean values for all states in sets lower than Q_i, can be done nondeterministically using space $O(\log n)$.

It is known, by Jones [1975], that the graph accessibility problem is in NLOGSPACE. Now, to check whether q'' is reachable from q', we restrict the graph accessibility test, replacing immediate accessibility with immediate reachability. This, as described above, can be done nondeterministically in space $O(\log n)$ (assuming we have Boolean values for all states in sets lower than Q_i). Thus, determining whether a given state in Q_i is reachable from another given state in Q_i can also be done nondeterministically in space $O(\log n)$. Hence, as the labeling of $q \in Q_i$ only involves a search for reachable states, we can determine its labeling nondeterministically in space $O(\log n)$, or, by Savitch [1970], deterministically in space $O(\log^2 n)$.

The case where $q \in Q_i$ for a universal set Q_i is symmetric. □

Theorems 5.1 and 5.3 provide us with the sizes and depths of the HAAs associated with formulas of CTL and CTL*. Hence, the theorem below follows from Proposition 3.2 and Theorem 5.6.

THEOREM 5.7

(1) *The model-checking problem for CTL can be solved in space $O(m\ log^2(mn))$, where m is the length of the formula and n is the size of the Kripke structure.*

(2) *The model-checking problem for CTL* can be solved in space $O(m(m + log\ n)^2)$, where m is the length of the formula and n is the size of the Kripke structure.*

The algorithm given in the proof of Theorem 5.6 can be viewed as a local model-checking algorithm. However, to be practical, the searches within the existential and universal components should be made deterministic, which cannot reasonably be done without forgoing some space efficiency and storing the markings as they are obtained. With these changes, the algorithm of the proof of Theorem 5.6 becomes an automata-theoretic counterpart of the algorithm presented in Vergauwen and Lewis [1993]. We note that once the search is made deterministic, the algorithm may need to generate computations of the Kripke structure that are not directly relevant for determining the truth of the formula in it. Still, the algorithm traverses, in the average, less states than bottom-up algorithms.

Now, let us define the *program complexity* [Vardi and Wolper 1986a] of model checking as the complexity of this problem in terms of the size of the input Kripke structure; that is, assuming the formula fixed.

THEOREM 5.8. *The program complexity of CTL and CTL* model checking is NLOGSPACE-complete.*

PROOF. Fixing the formula, we get an HAA of a fixed depth. According to the algorithm presented in the proof of Theorem 5.6, the nonemptiness problem for an hesitant alternating word automaton of a fixed depth is in NLOGSPACE. Thus, so is the program complexity of CTL and CTL* model checking. Hardness in NLOGSPACE is immediate by a reduction from the graph accessibility problem, proved to be NLOGSPACE-complete in Jones [1975]. □

The fact that CTL and CTL* formulas can be translated to HAAs plays a crucial role in our upper bounds. To see this, we prove in Theorem 5.9 below, that the 1-letter nonemptiness problem for weak alternating word automata is P-complete. Thus, the restricted structure of HAAs is essential for a space-efficient nonemptiness test.

THEOREM 5.9. *The* 1*-letter nonemptiness problem for weak alternating word automata is P-complete.*

PROOF. Membership in P follows from Theorem 4.7. Hardness in P follows by a reduction from the Alternating Graph Accessibility problem, proved to be P-complete in Immerman [1981], Chandra et al. [1981], and Greenlaw et al. [1995], to nonemptiness of weak alternating word automata. In the Alternating Graph Accessibility problem, we are given a directed graph $G = \langle V, E \rangle$, a partition $\mathcal{E} \cup \mathcal{U}$ of V, and two designated vertices s and t. The problem is whether *alternating_path*(s, t) is true, where *alternating_path*(x, y) holds if and only if:

(1) $x = y$, or
(2) $x \in \mathcal{E}$ and there exists z with $\langle x, z \rangle \in E$ and *alternating_path*(z, y), or
(3) $x \in \mathcal{U}$ and for all z with $\langle x, z \rangle \in E$, we have *alternating_path*(z, y).

Given G, $\mathcal{E}$, $\mathcal{U}$, s, and t, we define the weak alternating word automaton $\mathcal{A} = \langle \{a\}, V, \delta, s, \{t\} \rangle$ where δ is defined as follows:

—If $q \in \mathcal{E}$ and $q \neq t$, then $\delta(q, a) = \bigvee_{\langle q,q' \rangle \in E} q'$.
—If $q \in \mathcal{U}$ and $q \neq t$, then $\delta(q, a) = \bigwedge_{\langle q,q' \rangle \in E} q'$.
—$\delta(t, a) = \textbf{true}$.

It is easy to see that a partition of V into two sets, $V \backslash \{t\}$ and $\{t\}$, satisfies the weakness requirements and that the language of $\mathcal{A}$ is not empty iff *alternating_path*(s, t). □

Recall that while we can translate alternation-free μ-calculus formulas to WAA, we can not translate them to HAA. As we show in Theorem 5.10 below, this inability is reflected in the program complexity of alternation-free μ-calculus.

THEOREM 5.10. *The program complexity of alternation-free μ-calculus model checking is P-complete.*

PROOF. The upper bound follows from the known linear complexity of AFMC model checking. As in the proof of Theorem 5.9, we are doing a reduction from the Alternating Graph Accessibility problem. Given $G = \langle V, E \rangle$,

$\mathcal{E}$, $\mathcal{U}$, s, and t, we define the Kripke structure $K_G = \langle \{t, e, u\}, V, E, s, L \rangle$ where for every state $v \in V$, we have

$$L(v) = \left[\begin{array}{lll} t & \text{if} & v = t, \\ e & \text{if} & v \in \mathcal{E} \setminus \{t\}, \\ u & \text{if} & v \in \mathcal{U} \setminus \{t\}. \end{array} \right.$$

It is easy to see that the state s of K_G satisfies the fixed alternation-free μ-calculus formula $\mu y.(t \vee (e \wedge EXy) \vee (u \wedge AXy))$ iff *alternating_path* (s, t). □

6. *The Complexity of Model Checking for Concurrent Programs*

We consider a concurrent program P composed of n concurrent processes P_i. Each process is described by a transition system $D_i = \langle AP_i, AC_i, S_i, \Delta_i, s_i^0, L_i \rangle$ where AP_i is a set of atomic propositions, AC_i is an action alphabet, S_i is a finite set of states, $\Delta_i \subseteq S_i \times AC_i \times S_i$ is a transition relation, $s_i^0 \in S_i$ is an initial state, and L_i: $S_i \rightarrow 2^{AP_i}$ maps each state to the set of atomic propositions true in this state. We require that the atomic-proposition sets of the processes are disjoint.

A concurrent behavior of these processes is defined by the usual interleaving semantics: transition actions that appear in several processes are synchronized by common actions. Using this convention, one can obtain a global transition system D describing the joint behavior of the processes P_i. This global transition system is computed by constructing the reachable states of the product of the processes P_i. This product is the transition system $D = \langle AP, AC, S, \Delta, s^0, L \rangle$ where,

—$AP = \bigcup_{1 \leq i \leq n} AP_i$.

—$AC = \bigcup_{1 \leq i \leq n} AC_i$.

—$S = \Pi_{1 \leq i \leq n} S_i$. We denote the ith component of a state $s \in S$ by $s[i]$.

—$\langle s, a, s' \rangle \in \Delta$ if and only if

  —for all $1 \leq i \leq n$ such that $a \in AC_i$, we have $\langle s[i], a, s'[i] \rangle \in \Delta_i$, and

  —for all $1 \leq i \leq n$ such that $a \notin AC_i$, we have $s[i] = s'[i]$.

—$s^0 = \langle s_1^0, s_2^0, \ldots, s_n^0 \rangle$.

—For every $s \in S$, we have $L(s) = \bigcup_i L_i(s[i])$.

We define the complexity of model checking for a concurrent program P with respect to the size of its components P_i and the length of the formula being checked. Accordingly, the program complexity of model checking for concurrent programs is defined in terms of the size of the components of the concurrent program. It is shown in Kozen [1977] that that the nonemptiness problem for a concurrent program P is PSPACE-complete. This immediately implies a PSPACE lower bound for the complexity of CTL and CTL* model checking for concurrent programs. We still present a similar proof below, as we shall later use a variant of it in the proof of Theorem 6.3.

THEOREM 6.1. *The complexity of CTL and CTL* model checking for concurrent programs is PSPACE-complete.*

PROOF. We have just proved that the model-checking problem for CTL and CTL* can be solved in space polynomial in the length of the formula but only poly-logarithmic in the size of the Kripke structure. Since the product of the components of a concurrent program is at most exponentially larger than the program, membership in PSPACE follows directly by an argument similar to the one developed in Vardi and Wolper [1994]. To prove that it is hard in PSPACE, we do a reduction from polynomial space Turing machines.

We show that there exists a CTL formula ψ such that given a Turing machine T of space complexity $s(n)$, it is possible to build a concurrent program P of size $O(s(n))$ such that P satisfies ψ if and only if T accepts the empty tape.

Let $T = \langle \Gamma, Q, \rightarrow, q_0, F \rangle$ be a Turing machine where Γ is the alphabet, Q is the set of states, $\rightarrow \subseteq Q \times \Gamma \times Q \times \Gamma \times \{R, L\}$ is the transition relation (we use $(q, a) \rightarrow (q', b, \Delta)$ to indicate that when T is in state q and it reads the input a in the current tape cell, it moves to state q', writes b in the current tape cell, and its reading head moves one cell to the right/left, according to Δ), q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. The concurrent program P has $s(n)$ processes, one for each tape cell that is used. For all $1 \leq i \leq s(n)$, the process P_i is defined as follows:

(1) $AP_i = \{accept_i\}$ and $AC_i = \{i - 1, i, i + 1\} \times Q$.

(2) The state set of P_i is $(Q \times \Gamma) \cup \Gamma$. A state of the form (q, a) indicates that T is in state q, its reading head is on cell i, and the content of cell i is a. A state of the form a indicates that the content of cell i is a and the reading head is not on cell i.

(3) For each transition $(q, a) \rightarrow (q', b, \Delta)$ of T, we have the following transitions in P_i.

  (a) A transition from (q, a) to b labeled by $(i + 1, q')$ if $\Delta = R$ and by $(i - 1, q')$ if $\Delta = L$. This transition corresponds to the head moving from cell i to cell $i + 1$ or $i - 1$.

  (b) A transition from every $a \in \Gamma$ to (q', a) labeled by (i, q'). This transition corresponds to the head moving to cell i from cell $i + 1$ or $i - 1$.

(4) The initial state of P_i corresponds to the initial content of cell i. Thus, it is (q_0, ϵ) for $i = 1$ and it is ϵ for $1 < i \leq s(n)$.

(5) We label a state (q, a) with $accept_i$ if and only if $q \in F$.

The concurrent behavior of the processes embodies all the computations of T on the empty tape. To see this, observe that each reachable state s in P has exactly one $1 \leq i \leq n$ for which $s[i] \in Q \times \Gamma$. Thus, each reachable state in P corresponds to a configuration of T. Also, a transition from a state s_1 to a state s_2 corresponds to a possible transition from the configuration associated with s_1 to the one associated with s_2.

Now, let $accept = \bigvee_i accept_i$. Consider the CTL formula $\psi = EFaccept$. $P \models \psi$ if and only if there exists a computation of T on the empty tape in which eventually reaches an accepting state. Thus, $P \models \psi$ if and only if T accepts the empty tape. □

THEOREM 6.2. *The program complexity of CTL and CTL* model checking for concurrent programs is PSPACE-complete.*

PROOF. Clearly, the upper bound proved in Theorem 6.1 holds here too. Since proving the lower bound there we used a fixed formula, hardness in PSPACE holds also here. □

We now consider the program complexity of μ-calculus and alternation-free μ-calculus model checking for concurrent programs. The WAAs that correspond to alternation-free μ-calculus do not have the restricted structure of HAAs. In Example 4.5, we presented the WAA that corresponds to the formula $\psi = \mu y.(p \vee EXAXy)$ and in which an alternation between existential and universal states that belong to the same set is possible. Thus, while the syntax of CTL and CTL* makes the alternation of the path quantifiers A and E bounded by the length of the formula, the fixed-point operators in μ-calculus enable an unbounded alternation. This unbounded alternation is the key for the following theorem:

THEOREM 6.3. *The program complexity of alternation-free μ-calculus model checking for concurrent programs is EXPTIME-complete.*

PROOF. Clearly, the problem can be solved in EXPTIME by building the nondeterministic program corresponding to the concurrent program and using the model-checking algorithm from Section 4.1. To prove that it is hard in EXPTIME, we do a reduction from alternating linear-space Turing machines, proved to be EXPTIME-hard in Chandra et al. [1981].

Similarly to what we have done proving Theorem 6.1, we show that there exists an alternation-free μ-calculus formula ψ such that given an alternating Turing machine T of space complexity $s(n)$, it is possible to build, with a logarithmic space construction, a concurrent program P of size $O(s(n))$ such that P satisfies ψ if and only if T accepts the empty tape. The model of alternation we use is that the transition relation is universal in its even steps and is existential in its odd steps.

Given T, the construction of P is exactly the same as in Theorem 6.1. Consider the μ-calculus formula

$$\psi = \mu y.(accept \vee EX(accept \vee AXy)).$$

$P \models \psi$ if and only if there exists a computation of T on the empty tape in which all the leaves of the computation tree eventually reach an accepting state. Thus, $P \models \psi$ if and only if T accepts the empty tape. □

In Theorem 6.4 below, we show that μ-calculus model checking for concurrent programs can also be done in time exponential in the program. Thus, the program complexities of model checking for concurrent programs for alternation-free μ-calculus and general μ-calculus coincide.

THEOREM 6.4. *The program complexity of μ-calculus model checking for concurrent programs is EXPTIME-complete.*

PROOF. Clearly, hardness in EXPTIME follows from Theorem 6.3. To prove membership in EXPTIME, we use the algorithm suggested in Emerson and Lei [1986]. Given a concurrent program P and a μ-calculus formula ψ, the size of a Kripke structure K that models the nondeterministic expansion of P is of size exponential in the size of P. According to Emerson and Lei [1986], model checking of a Kripke structure K with respect to a μ-calculus formula ψ is of time complexity $O((\|\psi\| * \|K\|)^{n+1})$, where n is the alternation depth of ψ. The alternation depth of a formula ψ is the maximal number of alternations between μ and ν on any syntactic path from an occurrence of μy or νy to an occurrence of y (see Emerson and Lei [1986]). We clearly have that $n < \|\psi\|$ and, therefore, model checking of K with respect to ψ is of time complexity $O((\|\psi\| * 2^{|P|})^{\|\psi\|})$. Hence, fixing ψ we get that the program complexity of μ-calculus model checking for concurrent programs is in EXPTIME. □

Theorem 6.3 implies that while the time complexities of model checking for CTL and alternation-free μ-calculus coincide, there is probably a gap between the space complexities of model checking for these logics. Moreover, the program complexity of model checking for CTL* is probably lower than the program complexity of model checking for the alternation-free μ-calculus. To conclude, the automata-theoretic framework provides improved (and significant) space-complexity upper bounds for the model checking problem of CTL and CTL* and explains why similar improved bounds cannot be obtained for model checking of the μ-calculi.

7. *Discussion*

In this work, we argue that alternating tree automata provide a comprehensive and uniform framework for branching temporal logics. While the satisfiability problem for these logics reduces to the nonemptiness problem for alternating tree automata, the model checking problem reduces to the (much easier) 1-letter nonemptiness problem for these automata. The automata-theoretic approach separates the logical and the combinatorial aspects of reasoning about systems. The translation of specifications to automata handles the logic and shifts all the combinatorial difficulties to automata-theoretic problems. This enabled us to improve the space complexity of branching-time model-checking. In particular, we show that alternating tree automata provide a PSPACE procedure for CTL* model checking of concurrent programs, and provide an explanation why this bound can not be achieved for μ-calculus model checking and even for its alternation-free fragment.

The automata-theoretic approach to branching-time model checking described here has contributed to several other results in the area of specification and verification of reactive systems. In Kupferman and Vardi [1995], the framework is extended to handle (and improve the space complexity of) branching-time model checking for fair systems. In Henzinger et al. [1996], the framework is extended to handle real-time and hybrid systems, and in Kupferman and Vardi [1996], Alur et al. [1997], and Kupferman et al. [1997], it is extended for the verification of open systems. One of the difficulties present when reasoning about open and distributed systems is fact that the components of a system may have internal variables, thus the other components have incomplete information about the global configuration of the system. The structure of alternating tree automata has turned out to be particularly suitable for coping with incomplete information, and in Kupferman and Vardi [1997; 1999a; 1999b], the automata-theoretic approach is extended to handle verification and synthesis of open systems. Finally, the approach is combined with partial-order methods in Willems and Wolper [1996], combined with the assume-guarantee paradigm for modular model checking in Vardi [1995] and Kupferman and Vardi [1998b], and it contributes to the study of linear-time properties that can be checked efficiently and symbolically in Kupferman and Vardi [1998a].

ACKNOWLEDGMENTS. We thank Thomas Wilke for helpful comments on a previous draft of this paper.

REFERENCES

ALUR, R., HENZINGER, T. A., AND KUPFERMAN, O. 1997. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 100–109.

ANDERSEN, H. R., AND VERGAUWEN, B. 1995. Efficient checking of behavioural relations and modal assertions using fixed-point inversion. In *Computer Aided Verification*, *Proceedings of the 7th International Conference* (Liege, Belgium, July). Lecture Notes in Computer Science, vol. 939. Springer-Verlag, New York, pp. 142–154.

ANDERSON, H. R. 1992. Model checking and boolean graphs. In *Proceedings of the European Symposium on Programming (ESOP)*. Lecture Notes in Computer Science, vol. 582. Springer-Verlag, New York, pp. 1–19.

BANIEQBAL, B., AND BARRINGER, H. 1987. Temporal logic with fixed points. In *Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, Eds. Lecture Notes in Computer Science, vol. 398. Springer-Verlag, New York, pp. 62–74.

BEERI, C. 1980. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. Datab. Syst. 5*, 3 (Sept.), 241–259.

BEERI, C., AND BERNSTEIN, P. A. 1979. Computational problems related to the design of normal form relational schemas. *ACM Trans. Datab. Syst. 4*, 1 (Mar.), 30–59.

BERNHOLTZ, O., AND GRUMBERG, O. 1993. Branching time temporal logic and amorphous tree automata. In *Proceedings of the 4th Conference on Concurrency Theory* (Hildesheim, Aug.). Lecture Notes in Computer Science, vol. 715. Springer-Verlag, New York, pp. 262–277.

BHAT, G., AND CLEAVELAND, R. 1996a. Efficient local model-checking for fragments of the modal μ-calculus. In *Proceedings of the 1996 Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes on Computer Science, vol. 1055. Springer-Verlag, Berlin, Germany.

BHAT, G., AND CLEAVELAND, R. 1996b. Efficient model checking via the equational μ-calculus. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 304–312.

CHANDRA, A. K., KOZEN, D. C., AND STOCKMEYER, L. J. 1981. Alternation. *J. ACM 28*, 1 (Jan.), 114–133.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Syst. 8*, 2 (Apr.), 244–263.

CLARKE, E. M., GRUMBERG, O., AND LONG, D. 1993. Verification tools for finite-state concurrent systems. In *Decade of Concurrency—Reflections and Perspectives (Proceedings of REX School)* J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds. Lecture Notes in Computer Science, vol. 803. Springer-Verlag, New York, pp. 124–175.

CLEAVELAND, R. 1993. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Form. Meth. Syst. Des. 2*, 121–147.

COURCOUBETIS, C., VARDI, M. Y., WOLPER, P., AND YANNAKAKIS, M. 1992. Memory efficient algorithms for the verification of temporal properties. *Form. Meth. Syst. Des. 1*, 275–288.

DOWLING, W. F., AND GALLIER, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Logic Prog. 1*, 3, 267–284.

EMERSON, E. A. 1985. Automata, tableaux, and temporal logics. In *Proceedings of the Workshop on Logic of Programs*. Lecture Notes in Computer Science, vol. 193. Springer-Verlag, New York, pp. 79–87.

EMERSON, E. A. 1996. Automated temporal reasoning about reactive systems. In *Proceedings of the VIII-th BANFF Higher Order Workshop*. Lecture Notes in Computer Science, vol. 1043. Springer-Verlag, New York, pp. 41–101.

EMERSON, E. A., AND HALPERN, J. Y. 1986. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. ACM 33*, 1 (Jan.), 151–178.

EMERSON, E. A., AND JUTLA, C. 1988. The complexity of tree automata and logics of programs. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science* (White Plains, N.Y., Oct.) IEEE Computer Society Press, Los Alamitos, Calif., pp. 328–337.

EMERSON, E. A., AND JUTLA, C. 1991. Tree automata, Mu-calculus and determinacy. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science* (San Juan, P.R., Oct.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 368–377.

EMERSON, E. A., JUTLA, C., AND SISTLA, A. P. 1993. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proceedings of the 5th International Conference* (Elounda, Crete, June). Lecture Notes in Computer Science, vol. 697. Springer-Verlag, New York, pp. 385–396.

EMERSON, E. A., AND LEI, C.-L. 1986. Efficient model checking in fragments of the propositional μ-calculus. In *Proceedings of the 1st Symposium on Logic in Computer Science* (Cambridge, Mass., June). pp. 267–278.

EMERSON, E. A., AND SISTLA, A. P. 1984. Deciding branching time logic. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing* (Washington, D.C., Apr. 30–May 2). ACM, New York, pp. 14–24.

FISCHER, M. J., AND LADNER, R. E. 1979. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci. 18*, 194–211.

GRAEDEL, E., AND WALUKIEWICZ, I. 1999. Guarded fixed point logic. In *Proceedings of the 14th Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.

GREENLAW, R., HOOVER, H. J., AND RUZZO, W. L. 1995. *Limits of parallel computation*. Oxford University Press, Oxford, England.

HENZINGER, T. A., KUPFERMAN, O., AND VARDI, M. Y. 1996. A space-efficient on-the-fly algorithm for real-time model checking. In *Proceedings of the 7th Conference on Concurrency Theory* (Pisa, Italy, Aug.). Lecture Notes in Computer Science, vol. 1119. Springer-Verlag, New York, pp. 514–529.

IMMERMAN, N. 1981. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci. 22*, 3, 384–406.

JARD, C., AND JERON, T. 1989. On-line model-checking for finite linear temporal logic specifications. In Automatic Verification Methods for Finite State Systems, *Proceedings of the International Workshop* (Grenoble, Switzerland, June). Lecture Notes in Computer Science, vol. 407. Springer-Verlag, New York, pp. 189–196.

JONES, N. D. 1975. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci. 11*, 68–75.

JUTLA, C. S. 1990. Automata on infinite objects and modal logics of programs. Ph.D. dissertation, Univ. Texas, Austin, Texas.

KOZEN, D. 1977. Lower bounds for natural proof systems. In *Proceedings of the 18th IEEE Symposium on Foundation of Computer Science*. IEEE Computer Science Press, Los Alamitos, Calif., pp. 254–266.

KOZEN, D. 1983. Results on the propositional μ-calculus. *Theoret. Comput. Sci. 27*, 333–354.

KUPFERMAN, O., AND VARDI, M. Y. 1995. On the complexity of branching modular model checking. In *Proceedings of the 6th Conference on Concurrency Theory* (Philadelphia, Pa., Aug.). Lecture Notes in Computer Science, vol. 962. Springer-Verlag, New York, pp. 408–422.

KUPFERMAN, O., AND VARDI, M. Y. 1996. Module checking. In *Computer Aided Verification, Proceedings of the 8th International Conference*. Lecture Notes in Computer Science, vol. 1102. Springer-Verlag, New York, pp. 75–86.

KUPFERMAN, O., AND VARDI, M. Y. 1997. Module checking revisited. In *Computer Aided Verification*, *Proceedings of the 9th International Conference*. Lecture Notes in Computer Science, vol. 1254. Springer-Verlag, New York, pp. 36–47.

KUPFERMAN, O., AND VARDI, M. Y. 1998a. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 81–92.

KUPFERMAN, O., AND VARDI, M. Y. 1998b. Modular model checking. In *Proceedings of the Compositionality Workshop*. Lecture Notes in Computer Science, vol. 1536. Springer-Verlag, New York, pp. 381–401.

KUPFERMAN, O., AND VARDI, M. Y. 1998c. Weak alternating automata and tree automata emptiness. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing* (Dallas, Tex., May 23–26). ACM, New York, pp. 224–233.

KUPFERMAN, O., AND VARDI, M. Y. 1999a. Church's problem revisited. *Bull. Symb. Logic 5,* 2, 245–263.

KUPFERMAN, O., AND VARDI, M. Y. 1999b. Robust satisfaction. In *Proceedings of the 10th Conference on Concurrency Theory*. Lecture Notes in Computer Science. Springer-Verlag, New York.

KUPFERMAN, O., VARDI, M. Y., AND WOLPER, P. 1997. Module checking. *Inf. Comput*. to appear.

LAMPORT, L. 1980. Sometimes is sometimes "not never"—On the temporal logic of programs. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, ACM, New York, pp. 174–185.

LARSEN, K. G. 1992. Efficient local correctness checking. In *Proceedings of the 4th Conference on Computer Aided Verification* (Montreal, Que., Canada, June). Lecture Notes in Computer Science, Springer-Verlag, New York, pp. 30–43.

LICHTENSTEIN, O., AND PNUELI, A. 1985. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages* (New Orleans, La., Jan.) ACM, New York, pp. 97–107.

LYNCH, N. 1977. Log space recognition and translation of parenthesis languages. *J. ACM 24*, 4 (Oct.), 583–590.

MIHAIL, M., AND PAPADEMITRIOU, C. H. 1994. On the random walk method for protocol testing. In *Proceedings of the 5th Annual Conference on Computer Aided Verification* (Stanford, Calif., June). Lecture Notes in Computer Science, vol. 818. Springer-Verlag, New York, pp. 132–141.

MIYANO, S., AND HAYASHI, T. 1984. Alternating finite automata on ω-words. *Theoret. Comput. Sci. 32*, 321–330.

MULLER, D. E., SAOUDI, A., AND SCHUPP, P. E. 1986. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proceedings of the 13th International Colloquium on Automata, Languages and Programming*. Springer-Verlag, New York.

MULLER, D. E., SAOUDI, A., AND SCHUPP, P. E. 1988. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science* (Edinburgh, Scotland, July). IEEE Computer Society Press, Los Alamitos, Calif., pp. 422–427.

MULLER, D. E., AND SCHUPP, P. E. 1987. Alternating automata on infinite trees. *Theoret. Comput. Sci. 54*, 267–276.

PNUELI, A. 1981. The temporal semantics of concurrent programs. *Theoret. Comput. Sci. 13*, 45–60.

QUEILLE, J. P., AND SIFAKIS, J. 1981. Specification and verification of concurrent systems in Cesar. In *Proceedings of the 5th International Symposium on Programming*. Lecture Notes in Computer Science, vol. 137. Springer-Verlag, New York, pp. 337–351.

RABIN, M. O. 1969. Decidability of second order theories and automata on infinite trees. *Trans. AMS 141*, 1–35.

RABIN, M. O. 1970. Weakly definable relations and special automata. In *Proceedings of the Symposium on Mathematical Logic and Foundations of Set Theory*. North Holland, Amsterdam, New York, pp. 1–23.

SAVITCH, W. J. 1970. Relationship between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci. 4*, 177–192.

SLUTZKI, G. 1985. Alternating tree automata. *Theoret. Comput. Sci. 41*, 305–318.

STIRLING, C. 1996. Games and modal mu-calculus. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, vol. 1055. Springer-Verlag, New York, pp. 298–312.

STIRLING, C., AND WALKER, D. 1991. Local model checking in the modal Mu-calculus. *Theoret. Comput. Sci. 89*, 1, 161–177.

STREET, R. S., AND EMERSON, E. A. 1984. An elementary decision procedure for the μ-calculus. In *Proceedings of the 11th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 172. Springer-Verlag, New York, pp. 465–472.

TARJAN, R. E. 1972. Depth first search and linear graph algorithms. *SIAM J. Comput. 1*, 2, 146–160.

THOMAS, W. 1990. Automata on infinite objects. *Handb. Theoret. Comput. Sci.*, 165–191.

VARDI, M. Y. 1982. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, pp. 137–146.

VARDI, M. Y. 1995. On the complexity of modular model checking. In *Proceedings of the 10th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 101–111.

VARDI, M. Y., AND STOCKMEYER, L. 1985. Improved upper and lower bounds for modal logics of programs. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (Providence, R.I., May 6–8). ACM, New York, pp. 240–251.

VARDI, M. Y., AND WOLPER, P. 1984. Yet another process logic. In *Logics of Programs*. Lecture Notes in Computer Science, vol. 164. Springer-Verlag, New York, pp. 501–512.

VARDI, M. Y., AND WOLPER, P. 1986a. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science* (Cambridge, Mass., June). pp. 322–331.

VARDI, M. Y., AND WOLPER, P. 1986b. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci. 32*, 2 (Apr.), 182–221.

VARDI, M. Y., AND WOLPER, P. 1994. Reasoning about infinite computations. *Inf. Comput. 115*, 1 (Nov.), 1–37.

VERGAUWEN, B., AND LEWIS, J. 1993. A linear local model checking algorithm for CTL. In *Proceedings of the 4th Conference on Concurrency Theory* (Hildesheim, Aug.). Lecture Notes in Computer Science, vol. 715. Springer-Verlag, New York, pp. 447–461.

VISSER, W. 1998. Efficient CTL* model checking using games and automata. Ph.D. dissertation. Manchester University.

VISSER, W., AND BARRINGER, H. 1999. CTL* model checking for SPIN. In *Software Tools for Technology Transfer*. Lecture Notes in Computer Science, Springer-Verlag, New York.

WILLEMS, B., AND WOLPER, P. 1996. Partial-order methods for model checking: From linear time to branching time. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. (New Brunswick, N.J., July).

WOLPER, P. 1983. Temporal logic can be more expressive. *Inf. Control, 56*, 1–2, 72–99.

WOLPER, P. 1989. On the relation of programs and computations to models of temporal logic. In *Proceedings of the Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, Eds. Lecture Notes in Computer Science, Springer-Verlag, New York, pp. 75–123.

RECEIVED JUNE 1998; REVISED AUGUST 1998; ACCEPTED AUGUST 1999