To appear in: Computer Aided Verification, Proc, Int. Workshop, Lecture Notes in Computer Science, Springer–Verlag, Stanford, June, 1994.

An Automata-Theoretic Approach to Branching-Time Model Checking (Extended Abstract)

Orna Bernholtz[1] and Moshe Y. Vardi[2] and Pierre Wolper[3*]

[1] The Technion, Department of Computer Science, Haifa 32000, Israel.
Email: ornab@cs.technion.ac.il
[2] Rice University, Department of Computer Science, P.O. Box 1892, Houston, TX 77251-1892, U.S.A. Email: vardi@cs.rice.edu
[3] Université de Liège, Institut Montefiore, B28, 4000 Liège Sart-Tilman, Belgium.
Email: pw@montefiore.ulg.ac.be

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 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, as was shown by Muller et al., can they be used to obtain optimal decision procedures, 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.

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 [Pnu81]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [CES86, QS81]. 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

* The work of this author was supported by the Esprit BRA action REACT and by the Belgian Incentive Program "Information Technology" - Computer Science of the future, initiated by the Belgian State - Prime Minister's Office - Science Policy Office. Scientific responsibility is assumed by its authors.

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. A survey can be found in [Wol89].

We distinguish between two types of temporal logics: linear and branching [Lam80]. 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 on infinite words has been developed [VW86a, VW94]. The basic idea is to associate with each linear temporal logic formula a finite automaton on 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 the key to techniques such as *on-the-fly* verification [VW86a, JJ89, CVWY92] that help coping with the "state-explosion" problem.

For branching temporal logics, the automata-theoretic counterpart are automata on infinite trees. By reducing the satisfiability to the nonemptiness problem for these automata, optimal decision procedures have been obtained for various branching temporal logics [Eme85, EJ88, ES84, SE84, VW86b]. Unfortunately, the automata-theoretic approach does not seem to be applicable to branching-time model checking. Indeed, model checking can be done in linear running time for CTL [CES86, QS81] and the alternation-free fragment of the μ-calculus [Cle93], and is in NP$\cap$co-NP for the general μ-calculus [EJS93], whereas there is an exponential blow-up involved in going from formulas to automata. Thus, using the construction of a tree automaton as a step in a model-checking algorithm seems a non-starter, which can only yield algorithms with exponential running time. (Indeed, the proof in [EJS93] 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 in [BG93]. Amorphous automata have a flexible transition relation that can adapt to trees with varying branching degree. Simultaneous trees are trees in which each sub-tree 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 [MSS88, EJ91]. In fact, Emerson stated that "μ-calculus formulas are simply alternating tree automata" [Eme94]. Muller et al. 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, but rather the *1-letter* nonemptiness problem. This problem (testing the nonemptiness of an alternating automaton that is defined on trees labeled with a singleton alphabet) is substantially simpler. Thus, alternating tree automata provide a unifying and optimal framework for both 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. in [MSS88] 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. used the fact that the nonemptiness problem for these automata is in exponential time [MSS86]. We prove that their 1-letter nonemptiness is decidable in linear running time, which yields an automata-based model checking algorithm of linear running time for CTL. The same technique can also be used to show that model-checking for the alternation-free μ-calculus can be done in linear running time. For the general μ-calculus, it follows from the results in [EJ91] that μ-calculus formulas can be linearly translated to alternating *Rabin* automata. We prove here that the 1-letter nonemptiness of alternating Rabin 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 CTL [CES86] and the μ-calculus [Cle93, EJS93], 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 [MSS88] allows, in the spirit of [Wol83], 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 which are irrelevant for the formula to be checked, and hence addresses the issue raised in the work on *local* model checking [SW91, VL93], while preserving optimal complexity and ease of implementation.

The third advantage of the automata-theoretic approach is that it offers new and significant insights into the space complexity of CTL model checking. It comes from the observation that the weak alternating automata that are obtained from CTL formulas have a special structure: they have *bounded alternation*. A careful analysis of the 1-letter nonemptiness problem for weak alternating automata with this property yields a top-down model-checking algorithm for CTL that is in NLOGSPACE in the size of the Kripke structure. A similar result holds for CTL*. This is very significant since it implies that, for concurrent programs, model checking 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.

2 Preliminaries

2.1 Temporal Logics and μ-Calculi

The temporal logic CTL (Computation Tree Logic) provides temporal operators that are composed of a path quantifier immediately followed by a single linear-time operator. The path quantifiers are A ("for all paths") and E ("for some path"). The allowed linear-time operators are 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. For technical convenience, we use the linear-time operator $\tilde{U}$ as a dual of the U operator ($p\tilde{U}q \equiv \neg((\neg p)U(\neg q))$), and write all CTL formulas in positive normal form. The *closure*, $cl(\psi)$, of a CTL formula ψ is the set of all CTL subformulas of ψ (including ψ). It is easy to see that for every ψ, $|cl(\psi)| \leq |\psi|$. The logic CTL* is defined similarly to CTL except that arbitrary linear-time formulas can appear in the scope of a path quantifier. See [Eme90] for more details on CTL and CTL*.

The propositional μ-calculus is a propositional modal logic augmented with least and greatest fixpoint operators. 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. For example, the μ-calculus formula $\mu y(q \vee (p \wedge EXy))$ is equivalent to the CTL formula $EpUq$. Assuming μ-calculus formulas are written in positive normal form (negation only applied to atomic propositions), a formula is *alternation free* if there are no occurrences of ν (μ) on any syntactic paths from an occurrence of μy (νy) to an occurrence of y. For more details, see [Koz83, EJ88, Var88].

The semantics of the logics described above is defined with respect to a *Kripke structure*, $K = \langle W, R, w^0, L\rangle$, where 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 a set of atomic propositions true in this state. The notation $K, w \models \varphi$ indicates that a formula φ holds at a state w of the structure K. Also, $K \models \varphi$ iff $K, w^0 \models \varphi$.

2.2 Alternating Tree Automata

For an introduction to the theory of automata on infinite trees see [Tho90]. *Alternating automata* on infinite trees generalize nondeterministic tree automata and were first introduced in [MS87]. For simplicity, we refer first to automata over infinite binary trees. Consider a nondeterministic tree automaton $A = \langle \Sigma, Q, \delta, q_0, F \rangle$. The transition relation δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of pairs of states. Each such pair suggests a nondeterministic choice for the automaton's next configuration. When the automaton is in a state q and is reading a node x labeled by a letter σ, it proceeds by first choosing a pair $\langle q_1, q_2 \rangle \in \delta(q, \sigma)$ and then splitting into two copies. One copy enters the state q_1 and proceeds to the node $x \cdot 0$ (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 D, let $\mathcal{B}^+(D \times Q)$ be the set of positive Boolean formulas over $D \times Q$ (i.e., Boolean formulas built from elements in $D \times Q$ using $\wedge$ and $\vee$), where we also allow the formulas $\mathbf{true}$ and $\mathbf{false}$ and, as usual, $\wedge$ has precedence over $\vee$. We can represent δ 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 $\delta(q, \sigma) = (0, q_1) \wedge (1, q_2) \vee (0, q_3) \wedge (1, q_1)$.

In nondeterministic tree automata, each conjunction in δ has exactly one element associated with each direction. In alternating automata on binary trees, $\delta(q, \sigma)$ can be an arbitrary formula from $\mathcal{B}^+(\{0,1\} \times Q)$. 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, an *alternating tree automaton* is a tuple $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ where Σ is the input alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, F specifies the acceptance condition, and $\delta : Q \times \Sigma \rightarrow \mathbb{N}^+ \times \mathcal{B}^+(\mathbb{N} \times Q)$ is the transition function. We require that if $\delta(q, \sigma) = \langle k, \theta \rangle$, then $\theta \in \mathcal{B}^+(\{0, \ldots, k-1\} \times Q)$. In other words, a transition specifies a branching degree and a matching Boolean transition. A transition can only be applied to a node of a tree with a branching degree equal to the one specified by the transition. A run r of an alternating automaton A on a tree T is a tree where the root is labeled by q_0 and every other node is labeled by an element of $\mathbb{N} \times Q$. Each node of r corresponds to a node of T. Suppose that the path to a node y in r is labeled by $q_0, (c_1, q_1), \ldots, (c_m, q_m)$, then y corresponds to the node $x = c_1 \cdots c_m$ of T. Intuitively, the node y of r describes the automaton reading the node x of T. Note that many nodes of r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on T 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. The run is accepting if all its infinite paths satisfy the acceptance condition. For formal details see [MS87] (our definition here extends the definition in [MS87] by allowing trees with varying branching degrees).

Amorphous automata extend conventional tree automata in that they can handle trees with both varying and unspecified branching degrees. Their amorphous nature enables them to be adjusted during their run to any branching degree. Amorphous automata were first introduced in [BG93], where they extend nondeterministic Büchi tree automata. We introduce here an amorphous version of alternating automata. In a standard alternating automaton, each transition is a pair consisting of a branching degree k and a Boolean formula in $\mathcal{B}^+(\{0,\ldots,k-1\} \times Q)$. In amorphous alternating automata, the transition depends on the (unknown in advance) branching degrees of the nodes being read by the automaton. Formally, the transition is a function $\delta : Q \times \Sigma \times \mathbb{N}^+ \rightarrow \mathcal{B}^+(\mathbb{N} \times Q)$, such that $\delta(q, \sigma, k) \in \mathcal{B}^+(\{0,\ldots,k-1\} \times Q)$. The notion of runs of amorphous alternating automata is a natural extension of the notion of runs of alternating automata. When the automaton is in a state q as it reads a node x that is labeled by a letter σ and has k successors, it applies the transition $\delta(q, \sigma, k)$.

In [MSS86], Muller et al. introduce *weak alternating automata* (WAA). The definition applies also to amorphous alternating automata. In an (amorphous) WAA, $F \subseteq Q$ and there exists a partition of Q into disjoint sets, Q_i, such that for each set Q_i, either $Q_i \subseteq F$, in which case Q_i is an *accepting set*, or $Q_i \cap F = \phi$, 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 $k \in \mathbb{N}^+$, $Q_j \leq Q_i$. 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 iff Q_i is an accepting set.

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 $K \models \psi$. We solve this problem as follows. A Kripke structure K can be viewed as a tree, T_K, that corresponds to the unwinding of K from w^0. Let ψ be a branching temporal formula. Suppose that A_ψ is an amorphous alternating automaton that accepts exactly all the trees that satisfy ψ (amorphousness is used to handle the unspecified branching degrees of the nodes in the models of ψ and alternation reduces the size of the state set of A_ψ from exponential to linear in the length of ψ). The product of K and A_ψ either contains a single tree, T_K, in which case $K \models \psi$, or is empty, in which case $K \not\models \psi$. The model-checking problem can thus be solved as follows:

(1) Construct the amorphous alternating automaton A_ψ.
(2) Construct an alternating automaton $A_{K,\psi} = K \times A_\psi$ by taking the product of K and A_ψ. This automaton simulates a run of A_ψ on T_K.
(3) Output "Yes" if $\mathcal{L}(A_{K,\psi}) \neq \phi$, and "No", otherwise.

The type of A_ψ (i.e., its acceptance condition, its weakness, etc.) as well as the type of $A_{K,\psi}$ and consequently the complexity of the nonemptiness test depend on the logic in which ψ is specified. The crucial point in our approach is that the automaton $A_{K,\psi}$ is an automaton over a 1-letter alphabet; this reduces the complexity of the nonemptiness test. Note that in general, unlike the case for nondeterministic automata, the nonemptiness problem for alternating automata cannot be reduced to the 1-letter emptiness problem. It is taking the product with K that yields here an automaton over a 1-letter alphabet.

Let ψ be a branching temporal formula. We associate with ψ an amorphous alternating automaton $A_\psi = \langle 2^{AP}, cl(\psi), \delta_\psi, \psi, F_\psi \rangle$, which accepts precisely all the tree models of ψ. The details of the construction depends on the logic under consideration; we will see some examples in the next section. Let $K = \langle W, R, w^0, L \rangle$ be a Kripke structure. It is convenient to assume that the nodes of W are ordered. Thus, for every node w, let $succ_R(w) = \langle w_0, \ldots, w_{k-1} \rangle$ be the ordered list of w's R-successors.

We now define the product automaton. $A_{K,\psi} = \langle \{a\}, W \times cl(\psi), \langle w^0, \psi \rangle, \delta, F \rangle$ where δ and F are defined as follows.

- Let $\varphi \in cl(\psi)$, $w \in W$, $succ_R(w) = \langle w_0, \ldots, w_{k-1} \rangle$, and $\delta_\psi(\varphi, L(w), k) = \theta$. Then $\delta(\langle w, \varphi \rangle, a) = \langle k, \theta' \rangle$, where θ' is obtained from θ by replacing each atom (c, η) in θ by the atom $(c, \langle w_c, \eta \rangle)$.
- F is defined according to the acceptance condition F_ψ of A_ψ. For example, if $F_\psi \subseteq cl(\psi)$ is a Büchi condition, then $F = W \times F_\psi$ is also a Büchi condition. If $F_\psi = \{\langle L_1, U_1 \rangle, \ldots, \langle L_m, U_m \rangle\}$ is a Rabin condition, then $F = \{\langle W \times L_1, W \times U_1 \rangle, \ldots, \langle W \times L_m, W \times U_m \rangle\}$ is also a Rabin condition.

It is easy to see that $A_{K,\psi}$ is of the same type as A_ψ. In particular, if A_ψ is a WAA, then so is $A_{K,\psi}$.

Proposition 1. **(1)** $|A_{K,\psi}| = O(|K| * |A_\psi|)$.
(2) *$\mathcal{L}(A_{K,\psi})$ is nonempty iff $K \models \psi$.*

Proposition 1 can be viewed as an automata-theoretic generalization of Theorem 4.1 in [EJS93].

In conclusion, given a branching temporal formula ψ for which there exists an automaton A_ψ such that A_ψ accepts exactly all the trees that satisfy ψ, model checking of a Kripke structure K with respect to ψ, is reducible to checking the 1-letter emptiness of an automaton of the same type as A_ψ and of size $O(|K| * |A_\psi|)$. In the following sections, we show how this approach can be used to derive in a uniform way known coplexity bounds for model checking of CTL and μ-calculus formulas, as well as to obtain new space complexity bounds.

4 Applications

The efficiency of the method we presented in the previous section depends on the efficiency of the translation of formulas to automata, as well as the efficiency of the 1-letter nonemptiness test.

4.1 Model checking for CTL

Vardi and Wolper showed how to solve the satisfiability problem for CTL via an exponential translation of CTL formulas to Büchi automata on infinite trees [VW86a]. Müller et al. provided a simpler proof, via a linear translation of branching dynamic logic formulas to WAA [MSS88]. We extend here the ideas of Muller et al. by demonstrating a linear translation from CTL formulas to amorphous WAA.

Theorem 2. *Given a CTL formula ψ, we can construct in linear running time an amorphous WAA $A_\psi = \langle 2^{AP}, cl(\psi), \rho, \psi, F\rangle$ such that $\mathcal{L}(A_\psi)$ is exactly the set of tree models satisfying ψ.*

Proof. The set F of accepting states consists of all formulas in $cl(\psi)$ of the form $A\varphi_1\tilde{U}\varphi_2$ or $E\varphi_1\tilde{U}\varphi_2$. It remains to define the transition function ρ.

- $\rho(p, \sigma, k) = \mathbf{true}$ if $p \in \sigma$. – $\rho(p, \sigma, k) = \mathbf{false}$ if $p \notin \sigma$.
- $\rho(\neg p, \sigma, k) = \mathbf{true}$ if $p \notin \sigma$. – $\rho(\neg p, \sigma, k) = \mathbf{false}$ if $p \in \sigma$.
- $\rho(\varphi_1 \wedge \varphi_2, \sigma, k) = \rho(\varphi_1, \sigma, k) \wedge \rho(\varphi_2, \sigma, k)$.
- $\rho(\varphi_1 \vee \varphi_2, \sigma, k) = \rho(\varphi_1, \sigma, k) \vee \rho(\varphi_2, \sigma, k)$.
- $\rho(AX\varphi_2, \sigma, k) = \bigwedge_{c=0}^{k-1}(c, \varphi_2)$.
- $\rho(EX\varphi_2, \sigma, k) = \bigvee_{c=0}^{k-1}(c, \varphi_2)$.
- $\rho(A\varphi_1 U\varphi_2, \sigma, k) = \rho(\varphi_2, \sigma, k) \vee (\rho(\varphi_1, \sigma, k) \wedge \bigwedge_{c=0}^{k-1}(c, A\varphi_1 U\varphi_2))$.
- $\rho(E\varphi_1 U\varphi_2, \sigma, k) = \rho(\varphi_2, \sigma, k) \vee (\rho(\varphi_1, \sigma, k) \wedge \bigvee_{c=0}^{k-1}(c, E\varphi_1 U\varphi_2))$.
- $\rho(A\varphi_1 \tilde{U}\varphi_2, \sigma, k) = \rho(\varphi_2, \sigma, k) \wedge (\rho(\varphi_1, \sigma, k) \vee \bigwedge_{c=0}^{k-1}(c, A\varphi_1 \tilde{U}\varphi_2))$.
- $\rho(E\varphi_1 \tilde{U}\varphi_2, \sigma, k) = \rho(\varphi_2, \sigma, k) \wedge (\rho(\varphi_1, \sigma, k) \vee \bigvee_{c=0}^{k-1}(c, E\varphi_1 \tilde{U}\varphi_2))$.

To show that A_ψ is a WAA, we define a partition of Q into disjoint sets and a partial order over the sets. 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 describe an efficient algorithm to test 1-letter nonemptiness of WAA.

Theorem 3. *The 1-letter nonemptiness problem for weak alternating automata is decidable in linear running time.*

Proof. See Appendix A.1

Theorems 2 and 3 yield a model-checking algorithm for CTL with linear (in the size of the input structure and in the size of the input formula) running time. The bottom-up labeling of the algorithm used in the proof of Theorem 3 is clearly reminiscent of the bottom-up labeling that takes place in the standard algorithm for CTL model checking [CES86]. Thus, the automata-theoretic approach seems to capture the combinatorial essence of CTL model checking.

4.2 Model Checking for the μ-Calculus

The intimate connection between the μ-calculus and alternating automata has been noted in [EJ91, Eme94]. We show here that our automata-theoretic approach provides a clean proof that model checking for the μ-calculus is in NP∩co-NP. The key steps in the proof are in showing that μ-calculus formulas can be efficiently translated to amorphous alternating Rabin automata, and that the 1-letter nonemptiness problem for alternating Rabin automata is in NP.

Theorem 4. *Given a μ-calculus formula ψ, we can construct in linear running-time an amorphous alternating Rabin automaton $A_\psi = \langle 2^{AP}, cl(\psi), \rho, \psi, F\rangle$ such that $\mathcal{L}(A_\psi)$ is exactly the set of tree models satisfying ψ.*

Proof. Emerson and Jutla showed how to translate μ-calculus formulas to alternating *Streett* automata [EJ91]. The extension to amorphous automata is straightforward. By constructing an amorphous alternating Streett automaton for $\neg\psi$ and then complementing it (it is easy to complement alternating automata [MS87]), we obtain an amorphous alternating Rabin automaton.

Theorem 5. *The 1-letter nonemptiness problem for alternating Rabin automata is decidable in nondeterministic polynomial running time.*

Proof. Without loss of generality we can assume that we are dealing with automata on trees of fixed branching degree, say k. Since the 1-letter k-ary tree is homogeneous (i.e., all subtrees are the same), we can pretend that successor states which are going down the same branch of the tree, are actually going down separate branches. Thus, we can apply techniques from the theory of nondeterministic Rabin automata, developed in [Eme85, VS85], to show that the 1-letter nonemptiness problem is in NP.

Combining Theorems 4 and 5, Proposition 1, and the observation in [EJS93] 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.

For the alternation-free μ-calculus, we can prove an analogue to Theorem 2. It follows then from Theorem 3, that model checking for the alternation-free μ-calculus can be done in linear running time.

5 The Space Complexity of Model Checking

Pnueli and Lichtenstein 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 size of the structure that is typically the computational bottleneck [LP85]. 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 [VW86a]. Indeed, this problem reduces to checking the emptiness of a Büchi automaton on words which is NLOGSPACE-complete. Thus, if the Büchi automaton whose 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 will be 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 [CVWY92].

Is the same true of CTL 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 CTL model-checking, we are able to show that this is not so. Technically, this means that we will now prove that model-checking for CTL is NLOGSPACE-complete in the size of the Kripke structure. To prove this result, we will first show that the 1-letter WAA we construct for CTL model-checking have a special property (bounded alternation). Then, we will present an alternative algorithm for checking emptiness of WAA with this property.

Consider the product automaton $A_{K,\psi} = K \times A_\psi$ for a Kripke structure K and CTL formula ψ. The states of this automaton are elements of $W \times cl(\psi)$ and are partitioned into subsets Q_i according to their second component (two states are in the same Q_i 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 one examines the Q_i's closely, one notices 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. We call these *transient* Q_i's.
2. Sets Q_i such that, for all $q \in Q_i$, the transition $\delta(q, a, k)$ only contain 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 $cl(\psi)$. We call these *universal* Q_i's.
3. Sets Q_i such that, for all $q \in Q_i$, the transition $\delta(q, a, k)$ only contain disjunctively related elements of Q_i. 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 $cl(\psi)$. We call these *existential* Q_i's.

This means that it is only when moving from one Q_i to the next, that alternation actually occurs (alternation is moving from a state that is conjunctively related to its siblings to a state that is disjunctively related to its siblings, or vice-versa). If the number of Q_i's is fixed and if the depth of transitions is bounded (i.e., if their parse tree has bounded depth), we call a WAA that satisfies this property a *bounded-alternation* WAA.

Let us now turn to the nonemptiness problem for bounded-alternation WAA. Theorem 3 shows that the problem can be solved in linear running time. Notice that the algorithm used there is essentially a bottom-up labeling of the Boolean graph of the automaton. We will now show that by using a top-down exploration of this Boolean graph, we can get a space efficient 1-letter nonemptiness algorithm for bounded-alternation WAA.

Theorem 6. *The 1-letter nonemptiness problem for bounded-alternation WAA is NLOGSPACE-complete.*

Proof. See Appendix A.2

We note that for general WAA the 1-letter nonemptiness problem is P-complete.

Now, let us define the structure complexity of model-checking as the complexity of this problem in terms of the size of the input Kripke structure, i.e. assuming the formula fixed (this was called *program complexity* in [VW86a]). The following is then a direct consequence of Theorem 6.

Theorem 7. *The structure complexity of CTL model-checking is NLOGSPACE-complete.*

Theorem 7 can be extended to ECTL* [VW84]. The alternating automata that correspond to ECTL* formula are not in general weak. Nevertheless, a careful analysis shows that these automata do have a special structure and Theorem 6 can be extended to such automata.

If the Kripke structure is obtained as the product of the components of a concurrent program, this implies that CTL (and ECTL*) model-checking can be done in polynomial space with respect to the size of this program. It is also interesting to note that a less space-efficient deterministic version of the algorithm given in the proof of Theorem 6 can be viewed as the automata-theoretic counterpart of the algorithm presented in [VL93].

References

[Bee80] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5:241–259, 1980.

[BG93] O. Bernholtz and O. Grumberg. Branching time temporal logic and amorphous tree automata. In *Proc. 4th Conferance on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277, Hildesheim, August 1993. Springer-Verlag.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[Cle93] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[EJ88] E. A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, oct 1988.

[EJ91] E. A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Oct 1991.

[EJS93] E. A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Workshop*, volume 697, pages 385–396, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer-Verlag, 1985.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, pages 997–1072, 1990.

[Eme94] E.A. Emerson. Automated temporal reasoning about reactive systems. In *VIII-th BANFF Higher Order Workshop*, 1994. unpublished abstract of forthcoming talk.

[ES84] A.E. Emerson and A.P. Sistla. Deciding full branching time logics. *Information and Control*, 61(3):175–201, 1984.

[JJ89] C. Jard and T. Jeron. On-line model-checking for finite linear temporal logic specifications. In *Automatic Verification Methods for Finite State Systems, Proc. Int. Workshop, Grenoble*, volume 407, pages 189–196, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, January 1980.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54,:267–276, 1987.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming*. Springer-Verlag, 1986.

[MSS88] D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science*, pages 422–427, Edinburgh, July 1988.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.

[SE84] R. S. Street and E. A. Emerson. An elementary decision procedure for the mu-calculus. In *Proc. 11th Int. Colloquium on Automata, Languages and Programming*, volume 172. Lecture Notes in Computer Science, Springer-Verlag, July 1984.

[SW91] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of theoretical computer science*, pages 165–191, 1990.

[Var88] M.Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. on Principles of Programming Languages*, pages 250–259, San Diego, January 1988.

[VL93] B. Vergauwen and J. Lewi. A linear local model checking algorithm for ctl. In *Proc. CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 447–461, Hildesheim, August 1993. Springer-Verlag.

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.

[VW84] M.Y. Vardi and P. Wolper. Yet another process logic. In *Logics of Programs*, volume 398, pages 501–512. Lecture Notes in Computer Science, Springer-Verlag, 1984.

[VW86a] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[VW86b] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–21, April 1986.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 110(2), May 1994. to appear.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[Wol89] P. Wolper. On the relation of programs and computations to models of temporal logic. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Temporal Logic in Specification*, volume 398, pages 75–123. Lecture Notes in Computer Science, Springer-Verlag, 1989.

A Proofs

A.1 Theorem 3

We present an algorithm with linear running time for checking the nonemptiness of the language of a WAA $A = \langle \{a\}, Q, \delta, q_0, F \rangle$.

As 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 $\delta(q, a)$, $Q_j \leq Q_i$. 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 $Q_i \subseteq F$, or rejecting, if $Q_i \cap F = \phi$.

The algorithm labels the states of A with either 'T', standing for **true**, or 'F', standing for **false**. Intuitively, states $q \in Q$ for which the language of A^q (i.e., the language of A with q as the initial state) is nonempty are labeled with 'T' and states q for which the language of A^q is empty are labeled with 'F'. The language of A is thus nonempty iff the initial state q_0 is labeled with 'T'. The algorithm works in phases and proceeds up the partial order. Let $Q_1 \leq \ldots \leq Q_n$ 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 which still has not been labeled.

States that belong to a set Q_i that is minimal in the partial order, are labeled according to the classification of Q_i. Thus, they are labeled with 'T' if Q_i is an accepting set, and with 'F' if it is rejecting. Once a state $q \in Q_i$ is labeled with 'T' or 'F', transition functions in which q occurs are simplified accordingly, i.e., 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 obtain its truth value. q' is then labeled, and evaluation proceeds 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.

Using an AND/OR graph, as suggested in [Bee80], the algorithm can be implemented in linear running time. Typically, the graph, induced by the transition function, keeps the labeling performed during the algorithm execution. Simplification of each transition function $\delta(q, a)$ for all $q \in Q$, then costs $O(|\delta(q, a)|)$.

A.2 Theorem 6

The property of bounded-alternation WAA we use is that, from a state of a Q_i, it is possible to search for another reachable state of the same Q_i in NLOGSPACE. For 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. Assume, we have a Boolean value for all states in sets lower than Q_i. Then a state q' is immediately reachable from a state q, if it appears in the transition from q when this transition has been simplified using the know Boolean values for states in lower Q_i. Note that the simplified transition is always a conjunction for a state of a universal Q_i, and a disjunction for a state of an existential Q_i.

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

1. One starts at the initial state.
2. At a transient state q, one applies the procedure to the successor states. The labels that are obtained for these successor states are then substituted in the transition from q, and q is labeled with the Boolean value that is thus obtained for the transition.
3. At a state q of a universal Q_i, one proceeds as follows. 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.
   (a) One searches in NLOGSPACE for a reachable state q' of the same Q_i that is provably false (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, the state q is labeled 'F'.
   (b) If no such state exists, one searches in NLOGSPACE for a state q' of Q_i that is reachable from q and from itself. If such a state is found, q is labeled according to the classification of the Q_i.
   (c) if none of the first two cases apply, q is labeled 'T'.
4. At a state q of an existential Q_i, one proceeds as follows.
   (a) One searches in NLOGSPACE for a reachable state q' of the same Q_i that is provably true. If such a state q' is found, the state q is labeled 'T'.
   (b) If no such state exists, one searches in NLOGSPACE for a state q' of Q_i that is reachable from q and from itself. If such a state is found, q is labeled according to the classification of the Q_i.
   (c) if none of the first two cases apply, q is labeled 'F'.

The procedure is recursive, but as the depth of the transitions is bounded by the number of Q_i's, so does the depth of recursion. Since each invocation of the procedure can be executed in NLOGSPACE, the whole procedure is thus NLOGSPACE. Completeness in NLOGSPACE is immediate by reduction from the corresponding problem for nondeterministic sequential automata.

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