Freedom, Weakness, and Determinism: From Linear-time to Branching-time

Orna Kupferman[*]
UC Berkeley

Moshe Y. Vardi[†]
Rice University

Abstract Model checking is a method for the verification of systems with respect to their specifications. Symbolic model-checking, which enables the verification of large systems, proceeds by calculating fixed-point expressions over the system's set of states. The μ-calculus is a branching-time temporal logic with fixed-point operators. As such, it is a convenient logic for symbolic model-checking tools. In particular, the alternation-free fragment of μ-calculus has a restricted syntax, making the symbolic evaluation of its formulas computationally easy. Formally, it takes time that is linear in the size of the system. On the other hand, specifiers find the μ-calculus inconvenient. In addition, specifiers often prefer to use linear-time formalisms. Such formalisms, however, cannot in general be translated to the alternation-free μ-calculus, and their symbolic evaluation involves nesting of fixed-points, resulting in time complexity that is quadratic in the size of the system. In this paper we characterize linear-time properties that can be specified in the alternation-free μ-calculus. We show that a linear-time property can be specified in the alternation-free μ-calculus iff it can be recognized by a deterministic Büchi automaton. We study the problem of deciding whether a linear-time property, specified by either an automaton or an LTL formula, can be translated to an alternation-free μ-calculus formula, and describe the translation, when exists.

[*]Address: EECS Department, UC Berkeley, Berkeley CA 94720-1770, U.S.A. Email: orna@eecs.berkeley.edu, URL: http://www.eecs.berkeley.edu/~orna. Supported in part by ONR YIP award N00014-95-1-0520, by NSF CAREER award CCR-9501708, by NSF grant CCR-9504469, by AFOSR contract F49620-93-1-0056, by ARO MURI grant DAAH-04-96-1-0341, by ARPA grant NAG2-892, and by the SRC contract 95-DC-324.036.

[†]Address: CS Department, Rice University, Houston TX 77005-1892, U.S.A. Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi. Supported in part by NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation.

1 Introduction

The importance of verifying the correctness of hardware and software designs dates back to the early realization of the prevalence of design errors, i.e., "bugs". While testing has once been considered a satisfying method for detecting bugs, today's rapid development of complex and safety-critical systems requires more reliable methods. *Model checking* is such a more reliable method. In model checking [CE81, CES86, QS81, LP85, VW86], we check that a system meets a desired requirement by checking that a mathematical model of the system satisfies a formal specification that describes the requirement. The algorithmic nature of model checking makes it fully automatic, convenient to use, and very attractive to practitioners. At the same time, model checking is very sensitive to the size of the mathematical model of the system. Commercial model-checking tools need to cope with the exceedingly large state-spaces that are present in real-life designs, making the so-called *state-explosion problem* one of the most challenging areas in computer-aided verification. One of the most important developments in this area is the discovery of *symbolic* model-checking methods [BCM$^+$92, McM93]. In particular, the use of BDDs [Bry86] for model representation has yielded model-checking tools that can handle systems with 10^{120} states and beyond [CGL93].

Typically, symbolic model-checking tools proceed by computing fixed-point expressions over the model's set of states. For example, to find the set of states from which a state satisfying some predicate p is reachable, the model checker starts with the set S of states in which p holds, and repeatedly add to S the set $\exists \bigcirc S$ of states that have a successor in S. Formally, the model checker calculates the least fixed-point of the expression $S = p \vee \exists \bigcirc S$. The *$\mu$-calculus* is a logic that contains the modal operators $\exists \bigcirc$ and $\forall \bigcirc$, and the fixed-point operators μ and ν [Koz83]. As such, it describes fixed-point computations in a natural form. In particular, the *alternation-free* fragement of μ-calculus (AFMC, for short) [EL86] has a restricted syntax that does not allow nesting of fixed-point operators, making the evaluation of expressions very simple. Formally, the model-checking problem for AFMC can be solved in time that is linear in both the size of the model and the length of the formula [CS91]. The μ-calculus, however, is less ideal for specifiers. For them, logics with explicit temporal operators, such as $\Box$ ("always") and $\Diamond$ ("eventually") are much more readable and convenient. Consequently, model-checking tools often offer as their user interface a temporal logic that includes explicit temporal operators. The evaluation of these formulas by means of fixed points is then transparent to the user.

When, as in the model-checking tools SMV and VIS [McM93, BHSS$^+$96], the interface logic is the *branching-time* temporal logic CTL, the transition from the input formulas to fixed-point expressions is simple: each operator of CTL can be expressed also be means of fixed points. Formally, one can translate a CTL formula to an AFMC formula with a linear blow up. For example, the CTL formula $\forall \Box \exists \Diamond p$ is equivalent to the AFMC formula $\nu X.(\mu Y.p \vee \exists \bigcirc Y) \wedge \forall \bigcirc X$. Designers, however, often prefer to specify their systems using linear-time formalisms. Indeed, model-checking tools such as COSPAN, SPIN, and VIS [Kur94, HHK96, Hol97, BHSS$^+$96] handle specifications that are given as automata on infinite words or LTL formulas. Since linear-time formalisms such as LTL can express properties that are not expressible in AFMC (e.g., it follows from the results in [Rab70, MSS86, BVW94], that the LTL formula $\Diamond \Box p$ is not expressible in AFMC), symbolic model-checking methods become more complicated. For example, symbolic model checking of LTL involves a translation of LTL formulas to μ-calculus formulas of alternation depth 2, where nesting of fixed-point operators is allowed [EL86]. The evaluation of such μ-calculus formulas takes time that is quadratic in the size of the model. Since the models are very large, the difference with the linear complexity of AFMC is very significant [HKSV97].

In this paper we consider the problem of translating linear-time formalisms to AFMC; formally, we characterize ω-regular languages for which there exist equivalent AFMC formulas. Note that while each ω-regular language $\mathcal{L}$ describes a set of infinite words, each AFMC formula ψ describes a set of infinite trees. When we say that ψ is equivalent to $\mathcal{L}$, we mean that ψ is satisfied in exactly these trees all of whose paths are in $\mathcal{L}$. The ω-regular languages can be specified by either an automaton on infinite words or an LTL formula. We consider the problem of deciding whether a given automaton or formula meets this characterization, and the problem of translating a given automaton or formula to an equivalent AFMC formula when it exists.

Beyond the relevance of this problem to symbolic model checking, the study of the relationship between linear-time and branching-time formalisms goes back to the 1980's. In [CD88], Clarke and Draghicescu characterized CTL formulas that can be translated to LTL. The opposite direction, of characterizing LTL formulas that can be translated to CTL, turned out to be much harder and the problem has stayed open since then. The computational advantage of CTL model checking over LTL model checking makes this opposite direction the more interesting one. Indeed, a translation of LTL formulas to CTL formulas could be used in order to model check linear-time properties using CTL model-checking tools. A partial success for the above approach is presented in [KG96, Sch97], which identify certain fragments of LTL that can be easily translated to CTL. By characterizing LTL formulas that can be translated to AFMC, we solve a closely related problem. In fact, since symbolic CTL model checkers proceed by translating the specification to AFMC, our characterization is the more interesting one from a practical point of view.

In order to characterize ω-regular languages that can be translated to AFMC, we first characterize AFMC by means of tree automata. We show that a branching-time property can be specified in AFMC iff it can be specified by a *weak alternating tree automaton.* Weak alternating tree automata were first introduced in [MSS86], where they were related to weakly definable sets [Rab70]. The relevance of weak alternating automata to model checking was demonstrated in [BVW94]. The equivalence of AFMC and weak alternating automata is proved also in [AN92], where both formalisms are shown to be equivalent to the weak monadic second-order theory of trees [Rab70, MSS86]. Our proof is simpler and direct, and we describe linear translations between the two formalisms. We then use known relations between automata on infinite words and trees [KSV96] in order to show that an ω-regular language $\mathcal{L}$ can be translated to an AFMC formula iff $\mathcal{L}$ can be recognized by a deterministic Büchi word automaton [Büc62].

It follows from our results that deciding whether a linear-time property P can be translated to an AFMC formula can be reduced to the problem of deciding whether P can be recognized by a deterministic Büchi word automaton. The complexity of this problem depends on the form in which P is given. We show that the problem is NLOGSPACE-complete when P is given as a deterministic parity automaton, is PTIME-complete when P is given as a deterministic Rabin or Streett automaton, and is PSPACE-complete when P is given as a nondeterministic Büchi, parity, Rabin, or Streett automaton. When P is given as an LTL formula, the problem is in EXPSPACE and is PSPACE-hard. We then turn to consider the relative succinctness of the various formalisms. We describe translations to AFMC that is linear for deterministic automata, exponential for nondeterministic automata, and doubly exponential for LTL formula (when such translations exist), and we prove that the translation from LTL formulas must involve at least an exponential blow-up. It should be noted that the translation of LTL formulas to μ-calculus formulas of alternation depth 2 involves only a single exponential blow up [EL86]. Typically, however, the computational complexity of model checking is dominated by the size of the model, rather than the size of the specification. As model checking μ-calculus formulas of alternation depth 2 is quadratic in the size of the model, while model checking of AFMC formulas is linear in the size of the model, we conclude that symbolic model checking of LTL formulas by first translating them to AFMC might be a practical approach. Hopefully, our doubly exponential translation could be improved to an exponential one, matching our lower bound.

2 Definitions

2.1 Temporal logics

A *system* $S = \langle P, W, W_{in}, R, L\rangle$ consists of a set P of atomic propositions, a set W of states, an initial state $w_{in} \in W$, a total transition relation $R \subseteq W \times W$ (i.e., for every state $w \in W$, there exists at least w' with $R(w, w')$), and a labeling function $L : W \to 2^P$. A *computation* of S is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have that $R(w_i, w_{i+1})$. The computation π is *initial* if $w_0 = w_{in}$.

Formulas of the linear-time temporal logic *LTL* describe computations of systems. Given a set P of atomic propositions, an LTL formula is p, $\neg\varphi$, $\varphi \vee \psi$, $\Box\varphi$, $\Diamond\varphi$, $\bigcirc\varphi$, or $\varphi U \psi$, where $p \in P$, and φ and ψ are LTL formulas. The temporal operators $\Box$ ("always"), $\Diamond$ ("eventually"), $\bigcirc$ ("next"), and U ("until") enable convenient description of time-dependent events. For example, the LTL formula

$\Box(request \rightarrow \Diamond grant)$ states that every request is followed by a grant. For the full definition of LTL see [Pnu81].

The *alternation-free μ-calculus* (*AFMC*, for short) is a fragment of the modal logics μ-calculus [Koz83]. We define the AFMC by means of equational blocks, as in [CS91]. Formulas of AFMC are defined with respect to a set P of atomic propositions and a set Var of atomic variables. A *basic* AFMC formula is either p, X, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\forall \bigcirc \varphi$, or $\exists \bigcirc \varphi$, for $p \in P$, $X \in Var$, and basic AFMC formulas φ and ψ. The semantics of the basic formulas is defined with respect to a system $S = \langle P, W, W_{in}, R, L \rangle$ and a *valuation* $\mathcal{V} = \{\langle X_1, W_1 \rangle, \ldots, \langle X_n, W_n \rangle\}$ that assigns subsets of W to the variables in Var. Each basic AFMC formula defines a subset of the states of S in the standard way. We denote by $\varphi^S_{\mathcal{V}}$ the set of states define by φ under the evaluation $\mathcal{V}$. For example, $p^S_{\mathcal{V}} = \{w \in W : p \in L(w)\}$, ${X_i}^S_{\mathcal{V}} = W_i$, $(\varphi \vee \psi)^S_{\mathcal{V}} = \varphi^S_{\mathcal{V}} \cup \psi^S_{\mathcal{V}}$, and $(\exists \bigcirc \varphi)^S_{\mathcal{V}} = \{w \in W : \exists w' \in \varphi^S_{\mathcal{V}} \text{ and } R(w, w')\}$. An *equational block* has two forms, $\nu\{E\}$ or $\mu\{E\}$, where E is a list of equations of the form $X_i = \varphi_i$, where φ_i is a basic AFMC formula and the X_i are all different atomic variables. An atomic variable X that appears in the right-hand size of an equation in some block B may appear in the left-hand side of an equation in some other block B'. We then say that X is *free* in B and that B *depends* on B'. Such dependencies cannot be circular. This ensures that the formula is free of alternations. The semantics of an equational block is defined with respect to a system S and a valuation $\mathcal{V}$ that assigns subsets of W to all the free variables in the block. A block of the form $\nu\{E\}$ represents the greatest fixed-point of E and a block of the form $\mu\{E\}$ represents the least fixed-point of E. For example, $\nu\{X = p \wedge \exists \bigcirc X\}$ defines the set of states in S from which there exists a computation in which p always holds. For a set of blocks, evaluation proceeds so that whenever a block B is evaluated, all the blocks B' for which B depends on B' are already evaluated, thus all the free variables in B have values in $\mathcal{V}$. For the full definition see [CS91].

Given a system S and an LTL formula φ, the model-checking problem for S and φ is to determine whether all the initial computations of S satisfy φ. When φ is an AFMC formula with no free variables, the problem is to determine whether all the initial states of S satisfy φ, that is, whether $w_{in} \in \varphi^S_{\emptyset}$.

2.2 Automata

For an integer $d \geq 1$, let $[d] = \{1, \ldots, d\}$. An *infinite d-tree* is the set $T = [d]^*$. The elements of $[d]$ are *directions*, the elements of T are *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$, for $c \in [d]$, are the *successors* of x. A *path* of T is a set $\rho \subseteq T$ such that $\epsilon \in \rho$ and for each $x \in \rho$, exactly one successor of x is in ρ. Given an alphabet Σ, a *Σ-labeled d-tree* is a pair $\langle T, V \rangle$, where T is a d-tree and $V : T \rightarrow \Sigma$ maps each node of T to a letter in Σ. A Σ-labeled 1-tree is a *word* over Σ. For a language $\mathcal{L}$ of words over Σ, the *derived language* of $\mathcal{L}$, denoted $der(\mathcal{L})$ is the set of all Σ-labeled trees all of whose paths are labeled by words in $\mathcal{L}$. For $d \geq 2$, we denote by $der_d(\mathcal{L})$ the set of Σ-labeled d-trees in $der(\mathcal{L})$. For a system S with branching degrees in d, we denote by $tree(S)$ the 2^P-labeled d-tree obtained by unwinding S from its initial state.

An *alternating tree automaton* [MS87] $\mathcal{A} = \langle \Sigma, d, Q, q_0, \delta, \alpha \rangle$ runs on Σ-labeled d-trees. It consists of a finite set Q of states, an initial state $q_0 \in Q$, a transition function δ, and an acceptance condition α (a condition that defines a subset of Q^ω). For the set $[d]$ of directions, 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 **true** and **false**. The transition function $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+([d] \times Q)$ maps a state and an input letter to a formula that suggests a new configuration for the automaton. For example, when $d = 2$, having

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

means that when the automaton is in state q and reads the letter σ, it can either send two copies, in states q_1 and q_2, to direction 1 of the tree, or send a copy in state q_2 to direction 1 and two copies, in states q_2 and q_3, to direction 2. Thus, the transition function may require the automaton to send several copies to the same direction or allow it not to send copies to all directions.

A *run* of an alternating automaton $\mathcal{A}$ on an input Σ-labeled d-tree $\langle T, V\rangle$ is a labeled tree $\langle T_r, r\rangle$ (without a fixed branching degree) in which the root is labeled by q_0 and every other node is labeled by an element of $[d]^* \times Q$. 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. The labels of a node and its children have to satisfy the transition function. For example, if $\langle T, V\rangle$ is a 2-tree with $V(\epsilon) = a$ and $\delta(q_0, a) = ((1, q_1) \vee (1, q_2)) \wedge ((1, q_3) \vee (2, q_2))$, then the nodes of $\langle T_r, r\rangle$ at level 1 include the label $(1, q_1)$ or $(1, q_2)$, and include the label $(1, q_3)$ or $(2, q_2)$. Each infinite path ρ in $\langle T_r, r\rangle$ is labeled by a word $r(\rho)$ in Q^ω. Let $\mathit{inf}(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r\rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. We consider four types of acceptance conditions:

- *Büchi*, where $\alpha \subseteq Q$, and an infinite path ρ satisfies α iff $\mathit{inf}(\rho) \cap \alpha \neq \emptyset$.
- *parity*, where $\alpha = \{G_1, \ldots, G_k\}$ is a sequence of subsets of Q satisfying $G_i \subseteq G_{i+1}$, and an infinite path ρ satisfies α iff the minimal index i for which $\mathit{inf}(\rho) \cap G_i \neq \emptyset$ is even.
- *Rabin*, where $\alpha \subseteq 2^Q \times 2^Q$, and an infinite path ρ satisfies an acceptance condition $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ iff there exists $1 \leq i \leq k$ for which $\mathit{inf}(\rho) \cap G_i \neq \emptyset$ and $\mathit{inf}(\rho) \cap B_i = \emptyset$.
- *Streett*, where $\alpha \subseteq 2^Q \times 2^Q$, and an infinite path ρ satisfies an acceptance condition $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ iff for all $1 \leq i \leq k$, if $\mathit{inf}(\rho) \cap G_i \neq \emptyset$ then $\mathit{inf}(\rho) \cap B_i \neq \emptyset$.

An automaton accepts a tree iff there exists an accepting run on it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$; i.e., the set of all labeled trees that $\mathcal{A}$ accepts.

When $d = 1$, we say that $\mathcal{A}$ is a *word automaton*, we omit d from the specification of the automaton, and we describe its transitions by formulas in $\mathcal{B}^+(Q)$. We say that $\mathcal{A}$ is a *nondeterministic* automaton iff all the transitions of $\mathcal{A}$ have only disjunctively related atoms sent to the same direction; i.e., if the transitions are written in DNF, then every disjunct contains at most one atom of the form (c, q), for all $c \in [d]$. Note that a transition of nondeterministic word automata is a disjunction of states in Q, and we denote it by a set. We say that $\mathcal{A}$ is a *deterministic* automaton iff all the transitions of $\mathcal{A}$ have only disjunctively related atoms, all sent to different directions.

We denote each of the different types of automata by three letter acronyms in $\{D, N, A\} \times \{B, P, R, S\} \times \{W, T\}$, where the first letter describe the branching mode of the automaton (deterministic, nondeterministic, or alternating), the second letter describes the acceptance condition (Büchi, parity, Rabin, or Streett), and the third letter describes the object over which the automaton runs (words or trees). We use the acronyms also to refer to the set of words (or trees) that can be defined by the various automata. For example, DBW denotes deterministic Büchi word automata, as well as the set of ω-regular languages that can be recognized by a deterministic word automaton. Which interpretation we refer to would be clear from the context.

In [MSS86], Muller et al. introduce *weak alternating tree automata* (AWT). In an AWT, the acceptance condition is $\alpha \subseteq Q$ and there exists a partition of Q into disjoint sets, Q_i, such that for each set Q_i, either $Q_i \subseteq \alpha$, in which case Q_i is an *accepting set*, or $Q_i \cap \alpha = \emptyset$, 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 $\delta(q, \sigma)$ for some $\sigma \in \Sigma$, 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 an AWT ultimately gets "trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set.

3 Freedom, Weakness, and Determinism

In this section we show that the expressive power of the AFMC coincides with that of AWT. We then characterize ω-regular languages $\mathcal{L}$ for which $der(\mathcal{L})$ can be expressed by an AFMC formula.

We start by relating AFMC and AWT. While tree automata run on trees with some finite fixed set of branching degrees and can distinguish between the different successors of a node, AFMC formulas define trees of arbitrary branching degrees and cannot distinguish between different successors. Accordingly, discussion is restricted to trees over some fixed branching degree and the AFMC is *directed* (that is, the next-time operator is annotated with an explicit direction) [HT87]. For every $d \geq 1$, let AWT(d) be the set of AWT (and similarly for other types of tree automata) that contains AWT of branching degree d, and let AFMC(d) be the set of directed AFMC formulas where the next-time operator is annotated by directions in $[d]$. [1]

Theorem 3.1 *For every $d \geq 1$, we have $AWT(d) = AFMC(d)$.*

Proof: A linear translation of AFMC(d) formulas to AWT(d) is given in [BVW94]. For the other direction, we describe a linear translation of AWT(d) to AFMC(d) formulas. The translation is similar to the one described in [BC96], where ABT are translated to μ-calculus formulas of alternation depth 2. Consider an AWT(d) $\mathcal{A} = \langle \Sigma, d, Q, q_0, \delta, \alpha \rangle$. Let $Q_0 \leq Q_1 \leq \cdots \leq Q_n$ be the partition of Q into sets. We define an AFMC(d) formula $\psi_{\mathcal{A}}$ such that for every system S of branching degree d, we have $tree(S) \in \mathcal{L}(\mathcal{A})$ iff S satisfies $\psi_{\mathcal{A}}$. The formula $\psi_{\mathcal{A}}$ is has Σ as its set of atomic propositions. Each state $q \in Q$ induces an atomic variable X_q. Intuitively, once a fixed point is reached, a state w of S is a member of X_q iff the tree obtained from S by unwinding it from w is accepted by the automaton $\mathcal{A}$ with initial state q. Accordingly, the equations for atomic variables follow from the transition function δ. For a formula $\theta \in \mathcal{B}^+([d] \times Q)$, let $f(\theta)$ be the AFMC(d) formula obtained from θ by replacing an atom (c, q) by the assertion $\exists \bigcirc_c X_q$. Each state $q \in Q$ induces the equation

$$X_q = \bigvee_{\sigma \in \Sigma} \sigma \wedge f(\delta(q, \sigma))$$

in $\psi_{\mathcal{A}}$. The set of equations is partitioned into blocks with each set Q_i in the partition of Q inducing a block that contains the equations of X_q for $q \in Q_i$. Accepting sets induce greatest fixed-point blocks and rejecting blocks induce least fixed-point blocks. Since $\mathcal{A}$ is weak, the structure of the blocks in $\psi_{\mathcal{A}}$ satisfies the syntactic restrictions of the AFMC. $\square$

Remark 3.2 It is proved in [KV97] that alternating Büchi word automata can be translated to weak alternating word automata with a quadratic blow up. By replacing the branching modal operator $\forall\bigcirc$ with the linear model operator $\bigcirc$, the translation described in the proof of Theorem 3.1 can therefore be used in order to translate alternating Büchi word automata to the linear AFMC with a quadratic blow up. This is a doubly exponential improvement of the translations that follow from [AN92] and [VW94]. $\square$

We now characterize ω-regular languages $\mathcal{L}$ for which $der(\mathcal{L})$ can be characterized by an AFMC formula. Since trees in $der(\mathcal{L})$ are defined by means of a universal requirement on their paths, we do not need directed AFMC.

Theorem 3.3 *Given an ω-regular language $\mathcal{L}$, the following are equivalent.*

1. *$der(\mathcal{L})$ can be characterized by a AFMC formula.*

2. *$\mathcal{L}$ can be characterized by a DBW.*

[1] Alternatively, we could give up the directions and assume that the language of the AWT is symmetric [ES84]. Then, however, the translation described in Theorem 3.1 requires the formulas in the transition function of the AWT to be in DNF, and may therefore involve an exponential blow up.

Proof: Assume first that $der(\mathcal{L})$ has an equivalent AFMC formula. Then, $der_2(\mathcal{L})$ has an equivalent AFMC(2) formula and therefore, by Theorem 3.1, $der_2(\mathcal{L})$ can be recognized by an AWT(2). It is proved in [MSS86] that if a language of trees can be recognized by an AWT(2), then it can also be recognized by an NBT(2). It follows that $der_2(\mathcal{L})$ can be recognized by an NBT(2). It is proved in [KSV96] that for every ω-regular language $\mathcal{R}$, if $der_2(\mathcal{R})$ can be recognized by an NBT(2), then $\mathcal{R}$ can be recognized by a DBW. It follows that $\mathcal{L}$ can be recognized by a DBW.

Assume now that $\mathcal{L}$ can be recognized by a DBW. Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ be a DBW that recognizes $\mathcal{L}$. We define an AFMC formula $\psi_{\mathcal{L}}$ such that for every system S, we have $tree(S) \in der(\mathcal{L})$ iff S satisfies $\psi_{\mathcal{L}}$. The formula $\psi_{\mathcal{L}}$ has Σ as its set of atomic propositions. Each state $q \in Q$ induces two atomic variables X_q and X'_q with the following two equations:

$$X_q = \bigvee_{\sigma \in \Sigma} \sigma \wedge \forall \bigcirc X_{\delta(q,\sigma)} \wedge \forall \bigcirc X'_{\delta(q,\sigma)}.$$

$$X'_q = \left[\begin{array}{ll} \mathbf{true} & \text{if } q \in \alpha, \\ \bigvee_{\sigma \in \Sigma} \sigma \wedge \forall \bigcirc X'_{\delta(q,\sigma)} & \text{if } q \notin \alpha. \end{array} \right.$$

The set of equations is partitioned into two blocks. Equations with a left had side variable X_q constitute a greatest fixed-point block, and equations with a left had side variable X'_q constitute a least fixed-point block. Intuitively, once a fixed point is reached, each variable X'_q contains states w of S such that the run of $\mathcal{A}$ with initial state q on each of the computations starting at w eventually visits a state in α. Each variable X_q has one disjunct for every $\sigma \in \Sigma$. Its conjunct of the form $\forall \bigcirc X_t$ follows the transition function, and its conjunct of the form $\forall \bigcirc X'_t$ guarantees that a state from α is eventually visited. Since all variables X_q have a conjunct of the form $\forall \bigcirc X'_t$ in all the disjuncts in their equations, it is guaranteed that α is visited infinitely often.

$\square$

4 From ω-regular automata to AFMC

In this section we consider the case where specifications are given by ω-regular automata. We first study the problem of deciding whether a given automaton $\mathcal{A}$ can be translated to a DBW. By Theorem 3.3, the latter holds iff the specification given by $\mathcal{A}$ can be translated to the AFMC. We start, in Theorem 4.1, with the case where $\mathcal{A}$ is a deterministic automaton, and continue, in Theorem 4.2, with the case where $\mathcal{A}$ is a nondeterministic automaton.

In our proofs, we consider languages over an alphabet $\Sigma \times \{0,1\}$. For a word $w \in (\Sigma \times \{0,1\})^\omega$, let $w_1 \in \Sigma^\omega$ is the word obtained from w by projecting its letters on Σ, and similarly for w_2 and $\{0,1\}$. For words $x_1 \in \Sigma^\omega$ and $x_2 \in \{0,1\}^\omega$, let $x_1.x_2$ denote the word $w \in (\Sigma \times \{0,1\})^\omega$ with $w_1 = x_1$ and $w_2 = x_2$. For a word w and an index n, let $w[1 \ldots n]$ be the prefix of length n of w. Finally, let $\mathcal{L}_{fm}$ be the language over $\{0,1\}$ consisting of all words with only finitely many 0's.

Theorem 4.1

(1) *Deciding DPW $\mapsto$ DBW is NLOGSPACE-complete.*

(2) *Deciding $\{$DRW,DSW$\}$ $\mapsto$ DBW is PTIME-complete.*

Proof: We start with the upper bounds. The proof for Rabin and Streett automata is given in [KPB94]. Consider a DPW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$. A state $q \in Q$ is called *final* iff all the cycles that visit q are accepting. According to Landweber [Lan69], a deterministic automaton $\mathcal{A}$ recognizes a language that is in DBW iff for every accepting strongly connected component C of $\mathcal{A}$, all the strongly connected component C' with $C' \supset C$ are also accepting. This condition is used in [KPB94] in order to prove that $\mathcal{A}$ is in DBW iff the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by changing α to be the set of final states is equivalent to $\mathcal{A}$. Since $\mathcal{L}(\mathcal{A}')$ is always contained in $\mathcal{L}(\mathcal{A})$, checking whether $\mathcal{A}$ is in DBW can be reduced to checking whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(A')$. Since the emptiness problem for DPW is in NLOGSPACE, and since deciding whether a state q is final can be reduced to the emptiness problem, we are done.

The lower bound for parity automata follows by an easy reduction from the graph reachability problem. We prove here the lower bound for Rabin and Streett automata. For Streett automata, we do a reduction from DSW emptiness, proved to be PTIME-complete in [EL85]. Given a DSW $\mathcal{S}$, we define another DSW $\mathcal{A}$ such that $\mathcal{S}$ is empty iff $\mathcal{A}$ is in DBW. Let $\mathcal{S} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, and let $\mathcal{S}' = \langle \{0,1\}, Q', q_0', \delta', \alpha' \rangle$ be a DSW for $\mathcal{L}_{fm}$. We define $\mathcal{A} = \langle \Sigma \times \{0,1\}, Q \times Q', \langle q_0, q_0' \rangle, \delta'', \alpha'' \rangle$, where

- $\delta''(\langle q, q' \rangle, \langle \sigma, \sigma' \rangle) = \langle \delta(q, \sigma), \delta'(q', \sigma') \rangle$, and
- $\alpha'' = \{\langle L \times Q', R \times Q' \rangle : \langle L, R \rangle \in \alpha\} \cup \{\langle Q \times L', Q \times R' \rangle : \langle L', R' \rangle \in \alpha'\}$.

It is easy to see that

$$\mathcal{L}(\mathcal{A}) = \{w \ : w_1 \in \mathcal{L}(\mathcal{S}) \text{ and } w_2 \in \mathcal{L}_{fm}\}.$$

We prove that $\mathcal{S}$ is empty iff $\mathcal{A}$ is DBW. First, if $\mathcal{S}$ is empty, so is $\mathcal{A}$, and hence it is clearly in DBW. Assume now that $\mathcal{S}$ is not empty, we show that $\mathcal{L}(\mathcal{A})$ is not in DBW. The proof is very similar to the one showing that $\mathcal{L}_{fm}$ is not in DBW (c.f., [Tho90]), only that we have to accompany the words in $\{0,1\}^\omega$ with some word accepted by $\mathcal{S}$. Assume, by way of contradiction, that $\mathcal{L}(\mathcal{A})$ is in DBW. Then, by [Lan69], there exists a regular language $\mathcal{F}$ such that $\mathcal{L}(\mathcal{A}) = lim(\mathcal{F})$; that is $\mathcal{L}(\mathcal{A}) = \{w : w \text{ has infinitely many prefixes in } \mathcal{F}\}$. Let w be a word accepted by $\mathcal{S}$. Since $w.1^\omega$ is in $\mathcal{L}(\mathcal{A})$, there exists some p_1 such that $w.1^\omega[1 \ldots p_1]$ is in $\mathcal{F}$. Since $w.1^{p_1} \cdot 0 \cdot 1^\omega$ is in $\mathcal{L}(\mathcal{A})$, there exists some p_2 such that $w.1^{p_1} \cdot 0 \cdot 1^\omega[1 \ldots p_1 + 1 + p_2]$ is in $\mathcal{F}$. We can continue and obtain an infinite sequence of finite words $w.1^{p_1} \cdot 0 \cdot 1^{p_2} \cdot 0 \cdots 0 \cdot 1^{p_{k-1}} \cdot 0 \cdot 1^\omega[1 \ldots p_1 + 1 + p_2 + 1 + \ldots + 1 + p_k]$ $(k = 1, 2, 3, \ldots)$, all in $\mathcal{F}$. Hence, the infinite word $w.1^{p_1} \cdot 0 \cdot 1^{p_2} \cdot 0 \cdot 1^{p_3} \cdot 0 \cdot 1^{p_4} \cdots$ is in $lim(\mathcal{F})$ and thus in $\mathcal{L}(\mathcal{A})$, and we reach a contradiction.

For Rabin automata, we do a reduction from DRW universality, which is dual to DSW emptiness, and is therefore PTIME-complete. Given a DRW $\mathcal{R}$, we define a DRW $\mathcal{A}$ such that $\mathcal{R}$ is universal iff $\mathcal{A}$ is in DBW. Let $\mathcal{R} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, and let $\mathcal{R}' = \langle \{0,1\}, Q', q_0', \delta', \alpha' \rangle$ be a DRW for the language $\mathcal{L}_{fm}$. We define $\mathcal{A}$ as above. Here, however, $\mathcal{L}(\mathcal{A}) = \{w \ : w_1 \in \mathcal{L}(\mathcal{R}) \text{ or } w_2 \in \mathcal{L}_{fm}\}$, and $\mathcal{R}$ is universal iff $\mathcal{A}$ is DBW.

□

Theorem 4.2
Deciding {NBW,NPW,NRW,NSW} $\mapsto$ DBW is PSPACE-complete.

Proof: We start with the upper bound. We show that all the four types of automata can be translated to DPW with an exponential blow up. Then, by Theorem 4.1, checking whether an automaton of these types is in DBW can be done in polynomial space. Since Büchi and parity automata are special cases of Rabin and Streett automata, we describe the translation for NRW and NSW. Let $\mathcal{N}$ be either a NRW or a NSW with n states and h pairs. By [Saf88, Saf89], in both cases we can translate $\mathcal{N}$ to a DRW with $2^{O(nh \log nh)}$ states and nh pairs. It is shown in [KPBV95] that a DRW with m states and k pairs can be translated to a DPW with at most $m \cdot 2^{k \log k}$ states and k sets. Accordingly, we can translate $\mathcal{N}$ also to a DPW with $2^{O(nh \log nh)}$ states and nh sets.

We now prove the lower bound. Since Büchi automata are a special case of parity, Rabin, and Streett automata, we describe the proof for NBW. We do a reduction from NBW universality, proved to be PSPACE-hard in [MS72, Wol82]. Given an NBW $\mathcal{B}$, we define another NBW $\mathcal{A}$ such that $\mathcal{B}$ is universal iff $\mathcal{A}$ is in DBW. Let $\mathcal{B} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, and let $\mathcal{B}' = \langle \{0,1\}, Q', q_0', \delta', \alpha' \rangle$ be an NBW for the language $\mathcal{L}_{fm}$. We define $\mathcal{A} = \langle \Sigma \times \{0,1\}, Q \times Q', \langle q_0, q_0' \rangle, \delta'', \alpha'' \rangle$, where

- $\delta''(\langle q, q' \rangle, \langle \sigma, \sigma' \rangle) =$
  $\{\langle s, s' \rangle : s \in \delta(q, \sigma) \text{ and } s' \in \delta'(q', \sigma')\}$, and
- $\alpha'' = (\alpha \times Q') \cup (Q \times \alpha')$.

It is easy to see that $\mathcal{L}(\mathcal{A}) = \{w \ : w_1 \in \mathcal{L}(\mathcal{B}) \text{ or } w_2 \in \mathcal{L}_{fm}\}$. Proving that $\mathcal{B}$ is universal iff $\mathcal{A}$ is DBW follows exactly the same arguments as in the proof of Theorem 4.1. □

Theorems 4.1 and 4.2 consider the problem of checking whether a specification given by an automaton can be translated to a DBW, and hence also to an AFMC formula. We now consider the blow-up in the translation.

Theorem 4.3 *When exists, the translation*

(1) *{DBW,DPW,DRW} $\mapsto$ AFMC is linear.*

(2) *DSW $\mapsto$ AFMC is polynomial.*

(3) *{NBW,NPW,NRW,NSW} $\mapsto$ AFMC is exponential.*

Proof: A linear translation of DBW to AFMC is described in the proof of Theorem 3.3. It is proved in [KPB94], that if a DRW $\mathcal{A}$ is in DBW, then it has a DBW of the same size. When $\mathcal{A}$ is in DSW, its translation to DBW, when exists, involves a polynomial blow up [KPB95]. Hence the bounds for DPW, DRW, and DSW. All the types of nondeterministic automata $\mathcal{A}$ in **(3)** have an exponential translation to DRW [Saf88, Saf92]. Therefore, by **(1)**, if $\mathcal{A}$ is in DBW, it has an exponential equivalent AFMC formula. □

It follows from [Mic88] that the exponential translation from nondeterministic automata to DBW cannot be improved. This, however, does not imply an exponential lower bound for the translation to the AFMC.

5 From LTL to AFMC

In this section we consider the case where specifications are given by LTL formulas. As in Section 4, we first consider the problem of checking whether a given LTL formula can be translated to a DBW and hence, also to an AFMC formula.

Theorem 5.1 *Deciding LTL $\mapsto$ DBW is in EXPSPACE and is PSPACE-hard.*

Proof: Given an LTL formula ψ of length n, let $\mathcal{B}_\psi$ be an NBW that recognizes ψ. By [VW94], $\mathcal{B}_\psi$ has $2^{O(n)}$ states. Membership in EXPSPACE then follows from Theorem 4.2. For the lower bound, we do a reduction from LTL satisfiability. Given an LTL formula ψ over some set P of propositions, let p be a proposition not in P. Using the same arguments as in the proof of Theorem 4.1, one can prove that ψ is not satisfiable iff $\psi \wedge \Diamond\Box p$ is in DBW. □

We now discuss the blow-up involved in translating a given LTL formula ψ to an equivalent AFMC formula (when exists). One possibility is to first translate ψ to a DBW. Below we show that such an approach is inherently doubly exponential.

Theorem 5.2
When exists, the translation LTL $\mapsto$ DBW is doubly exponential.

Proof: The upper bound follows from the exponential translation of LTL formulas to NBW [VW94], and Theorem 4.3. For the lower bound, consider the regular language

$$\mathcal{L}_n = \{\{0,1,\#\}^* \cdot \# \cdot w \cdot \# \cdot \{0,1,\#\}^* \cdot \$ \cdot w \ : \ w \in \{0,1\}^n\}.$$

A word τ is in $\mathcal{L}_n$ iff the suffix of length n that comes after the single \$ in τ appears somewhere before the \$. By [CKS81], the smallest deterministic automaton on finite words that accepts $\mathcal{L}_n$ has at least 2^{2^n} states (reaching the \$, the automaton should remember the possible set of words in $\{0,1\}^n$ that have appeared before). We can specify $\mathcal{L}_n$ with an LTL formula of length quadratic in n (we ignore here the technical fact that DBW and LTL formulas describe infinite words). The formula makes sure that there is only one \$ in the word and that eventually there exists a position in which # is true and the ith letter from this position, for $1 \leq i \leq n$, agrees with the i'th letter after the \$. Formally, the formula is

$$[(\neg\$)U(\$ \wedge \bigcirc\Box\neg\$)] \ \wedge$$

$$\Diamond[\# \wedge \bigwedge_{1\leq i\leq n}((\bigcirc^i 0 \wedge \Box(\$ \rightarrow \bigcirc^i 0)) \vee (\bigcirc^i 1 \wedge \Box(\$ \rightarrow \bigcirc^i 1)))].$$

Note that the argument about the size of the smallest deterministic automaton that recognizes

$\mathcal{L}_n$ is independent of the automaton's acceptance condition. Thus, the theorem holds for DPW, DRW, and DSW as well. □

Translating LTL to AFMC by going through DBW involves a doubly exponential blow up. Hopefully, this blow-up can be improved to an exponential one, matching the lower bound we describe below.

Theorem 5.3
When exists, the translation LTL $\mapsto$ AFMC is doubly exponential and is at least exponential.

Proof: Since, by Theorem 4.3, DBW can be linearly translated to AFMC, the upper bound follows from Theorem 5.2.

For the lower bound, consider again the language $\mathcal{L}_n$ from Theorem 5.2. We prove that an NBT for the language $der_2(\mathcal{L}_n)$ needs to have at least 2^{2^n} states. Then, since AFMC formulas have a linear translation to AWT [BVW94], and therefore also an exponential translation to NBT [MSS86], we are done. Let $S = \{0,1\}^n$ be the set of all words over $\{0,1\}$ of length n. Consider a full finite $\{0,1,\#\}$-labeled tree $\langle T, V\rangle$ such that for every nonempty subset S' of S, there exists a path ρ in T such that the set of all the words w for which $\# \cdot w \cdot \#$ appears in $V(\rho)$ is exactly S'. It is not hard to see that such a tree $\langle T, V\rangle$ of branching degree 2 exists. For each leaf x in $\langle T, V\rangle$, let $S^x \subseteq S$ be the set of all the words in S that appear between #'s in the path from the root to x, and let $\langle T_x, V_x\rangle$ be a finite $\{0,1,\$\}$-labeled tree such that the root of $\langle T_x, V_x\rangle$ is labeled \$ and the paths of $\langle T_x, V_x\rangle$ are labeled by exactly all the words in S^x prefixed by \$. The tree $\langle T_x, V_x\rangle$ can be defined as a full tree of branching degree 2, i.e., $T_x = \bigcup_{0 \le i \le n} \{0,1\}^i$. Now, note that S^x is a subset of the leaves of T_x. We can therefore define V_x as follows. First, we consider nodes $y \cdot c \in T_x$, for $y \in \{0,1\}^*$ and $c \in \{0,1\}$, for which $y \cdot c \in \rho$ for some path ρ whose leaf is in S^x. We label such nodes $y \cdot c$ by c. Then, it is easy to fill the labels of the other nodes so that no word not in S^x is generated. Consider now the finite tree $\langle T', V'\rangle$ obtained from $\langle T, V\rangle$ by concatenating to each leaf x in T the tree $\langle T_x, V_x\rangle$. The tree $\langle T', V'\rangle$ is in $der_2(\mathcal{L}_n)$.

Assume now, by way of contradiction, that $der_2(\mathcal{L}_n)$ can be recognized by an NBT $\mathcal{A}$ with less than 2^{2^n} states. Then, in the accepting run r of $\mathcal{A}$ on $\langle T', V'\rangle$, there are at least two nodes x_1 and x_2, such that both nodes are leaves in $\langle T, V\rangle$, $S^{x_1} \neq S^{x_1}$, and $r(x_1) = r(x_2)$. Consider now the tree obtained from $\langle T', V'\rangle$ by replacing the subtree $\langle T_{x_1}, V_{x_1}\rangle$ by the subtree $\langle T_{x_2}, V_{x_2}\rangle$. Though this tree is not in $der_2(\mathcal{L}_n)$, the automaton $\mathcal{A}$ accepts it, and we reach a contradiction. □

References

[AN92] A. Arnold and D. Niwiński. Fixed point characterization of weak monadic logic definable sets of trees. In *Tree Automata and Languages*, pp. 159–188, Amsterdam, 1992. Elsevier.

[BC96] G. Bhat and R. Cleavland. Efficient model checking via the equational μ-calculus. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pp. 304–312, June 1996.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BHSS+96] R.K. Brayton, G.D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. Edwards, S. Khatri, T. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In *Computer Aided Verification, Proc. 8th Int. Conference*, LNCS 1102, pp. 428–432, 1996.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pp. 1–12, Stanford, 1962. Stanford University Press.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Computer Aided Verification, Proc. 6th Int. Conference*, LNCS 818, pp. 142–155, 1994.

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, LNCS 354, pp. 428–437, 1988.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pp. 52–71, 1981.

[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.

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Decade of Concurrency – Reflections and Perspectives (Proceedings of REX School)*, LNCS 803, pp. 124–175, 1993.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[CS91] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. In *Proc. 3rd Conference on Computer Aided Verification*, volume LNCS 575, pp. 48–58, 1991.

[EL85] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, North Holywood, 1985. Western Periodicals Company.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the proposoitional Mu-calculus. In *Proc. 1st Symposium on Logic in Computer Science*, pp. 267–278, 1986.

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

[HHK96] R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *Computer Aided Verification, Proc. 8th Int. Conference*, LNCS 1102, pp. 423–427, 1996.

[HKSV97] R.H. Hardin, R.P. Kurshan, S.K. Shukla, and M.Y. Vardi. A new heuristic for bad cycle detection using BDDs. In *Computer Aided Verification, Proc. 9th Int. Conference*, LNCS 1254, pp. 268–278, 1997.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.

[HT87] Th. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th International Coll. on Automata, Languages, and Programming*, LNCS 267, pp. 269–279, 1987.

[KG96] O. Kupferman and O. Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, 1996.

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

[KPB94] S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Buchi automata. In *Algorithms and Computations*, LNCS 834, pp. 378–386, 1994.

[KPB95] S.C. Krishnan, A. Puri, and R.K. Brayton. Structural complexity of ω-automata. In *Symposium on Theoretical Aspects of Computer Science*, LNCS 900, 1995.

[KPBV95] S.C. Krishnan, A. Puri, R.K. Brayton, and P.P. Varaiya. The Rabin index and chain automata, with applications to automata and games. In *Computer Aided*

Verification, Proc. 7th Int. Conference, LNCS 939, pp. 253–266, 1995.

[KSV96] O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pp. 322–333, June 1996.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes.* Princeton Univ. Press, 1994.

[KV97] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, pp. 147–158. IEEE Computer Society Press, 1997.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symposium on Principles of Programming Languages*, pp. 97–107, New Orleans, January 1985.

[McM93] K.L. McMillan. *Symbolic model checking.* Kluwer Academic Publishers, 1993.

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[MS72] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th IEEE Symp. on Switching and Automata Theory*, pp. 125–129, 1972.

[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*, 1986.

[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 International Symp. on Programming*, LNCS 137, pp. 337–351, 1981.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pp. 1–23. North Holland, 1970.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pp. 319–327, White Plains, October 1988.

[Saf89] S. Safra. *Complexity of automata on infinite objects.* PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.

[Saf92] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symposium on Theory of Computing*, 1992.

[Sch97] K. Schneider. CTL and equivalent sublanguages of CTL*. In *Proceedings of IFIP Conference on Computer Hardware Description Languages and Applications*, pp. 40–59, 1997. Chapman and Hall.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pp. 165–191, 1990.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First Symposium on Logic in Computer Science*, pp. 322–331, 1986.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[Wol82] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications.* PhD thesis, Stanford University, 1982.