From Linear Time to Branching Time

ORNA KUPFERMAN
Hebrew University
and
MOSHE Y. VARDI
Rice University

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 article, 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 possible.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Theory, Verification

Additional Key Words and Phrases: Linear temporal logic, alternation-free μ-calculus

A preliminary version, titled "Freedom, weakness, and determinism: From linear-time to branching—time" appeared in *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, Los Alamitos, Calif., 1998.
O. Kupferman was partly supported by BSF grant 9800096, and by a grant from Minerva.
M. Y. Vardi was partly supported by NSF grant CCR-9700061, and CCR-9988322, BSF grant 9800096, and by a grant from the Intel Corporation.
Authors' addresses: O. Kupferman, School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel; email: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/∼orna; M. Y. Vardi, Department of Computer Science, Rice University, Houston, TX 77005-1892; email: vardi@cs.rice.edu, URL: http://www.rice.edu/∼vardi.

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, that is, "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 [Clarke et al. 1986; Lichtenstein and Pnueli 1985; Vardi and Wolper 1986], 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 [Burch et al. 1992; McMillan 1993]. In particular, the use of BDDs [Bryant 1986] for model representation has yielded model-checking tools that can handle systems with 10^{120} states and beyond. For a survey, see Clarke et al. [1999].

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 [Kozen 1983]. As such, it describes fixed-point computations in a natural form. In particular, the *alternation-free* fragment of μ-calculus (AFMC, for short) [Emerson and Lei 1986] 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 [Cleaveland and Steffen 1991]. 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 [Cadence].

When, as in the model-checking tools SMV and VIS [McMillan 1993; Brayton et al. 1996], 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 and SPIN [Kurshan 1994; Hardin et al. 1996; Holzmann 1997] 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 Rabin [1970], Muller et al. [1986], and Kupferman et al. [2000], 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 [Emerson and Lei 1986]. 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 [Hardin et al. 2001].

In this article, 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. Clarke and Draghicescu [1988] 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 Kupferman and Grumberg [1996], and Schneider [1997], which identify certain fragments of LTL that can be easily translated to CTL. Recently, Maidl [2000] gave a characterization of LTL formulas that can be translated into the universal fragment of CTL. A study of the possible use of CTL model-checking tools for the verification of such properties without first translating them to CTL is described in Kupferman and Vardi [1998], and Bloem et al. [1999]. 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 Muller et al. [1986], where they were related to weakly definable sets [Rabin 1970]. The relevance of weak alternating automata to model checking was demonstrated in Kupferman et al. [2000]. The equivalence of AFMC and weak alternating automata is proved also in Arnold and Niwiński [1992], where both formalisms are shown to be equivalent to the weak monadic second-order theory of trees [Rabin 1970; Muller et al. 1986]. Our proof is simpler and direct, and we describe linear translations between the two formalisms. Also, while the result in Arnold and Niwiński [1992] refers to directed trees (i.e., the next-time operator of AFMC is parameterized with a direction), our result here refers to general trees and symmetric tree automata [Janin and Walukiewicz 1995; Wilke 1999]. We then use known relations between automata on infinite words and trees [Kupferman et al. 1996] 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 [Büchi 1962; Landweber 1969].

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 [Emerson and Lei 1986]. 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. As we discuss in Section 6, the question whether our doubly exponential translation can be improved to an exponential one is still open.

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 $w_0 = w_{in}$. We denote the ith state in π by $\pi[i]$.

Formulas of the linear-time temporal logic *LTL* describe computations of systems [Pnueli 1981]. Given a set P of atomic propositions, an LTL formula is **true**, **false**, 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 $\Box(\mathit{request} \rightarrow \Diamond \mathit{grant})$ states that every request is followed by a grant. In order to define formally the semantics of LTL, consider a computation π of a system. We use π^i to denote the suffix $\pi[i], \pi[i+1], \ldots$ of π, and we use $\pi \models \psi$ to indicate that the computation π satisfies the LTL formula ψ. The relation $\models$ is inductively defined as follows[1]:

—For all π, we have that $\pi \models$ **true** and $\pi \not\models$ **false**.

—For an atomic proposition $p \in AP$, $\pi \models p$ iff $p \in L(\pi[0])$.

—$\pi \models \neg\psi_1$ iff $\pi \not\models \psi_1$.

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

—$\pi \models \bigcirc\psi_1$ iff $\pi^1 \models \psi_1$.

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

The *alternation-free μ-calculus* (*AFMC*, for short) is a fragment of the modal logics μ-calculus [Kozen 1983]. We define the AFMC by means of equational blocks, as in Cleaveland and Steffen [1991]. Formulas of AFMC are defined with respect to a set P of atomic propositions and a set Var of atomic variables. We consider here formulas in a positive normal form, where negation can be applied to atomic propositions only. A *basic* AFMC formula is either p, $\neg p$, X, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\exists\bigcirc\varphi$, or $\forall\bigcirc\varphi$, for $p \in P$, $X \in \mathit{Var}$, and basic AFMC formulas φ and ψ. Basic AFMC formulas appear in *equational blocks*. 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 distinct atomic variables. Each variable X_i can be in the left-hand side of at most one equational block. 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 B *depends* on B'. Such dependencies cannot be circular (note that B and B' are distinct). This ensures that the formula is free of alternations. A variable X is *free* in B if it appears only in the right-hand side of the equations in B. An AFMC formula is of the form $X(S)$, for a variable X and a system S. If ψ_1 and ψ_2 are AFMC formulas, then $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, $\exists\bigcirc\psi_1$, and $\forall\bigcirc\psi_1$ are also AFMC formulas.

The semantics of AFMC 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; that is, for all $1 \leq i \leq n$, we have $W_i \subseteq W$. Each basic AFMC formula defines a subset of the states of S in the standard way. We denote by $\varphi_{\mathcal{V}}^S$ the set of states define by φ under the evaluation $\mathcal{V}$. Formally,

[1]The temporal operators $\Box$ and $\Diamond$ are defined in terms of the operator U. Specifically, $\Diamond\varphi = \mathbf{true} U \varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$.

—$p^S_{\mathcal{V}} = \{w \in W : p \in L(w)\}$,
—$\neg p^S_{\mathcal{V}} = \{w \in W : p \notin L(w)\}$,
—${X_i}^S_{\mathcal{V}} = W_i$,
—$(\varphi \vee \psi)^S_{\mathcal{V}} = \varphi^S_{\mathcal{V}} \cup \psi^S_{\mathcal{V}}$,
—$(\varphi \wedge \psi)^S_{\mathcal{V}} = \varphi^S_{\mathcal{V}} \cap \psi^S_{\mathcal{V}}$,
—$(\exists\bigcirc\varphi)^S_{\mathcal{V}} = \{w \in W :$ there exists $w' \in \varphi^S_{\mathcal{V}}$ such that $R(w, w')\}$.
—$(\forall\bigcirc\varphi)^S_{\mathcal{V}} = \{w \in W :$ for all w' with $R(w, w')$, we have $w' \in \varphi^S_{\mathcal{V}}\}$.

For the nonbasic formulas, the semantics is as above for formulas of the form $\psi_1 \vee \psi_2$, $\psi_1 \wedge \psi_2$, $\exists\bigcirc\psi_1$, and $\forall\bigcirc\psi_1$. For a formula of the form $X(S)$, the semantics is defined as follows. The variable X is in the left-hand side of some block B. 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. Formally, let B be a block containing the equations $X_1 = \varphi_1, \ldots, X_n = \varphi_n$. Given a valuation $\mathcal{V}$ to the free variables in B, we define a function $f^S_{B,\mathcal{V}} : (Var \times 2^W)^n \to (Var \times 2^W)^n$ that transfers a valuation $\mathcal{V}'$ of $X_1, \ldots, X_n$ to the valuation obtained by solving the equations in B with respect to the valuation $\mathcal{V} \cup \mathcal{V}'$ (note that the valuation $\mathcal{V} \cup \mathcal{V}'$ assigns values to all the variables in the right-hand side of equations in B; indeed, each variable in the right-hand side is either free, in which case it is assigned to a value by $\mathcal{V}$, or is not free, in which case it is some X_i and is assigned to a value by $\mathcal{V}'$). Formally, $f^S_{B,\mathcal{V}}(\{\langle X_1, W_1\rangle, \ldots, \langle X_n, W_n\rangle\}) = \{\langle X_1, W'_1\rangle, \ldots, \langle X_n, W'_n\rangle\}$, where for all $1 \leq i \leq n$, we have that $W'_i = {\varphi_i}^S_{\mathcal{V}\cup\mathcal{V}'}$. Then, the assignment to the variables in the left hand side of the equations in $\nu\{E\}$ is the greatest fixed point of $f^S_{B,\mathcal{V}}$, and the assignment to the variables in the left hand side of the equations in $\mu\{E\}$ is the least fixed point of $f^S_{B,\mathcal{V}}$. Note that the operators of basic formulas are monotonic. Hence, by Tarski–Knaster Theorem, both fixed points exist, and, for a finite S, can be computed iteratively. 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 $\mathcal{V}$. Since there is no circular dependency among the blocks, such an order exists. As detailed in Cleaveland and Steffen [1991], the evaluation can be completed in linear time (see also Kupferman et al. [2000]).

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 the initial state of S satisfies φ, that is, whether $w_{in} \in \varphi^S_{\emptyset}$. The semantics of AFMC, as well as our results here can be easily extended to systems with multiple initial states.

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 $V : T \to \Sigma$ 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 a fixed branching degree d, we denote by $tree(S)$ the 2^P-labeled d-tree obtained by unwinding S from its initial state. Formally, given $S = \langle P, W, w_{in}, R, L\rangle$ and a state $w \in S$, let $succ_R(w) = \langle w_1, \ldots, w_k\rangle$ be an ordered tuple of the successors of w in S (as we elaborate below, the assumptions that the branching degree is fixed to d and that the successors are ordered is of technical convenience. Our results hold also without these assumptions). Unwinding of S results in the W-labeled d-tree $\langle [d]^*, V\rangle$ in which $V(\epsilon) = w_{in}$, and for every $x \in [d]^*$ with $V(x) = w$, we have $\langle V(x \cdot 1), \ldots, V(x \cdot d)\rangle = succ_R(w)$. Then, $tree(S)$ is the 2^P-labeled d-tree $\langle [d]^*, V'\rangle$ with $V'(x) = L(V(x))$, for every $x \in [d]^*$.

An *alternating tree automaton* [Muller and Schupp 1987] $\mathcal{A} = \langle \Sigma, d, Q, q_0, \delta, \alpha\rangle$ 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 a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. The transition function $\delta : Q \times \Sigma \to \mathcal{B}^+([d] \times Q)$ 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. Formally, $\langle T_r, r\rangle$ is a Σ_r-labeled tree where $\Sigma_r = [d]^* \times Q$ and $\langle T_r, r\rangle$ satisfies the following:

(1) $\epsilon \in T_r$ and $r(\epsilon) = (\epsilon, q_0)$, for some $q_0 \in Q_0$.
(2) Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_1, q_1), (c_2, q_2), \ldots, (c_n, q_n)\} \subseteq [d] \times Q$, such that the

following hold:
—S satisfies θ, and
—for all $1 \leq i \leq n$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, q_i)$.

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 $(2, q_2)$. Note that if $\theta = \textbf{true}$, then y need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta = \textbf{false}$, we cannot have a run that takes a transition with $\theta = \textbf{false}$. Each infinite path ρ in $\langle T_r, r \rangle$ is labeled by a word $r(\rho)$ in $([d]^* \times Q)^\omega$. Let $\mathit{inf}(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. Thus,

$$\mathit{inf}(\rho) = \{q : r(y) \in [d]^* \times \{q\} \text{ for infinitely many } y \in \rho\}.$$

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 partition of Q into disjoint sets; that is $Q = \bigcup_{1 \leq i \leq k} G_i$, and $G_i \cap G_j = \emptyset$ for all $1 \leq i \neq j \leq k$. An infinite path ρ satisfies α iff the minimal index i for which $\mathit{inf}(\rho) \cap (\bigcup_{1 \leq j \leq i} G_j) \neq \emptyset$ 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 $\mathcal{A}$; that is, 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; that is, 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.

While tree automata run on trees with a fixed branching degree, branching-time temporal logic formulas define trees of variable branching degrees. Indeed, when a tree automaton reads a node x in the input d-tree, it knows that x has exactly d successors, and its transition function depends on d. On the other hand the formula $\forall \bigcirc \varphi$ can be evaluated on states with an arbitrary number of successors. In addition, while automata can distinguish between the different successors of a node (e.g., refer to the leftmost successor), formulas cannot. *Symmetric automata* [Janin and Walukiewicz 1995; Wilke 1999] are tree automata in which successors are sent in either an existential or a universal manner, and they are suitable for reasoning about temporal logic.

Let $\Omega = \{\Box, \diamond\}$. A symmetric alternating automaton is an alternating tree automaton in which the transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(\Omega \times Q)$ maps a state and a letter to a formula in $\mathcal{B}^+(\Omega \times Q)$. Intuitively, an atom $\langle \Box, q \rangle$ corresponds to d copies of the automaton in state q, sent to all the successors of the current node (d is the number of successors of the current node, which may not be known in advance). An atom $\langle \diamond, q \rangle$ corresponds to a copy of the automaton in state q, sent to some successor of the current node. When, for instance, the automaton is in state q, reads a node x with successors $x \cdot c_1, \ldots, x \cdot c_d$ and

$$\delta(q, V(x)) = (\Box, q_1) \wedge (\diamond, q_2) \vee (\diamond, q_2) \wedge (\diamond, q_3),$$

it can either send d copies in state q_1 to the nodes $x \cdot c_1, \ldots, x \cdot c_d$ and send a copy in state q_2 to some node in $x \cdot c_1, \ldots, x \cdot c_d$, or send one copy in state q_2 to some node in $x \cdot c_1, \ldots, x \cdot c_d$ and send one copy in state q_3 to some node in $x \cdot c_1, \ldots, x \cdot c_d$. Thus, symmetric automata cannot distinguish between left and right and can send copies to successor nodes only in either a universal or an existential manner.

A *run* of a symmetric automaton is defined in a way similar to the way a run of an alternating automaton is defined, only that here, for a node $y \in T_r$ with $r(y) = (x, q)$, $\delta(q, V(x)) = \theta$, and successors $x \cdot c_1, \ldots, x \cdot c_d$ of x in $\langle T, V \rangle$, for $d \geq 0$, there is a (possibly empty) set $S \subseteq \Omega \times Q$, such that S satisfies θ, and for all $(c, s) \in S$, the following hold:

—If $c = \Box$, then for each $1 \leq i \leq d$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, s)$.
—If $c = \diamond$, then for some $1 \leq i \leq d$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, s)$.

Muller et al. [1986] 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.

We denote each of the different types of automata by three letter acronyms in $\{D, N, A\} \times \{F, B, P, R, S, W\} \times \{W, T\}$, where the first letter describe the branching mode of the automaton (deterministic, nondeterministic, or alternating), the second letter describes the acceptance condition (finite, Büchi, parity, Rabin, Streett, or weak), 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.

We define the *size* $|\mathcal{A}|$ of an alternating automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ as $|Q| + |\alpha| + |\delta|$, where $|Q|$ and $|\alpha|$ are the respective cardinalities of the sets Q and α, and where $|\delta|$ is the sum of the lengths of the satisfiable (i.e., not **false**) formulas that appear as $\delta(q, \sigma)$ for some q and σ.

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 (e.g., refer to the leftmost successor), AFMC formulas define trees of variable branching degrees and cannot distinguish between different successors. Accordingly, discussion is restricted to symmetric automata. Alternatively, we could restrict attention to trees over some fixed branching degree and *directed AFMC*, where the next-time operator is annotated with an explicit direction [Hafer and Thomas 1987]. In addition, as AFMC formulas are interpreted with respect to systems in which each state is labeled by a set of atomic propositions, we consider automata with alphabet $\Sigma = 2^P$ for some set P of atomic propositions.

THEOREM 3.1. *Symmetric AWT = AFMC.*

PROOF. A linear translation of AFMC formulas to symmetric AWT is given in Kupferman et al. [2000]. For the other direction, we describe a linear translation of symmetric AWT to AFMC formulas. The translation is similar to the one described in Bhat and Cleavland [1996], where ABT are translated to μ-calculus formulas of alternation depth 2. Here, we consider AWT, and alternation between fixed points is not required. Consider a symmetric AWT $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$. Let $Q_0 \leq Q_1 \leq \cdots \leq Q_n$ be the partition of Q into sets. Recall that $\Sigma = 2^P$ for some set P of atomic propositions. For $\sigma \in \Sigma$, we use σ to abbreviate the propositional formula $(\bigwedge_{p\in\sigma} p) \wedge (\bigwedge_{p\notin\sigma} \neg p)$. We define an AFMC formula $\psi_{\mathcal{A}}$ such that for every system $S = \langle P, W, w_{in}, R, L \rangle$, we have $tree(S) \in \mathcal{L}(\mathcal{A})$ iff S satisfies $\psi_{\mathcal{A}}$. The formula $\psi_{\mathcal{A}}$ has P 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 $\theta \in \mathcal{B}^+(\{\Box, \Diamond\} \times Q)$, let $f(\theta)$ be the AFMC formula obtained from θ by replacing an atom $(\Diamond, q)$ by the formula $\exists\bigcirc X_q$, and replacing an atom $(\Box, q)$ by the formula $\forall\bigcirc X_q$. 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 Q_i induce greatest fixed-point blocks $\nu\{E_i\}$ and rejecting blocks Q_i

induce least fixed-point blocks $\mu\{E_i\}$. Since $\mathcal{A}$ is weak, the structure of the blocks in $\psi_{\mathcal{A}}$ satisfies the syntactic restrictions of the AFMC. Then, $\psi_{\mathcal{A}}$ is $X_{q_0}(B_0)$.

We prove that for every system S, we have $tree(S) \in \mathcal{L}(\mathcal{A})$ iff S satisfies $\psi_{\mathcal{A}}$. Recall the partition $Q_0 \leq \cdots \leq Q_n$ on the states of $\mathcal{A}$. For a state $q \in Q$, let $W_q \subseteq W$ be such that $w \in W_q$ iff the tree obtained by unwinding S from w is accepted by the automaton $\mathcal{A}$ with initial state q. Consider a block $\lambda_i\{E_i\}$ associated with Q_i ($\lambda_i \in \{\nu, \mu\}$). We prove by induction on i that the fixed point (greatest, in case $\lambda = \nu$, and least, in case $\lambda = \mu$) of the equations in E_i is obtained by assigning to a variable X_q with $q \in Q_i$ the set W_q (recall that the variables on the left-hand side of the equations in E_i correspond to states in Q_i).

We start with Q_0. Note that since the transitions from states in Q_0 involve only states in Q_0, there are no free variables in E_0, thus its evaluation is done with respect to no valuation. Assume first that $\lambda_0 = \mu$. The valuation of the least fixed point of E_0 proceeds in iterations as follows. For each variable X_q, for $q \in Q_0$ and $j \geq 0$, let $X_q[j]$ denote the value of X_q after the jth iteration of the valuation. Since $\lambda_0 = \mu$, all variables are empty at initialization. Thus, $X_q[0] = \emptyset$ for all $q \in Q_0$. At the end of the first iteration, each variable X_q, for $q \in Q_0$, contains states w for which $\delta(q, L(w)) = \mathbf{true}$. Thus, $X_q[1] = \{w : \delta(q, L(w)) = \mathbf{true}\}$. Then, in the $(j+1)$th iteration, the assignment to each variable X_q is extended to states w for which $\delta(q, L(w))$ is satisfied by some set $\{(c_1, X_{q_1}), \ldots, (c_l, X_{q_l})\} \subseteq \{\Box, \Diamond\} \times Q$ such that for every $1 \leq k \leq l$, if $c_k = \Box$, then $X_{q_k}[j]$ contains all the successors of w, and if $c_k = \Diamond$, then $X_{q_k}[j]$ already contains some successor of w. Thus, after the jth iteration, the set $X_q[j]$ contains all states w for which there is a run r of $\mathcal{A}$ with initial state q on the tree obtained by unwinding S from w such that all the paths of r gets to a state where the transition function imposes no obligations within at most j transitions. When we reach a fixed point and $X_q[j] = X_q[j+1]$, the set $X_q[j]$ is W_q as required. Indeed, as Q_0 is rejecting, W_q is the set of states w for which there is a run r of $\mathcal{A}$ with initial state q on the tree obtained by unwinding S from w such that all the path of r eventually gets to a state where the transition function imposes no obligations.

Assume now that $\lambda_0 = \nu$. The valuation of the greatest fixed point of E_0 proceeds as follows. Now, $X_q[0] = W$, and at the end of the first iteration, each variable X_q, for $q \in Q_0$, contains states w for which $\delta(q, L(w)) \neq \mathbf{false}$. Thus, $X_q[1] = \{w : \delta(q, L(w)) \neq \mathbf{false}\}$. Then, in the $(j+1)$th iteration, the set $X_q[j+1]$ is restricted to states w for which $\delta(q, L(w))$ is satisfied by some set $\{(c_1, X_{q_1}), \ldots, (c_l, X_{q_l})\} \subseteq \{\Box, \Diamond\} \times Q$ such that for every $1 \leq k \leq l$, if $c_k = \Box$, then $X_{q_k}[j]$ contains all the successors of w, and if $c_k = \Diamond$, then $X_{q_k}[j]$ contains some successor of w. When we reach a fixed point and $X_q[j] = X_q[j+1]$, the set $X_q[j]$ is W_q as required. Indeed, as Q_0 is accepting, W_q is the set of states w for which there is a run r of $\mathcal{A}$ with initial state q on the tree obtained by unwinding S from w such that r stays forever in states in Q_0.

For the induction step, consider the valuations $\mathcal{V}_i = \bigcup_{q \in Q_0 \cup \cdots \cup Q_{i-1}} \{\langle X_q, W_q \rangle\}$, for $1 \leq i \leq n$. By the induction hypothesis, $\mathcal{V}_i$ is the assignment in $\psi_{\mathcal{A}}$ to the free variables in E_i, which are associated with states in sets lower than Q_i.

Now, when $\lambda_i = \mu$, the valuation of the least fixed point of E_i is such that at the end of the first iteration, each variable X_q, for $q \in Q_i$, contains the set of states w for which $\delta(q, L(w))$ is evaluated to **true** with respect to $\mathcal{V}_i$. The $(j+1)$th iteration is as described above for the case $i = 0$ (only that the X_{q_k} variables can be free and have their assignments in $\mathcal{V}_i$). Since Q_i is rejecting, this set is W_q as required (indeed, a run cannot stay in Q_i forever). When $\lambda_0 = \nu$, the valuation of the greatest fixed point of E_i is such that at the end of the first iteration, each variable X_q, for $q \in Q_i$, contains the set of states w for which the formula obtained from $\delta(q, L(w))$ by replacing variables that correspond to states in Q_i with **true** is not evaluated to **false** with respect to $\mathcal{V}_i$. Again, the $(j+1)$th iteration is as described above for the case $i = 0$. Since Q_i is accepting, this set is W_q as required. Indeed, now the run can get stuck in Q_i or move to a lower set from which it accepts. □

Remark 3.2. It is proved in Kupferman and Vardi [2001] that alternating Büchi word automata can be translated to weak alternating 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 Arnold and Niwiński [1992] and Vardi and Wolper [1994].

We now characterize ω-regular languages $\mathcal{L}$ for which $der(\mathcal{L})$ can be characterized by an AFMC formula. Our characterization involves DBW, and we first need the following lemma.

Lemma 3.3. *Given a DBW $\mathcal{A}$ with n states, there is a symmetric AWT $\mathcal{A}'$ with $O(n)$ states such that $\mathcal{L}(\mathcal{A}') = der(\mathcal{L}(\mathcal{A}))$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_0, M, \alpha \rangle$. We first translate $\mathcal{A}$ into an NWW $\mathcal{U}$ for $\Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$, then expand $\mathcal{U}$ to a symmetric NWT $\mathcal{U}'$ that accepts a tree iff it contains a path labeled by a word in $\mathcal{L}(\mathcal{U})$. The automaton $\mathcal{A}'$ is then obtained by dualizing $\mathcal{U}'$. Equivalently, one could use Kupferman and Vardi [2001] in order to translate $\mathcal{A}$ into a UWW and then expand the UWW into an AWT for $der(\mathcal{L}(\mathcal{A}))$.

Intuitively, $\mathcal{U}$ accepts a word w not in $\mathcal{L}(\mathcal{A})$ by guessing the position when the single run of $\mathcal{A}$ on w no longer visits states in α. For that, $\mathcal{U}$ has two copies of $\mathcal{A}$, where the second copy does not contain states in α. A run of $\mathcal{U}$ is accepting if it eventually moves to the second copy, from which is cannot return to the first copy [Kurshan 1987]. Formally,

$$\mathcal{U} = \langle \Sigma, (Q \times \{1\}) \cup ((Q \setminus \alpha) \times \{2\}), \langle q_0, 1\rangle, \delta, (Q \setminus \alpha) \times \{2\} \rangle,$$

where for all $q \in Q$ and $\sigma \in \Sigma$, we distinguish between two cases. If $M(q, \sigma) = q' \notin \alpha$, then $\delta(\langle q, 1\rangle, \sigma) = \{\langle q', 1\rangle, \langle q', 2\rangle\}$ and $\delta(\langle q, 2\rangle, \sigma) = \{\langle q', 2\rangle\}$. Otherwise (that is, $M(q, \sigma) = q' \in \alpha$, then $\delta(\langle q, 1\rangle, \sigma) = \{\langle q', 1\rangle\}$ and $\delta(\langle q, 2\rangle, \sigma) = \emptyset$. It is easy to see that $\mathcal{U}$ is indeed weak. Now, the symmetric NWT is obtained from $\mathcal{U}$ by replacing a transition $\delta(s, \sigma) = S$ by the transition $\delta(s, \sigma) = \bigvee_{s' \in S} (\diamond, s')$. □

THEOREM 3.4. *Given an ω-regular language $\mathcal{L}$, the following are equivalent.*

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

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

PROOF. Assume first that *der*($\mathcal{L}$) has an equivalent AFMC formula. Then, by Theorem 3.1, *der*($\mathcal{L}$) can be recognized by a symmetric AWT, and $der_2(\mathcal{L})$ can be recognized by the restriction of this automaton to trees of branching degree 2. It is proved in Muller et al. [1986] that if a language of trees can be recognized by an AWT, then it can also be recognized by an NBT. It follows that $der_2(\mathcal{L})$ can be recognized by an NBT. By Kupferman et al. [1996], for every ω-regular language $\mathcal{R}$, if $der_2(\mathcal{R})$ can be recognized by an NBT, 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 $\mathcal{L}$. As described in Lemma 3.3, we can translate $\mathcal{A}$ to a symmetric AWT $\mathcal{A}'$ for *der*($\mathcal{L}$). By Theorem 3.1, we can then translate $\mathcal{A}'$ into an AFMC formula, and we are done.

For completeness, we describe below the direct construction of the AFMC formula. 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 $\psi_{\mathcal{L}}$. Recall that $\Sigma = 2^P$ for some set of atomic propositions. The formula $\psi_{\mathcal{L}}$ has P 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:

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

$$X'_q = \begin{bmatrix} \textbf{true} & \text{if } q \in \alpha, \\ \bigvee_{\sigma \in \Sigma} \sigma \wedge \forall \bigcirc X'_{\delta(q,\sigma)} & \text{if } q \notin \alpha. \end{bmatrix}$$

The set of equations is partitioned into two blocks. Equations with a left-hand side variable X_q constitute a greatest fixed-point block $\nu\{E_1\}$, and equations with a left had side variable X'_q constitute a least fixed-point block $\mu\{E_2\}$.

We prove that for every system S, we have *tree*(S) $\in$ *der*($\mathcal{L}$) iff S satisfies $\psi_{\mathcal{L}}$. For a state $q \in Q$, let $W_q \subseteq W$ be such that $w \in W_q$ iff all the computations of S that start in w are accepted by the automaton $\mathcal{A}$ with initial state q. We prove that the greatest fixed point of the equations in E_1 is obtained by assigning W_q to the variable X_q.

It is easy to see that the least fixed point of the equations in E_2 is obtained by assigning to a variable X'_q the set of states w 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. □

Example 3.5. Consider the DBW $\mathcal{A}$ for the LTL formula $\Box \Diamond p$. Formally, $\mathcal{A} = \langle \{\{p\}, \emptyset\}, \{s, t\}, s, \delta, \{t\} \rangle$, with $\delta(s, \{p\}) = \delta(t, \{p\}) = t$ and $\delta(s, \emptyset) = \delta(t, \emptyset) = t$. The equivlanet AFMC formula generated as above consists of the following two blocks (simplified by replacing X_t' with **true**).

$$\nu \left\{ \begin{array}{l} X_s = (p \wedge \forall \bigcirc X_t) \vee (\neg p \wedge \forall \bigcirc X_s \wedge \forall \bigcirc X_s'). \\ X_t = (p \wedge \forall \bigcirc X_t) \vee (\neg p \wedge \forall \bigcirc X_s \wedge \forall \bigcirc X_s'). \end{array} \right\}$$

$$\mu\{X_s' = (p \vee \forall \bigcirc X_s')\}.$$

Remark 3.6. Note that the AWT constructed in Lemma 3.3 is actually a UWT. As can also be seen in the direct construction of $\psi_{\mathcal{L}}$ described above, this implies that the AFMC formula we get is in the universal fragment ∀*AFMC* of AFMC, in which the only next-time operator allowed is $\forall\bigcirc$. It follows that ∀AFMC is sufficiently strong to express all LTL formulas that have an equivalent AFMC formula. We note that while the above may seem expected, the corresponding problem is still open for CTL. That is, it is still not known whether the intersection of LTL and universal fragment ∀CTL of CTL is a strict fragment of the intersection of LTL and CTL [Maidl 2000].

Remark 3.7. Note that while DBW is not closed under complementation, the AFMC is closed under complementation. When, however, we complement a language $der(\mathcal{L})$, we do not get the language $der(\tilde{\mathcal{L}})$. Rather, we get the language of trees in which at least one path is in $\tilde{\mathcal{L}}$.

In Sections 4 and 5, we use the characterization and the translations above in order to study the transition from various linear-time formalisms to the AFMC.

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 3.4, 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 $w[1 \cdots n]$ 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 Krishnan et al. [1994]. Consider a DPW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$. We say that a set of states $S \subseteq Q$ is *accepting* if every run r with $inf(r) = S$ satisfies the acceptance condition α. A state $q \in Q$ is called *final* iff all the cycles that visit q are accepting. According to Landweber [1969], 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 Krishnan et al. [1994] 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 $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$. As suggested in Krishnan et al. [1994], the latter check can be performed by checking whether the automaton obtained from $\mathcal{A}$ by deleting its final states is nonempty. 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.

We now prove the lower bounds. We start with DPW, where we do a reduction from the graph reachability problem, proved to be NLOGSPACE-hard in Jones [1975]. Given a directed graph $G = \langle V, E\rangle$ and two designated nodes s and t, we describe a DPW $\mathcal{A}_G$ such that $\mathcal{A}_G$ is in DBW iff t is not reachable from s in G. Let $\mathcal{A} = \langle\{0, 1\}, \{q_0, q_1\}, q_0, \delta, \{\{q_0\}, \{q_1\}\}\rangle$ be a DPW for $\mathcal{L}_{fm}$ (from both states, the transition function δ moves to q_0 when it reads 0 and moves to q_1 when it reads 1). Intuitively, $\mathcal{A}_G$ is obtained by concatenating $\mathcal{A}$ to the vertex t of G. Let d be the maximal branching degree in G. The alphabet of $\mathcal{A}_G$ is $\{0, 1, \ldots, d + 1\}$. Its state space is $V \cup \{q_0, q_1, v_{acc}\}$, where v_{acc} is a new state. The d letters $2, \ldots, d + 1$ label the edges of G so that all the edges that leave a vertex are labeled differently. In addition, if a vertex v has less than d successors, we add an edge from v to v_{acc} labeled by all the letters not yet labeling edges that leave v. The state v_{acc} has a self loop labeled by all the letters $2, \ldots, d + 1$. An edge labeled 1 is added from t to the state q_1 of $\mathcal{A}$. Finally, the initial state of $\mathcal{A}_G$ is s, and its parity acceptance condition is $\{\{q_0\}, V \cup \{q_1, v_{acc}\}\}$. It is easy to see that $\mathcal{A}_G$ is a DPW and that $\mathcal{L}(\mathcal{A}_G) \subseteq \{2, \ldots, d + 1\}^\omega \cup \{2, \ldots, d + 1\} \cdot \mathcal{L}_{fm}$. We prove that $\mathcal{A}_G$ is in DBW iff t is not reachable from s.

Assume first that t is not reachable from s. For all $v \in V \cup \{v_{acc}\}$ and $\sigma \in \{2, \ldots, d_1\}$, there is a transition from v to $V \cup \{v_{acc}\}$ labeled σ. Also, as long as the run stays in $V \cup \{v_{acc}\}$, it cannot read the letters 0 or 1. Since a run of $\mathcal{A}_G$ that stays forever in $V \cup \{v_{acc}\}$ is accepting, it follows that when t is not reachable from s, the language of $\mathcal{A}_G$ is $\{2, \ldots, d + 1\}^\omega$, so $\mathcal{A}_G$ is in DBW. Assume now that t is reachable from s. Let $w \in \{2, \ldots, d + 1\}^*$ be a word labeling a path from s to t. Assume, by way of contradiction, that $\mathcal{L}(\mathcal{A}_G)$ is in DBW. Then, by Landweber [1969], there exists a regular language $\mathcal{F}$ such that $\mathcal{L}(\mathcal{A}_G) = lim(\mathcal{F})$; that is $\mathcal{L}(\mathcal{A}) = \{w : w$ has infinitely many prefixes in $\mathcal{F}\}$. Since $w \cdot 1^\omega$ is in $\mathcal{L}(\mathcal{A}_G)$, there exists some p_1 such that $w \cdot 1^\omega[1 \cdots p_1]$ is in $\mathcal{F}$. Since $w \cdot 1^{p_1} \cdot 0 \cdot 1^\omega$ is in $\mathcal{L}(\mathcal{A})_G$, there exists some p_2 such that $w \cdot 1^{p_1} \cdot 0 \cdot 1^\omega[1 \cdots p_1 + 1 + p_2]$ is in $\mathcal{F}$. We can continue and obtain an infinite sequence of finite words $w \cdot 1^{p_1} \cdot 0 \cdot 1^{p_2} \cdot 0 \cdots 0 \cdot 1^{p_{k-1}} \cdot 0 \cdot 1^\omega[1 \cdots p_1 + 1 + p_2 + 1 + \cdots + 1 + p_k]$ $(k = 1, 2, 3, \ldots)$, all in $\mathcal{F}$. Hence, the infinite word $w \cdot 1^{p_1} \cdot 0 \cdot 1^{p_2} \cdot 0 \cdot 1^{p_3} \cdot 0 \cdot 1^{p_4} \cdots$ is in $lim(\mathcal{F})$ and thus in $\mathcal{L}(\mathcal{A}_G)$. On the other hand, $\mathcal{L}(\mathcal{A}_G)$ contains only words with finitely many 0's, and we reach a contradiction.

It is left to prove the lower bound for Rabin and Streett automata. For Streett automata, we do a reduction from DSW emptiness, proved to be PTIME-complete in Emerson and Lei [1985]. 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., Thomas [1990]), 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 Landweber [1969], there exists a regular language $\mathcal{F}$ such that $\mathcal{L}(\mathcal{A}) = lim(\mathcal{F})$. 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 $w.1^\omega[1 \cdots p_1]$ 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 $w.1^{p_1} \cdot 0 \cdot 1^\omega[1 \cdots p_1 + 1 + p_2]$ is in $\mathcal{F}$. We can continue and obtain an infinite sequence of finite words $w.1^{p_1} \cdot 0 \cdot 1^{p_2} \cdot 0 \cdots 0 \cdot 1^{p_{k-1}} \cdot 0 \cdot 1^\omega[1 \cdots p_1 + 1 + p_2 + 1 + \cdots + 1 + p_k]$ ($k = 1, 2, 3, \ldots$), 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,

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

We prove that $\mathcal{R}$ is universal iff $\mathcal{A}$ is DBW. First, if $\mathcal{R}$ is universal, so is $\mathcal{A}$, and hence it is clearly in DBW. Assume now that $\mathcal{R}$ is not universal, 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, only that, similarly to the case of Streett detailed above, we have to accompany the words in $\{0,1\}^\omega$ with some word rejected by $\mathcal{R}$. □

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. Since the translation can be done on-the-fly, it would follow from Theorem 4.1 that 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 Safra [1988, 1989], 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 Krishnan et al. [1995b] 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 Meyer and Stockmeyer [1972], and Wolper [1982]. 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. □

By Theorems 3.4, 4.1, and 4.2, we have the following corollary.

COROLLARY 4.3

(1) Deciding DPW $\mapsto$ AFMC is NLOGSPACE-complete.
(2) Deciding {*DRW*, *DSW*} $\mapsto$ AFMC is PTIME-complete.
(3) Deciding {*NBW*, *NPW*, *NRW*, *NSW*} $\mapsto$ AFMC is PSPACE-complete.

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. For a blow-up f (e.g., linear, exponential), we say that a translation is *tightly* f if f is both an upper and lower bound for the blow-up of the translation.

THEOREM 4.4. *When possible, the translation*

(1) {*DBW, DPW, DRW*} $\mapsto$ *AFMC is tightly linear.*
(2) *DSW* $\mapsto$ *AFMC is tightly polynomial.*
(3) {*NBW, NPW, NRW, NSW*} $\mapsto$ *AFMC is tightly exponential.*

PROOF. A linear translation of DBW to AFMC is described in the proof of Theorem 3.4. It is proved in Krishnan et al. [1994], 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 possible, involves a polynomial blow-up Krishnan et al. [1995a]. Hence the bounds for DPW, DRW, and DSW. All the types of nondeterministic automata $\mathcal{A}$ in **(3)** have an exponential translation to DRW [Safra 1988, 1992]. Therefore, by **(1)**, if $\mathcal{A}$ is in DBW, it has an equivalent AFMC formula of exponential length. □

By Wilke [1999], the exponential blow-ups in Theorem 4.4 cannot be improved: the LTL specification $\psi_n = \Box p_1 \vee \Box p_2 \vee \cdots \vee \Box p_n$ has an NBW of size $O(n)$ but the smallest AFMC formula for it is of length exponential in n [Wilke 1999].

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 Vardi and Wolper [1994], $\mathcal{B}_\psi$ 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. We prove that ψ is not satisfiable iff $\psi \wedge \Diamond\Box p$ is in DBW. Assume first that ψ is not satisfiable. Then, $\psi \wedge \Diamond\Box p$ is not satisfiable as well and is in DBW. Assume now that $\psi \wedge \Diamond\Box p$ is in DBW. Then, by Landweber [1969], there exists a regular language $\mathcal{F}$ such that $\psi \wedge \Diamond\Box p = lim(\mathcal{F})$. Assume, by way of contradiction, that ψ is satisfiable. Let π be a computation in $(2^P)^\omega$ that satisfies ψ. Then, using the same arguments as in the proof of Theorem 4.1, we can annotate the labels of some of the states along π with p and get a computation $\pi' \in (2^{P\cup\{p\}})^\omega$ that does not satisfy $\Diamond\Box p$ and still belongs to $lim(\mathcal{F})$. □

By Theorems 3.4 and 5.1, we have the following corollary.

COROLLARY 5.2. *Deciding LTL $\mapsto$ AFMC is in EXPSPACE and is PSPACE-hard.*

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

THEOREM 5.3. *When possible, the translation LTL $\mapsto$ DBW is tightly doubly exponential.*

PROOF. Let ψ be an LTL formula of length n and let $\mathcal{B}_\psi$ be an NBW that recognizes ψ [Vardi and Wolper 1994]. The automaton $\mathcal{B}_\psi$ has 2^n states. By determinizing $\mathcal{B}_\psi$, we get a DRW $\mathcal{R}_\psi$ with $2^{2^{O(n)}}$ states [Safra 1988]. By Krishnan et al. [1994], if $\mathcal{R}_\psi$ is in DBW, it can be translated to a DBW with $2^{2^{O(n)}}$ states. Hence, the upper bound.

For the lower bound, consider the regular language

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

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 Chandra et al. [1981], the smallest deterministic automaton on finite words that accepts $\mathcal{L}_n$ has at least 2^{2^n} states. (The proof in Chandra et al. [1981] considers the language of the finite words obtained from $\mathcal{L}_n$ by omitting the $\#^\omega$ suffix. The proof, however, is independent of this technical detail: 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. 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 ith letter after the \$. Formally, the formula is

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

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. $\Box$

Translating LTL to AFMC by going through DBW involves a doubly exponential blow up. Closing this gap is an open problem.

THEOREM 5.4. *When possible, the translation LTL $\mapsto$ AFMC is doubly exponential and is at least exponential.*

PROOF. Since, by Theorem 4.4, DBW can be linearly translated to AFMC, the upper bound follows from Theorem 5.3. Also, as proved in Wilke [1999], the smallest AFMC formula equivalent to $\psi_n = \Box p_1 \vee \Box p_2 \vee \cdots \vee \Box p_n$ is of length exponential in n. $\Box$

Remark 5.5. While the tightest upper bound known for the translation of LTL to AFMC is, as described in Theorem 5.4, doubly exponential, many LTL formulas that are used in practice have equivalent AFMC formulas of linear size. In particular, for many LTL formulas, prefixing each temporal operator with the universal path quantifier $\forall$ results in an equivalent $\forall$CTL formula, which can be easily and linearly translated into the AFMC. For example, the LTL formula $\Box\Diamond p$ is equivalent to the $\forall$CTL formula $\forall\Box\forall\Diamond p$, and hence also to the AFMC formula $\nu X.((\mu Y.p \vee \forall\bigcirc Y) \wedge \forall\bigcirc X)$. The problem of deciding whether an LTL formula ψ is equivalent to the $\forall$CTL formula $\psi_\forall$ obtained as above is PSPACE-complete, and a syntactic characterization of such formulas is given in Maidl [2000]. For other LTL formulas ψ, it may be that ψ is not equivalent to $\psi_\forall$ and still ψ has an equivalent short CTL formula. For example, the LTL formula $\bigcirc p \vee \bigcirc q$ is not equivalent to $\forall\bigcirc p \vee \forall\bigcirc q$, but is equivalent to the $\forall$CTL formula $\forall\bigcirc(p \vee q)$. Finally, some LTL formulas do not have equivalent CTL formulas, but do have short equivalent AFMC formulas. For example, the formula $\Diamond(p \wedge \bigcirc p)$ does not have an equivalent CTL formula [Emerson and Halpern 1986] and is equivalent to the AFMC formula $\mu X.((\neg p \wedge \forall\bigcirc X) \vee (p \wedge \forall\bigcirc \mu Y.(p \vee \forall\bigcirc X)))$. An additional discussion on LTL formulas that have short equivalent formulas can be found in Kupferman and Vardi [1998], and Bloem et al. [1999].

6. DISCUSSION

We considered the translation of linear-time specification formalisms into AFMC. We first showed that AFMC is as expressive as AWT, and concluded that a linear property ψ can be specified in AFMC iff ψ can be recognized by a DBW. We then studied the problem of deciding, for a specification formalism

$\mathcal{F}$	Deciding $\mathcal{F} \mapsto$ AFMC	Blow-up translating $\mathcal{F} \mapsto$ AFMC
DBW	always exists	linear
DPW	NLOGSPACE	linear
DRW	PTIME	linear
DSW	PTIME	polynomial
N{B,P,R,S}W	PSPACE	exponential
LTL	EXPTIME (upper bound) PSPACE (lower bound)	doubly exponential (upper bound) exponential (lower bound)

Fig. 1. Summary of results.

$\mathcal{F}$, whether a property in $\mathcal{F}$ can be translated into the AFMC, and the blow up that such a translation may involve. Our results are summarized in Figure 1.

All the results except for these about LTL are tight. In the case of LTL, the gaps follow from the fact that the best approach we now have is to first translate the LTL property into an NBW, then check whether the NBW can be translated into a DBW, and finally obtain an AFMC formula from this DBW, when possible. While the translation of LTL into DBW is tightly doubly exponential, it may well be that one could circumvent such a translation. In particular, an improvement of the upper bound would follow from an exponential translation of LTL into UBW (when possible). Indeed, the linear translation of DBW into AFMC described in Theorem 3.4 can be applied also for UBW. While UBW are as expressive as DBW, they are exponentially more succinct (in fact, as NFW are exponentially more succinct than DFW [Rabin and Scott 1959], the above holds already for automata on finite words, thus UFW are exponentially more succinct than DFW). An equivalent open problem is to find an exponential translation of LTL into nondeterministic co-Büchi automata (when possible). Given the ease of symbolic AFMC model checking, a positive reply to these open problems could be of practical interest.

REFERENCES

Arnold, A. and Niwiński, D. 1992. Fixed point characterization of weak monadic logic definable sets of trees. In *Tree Automata and Languages*, M. Nivat and A. Podelski, Eds. Elsevier, Amsterdam, The Netherlands, 159–188.

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

Bloem, R., Ravi, K., and Somenzi, F. 1999. Efficient decision procedures for model checking of linear time logic properties. In *Computer-Aided Verification, Proceedings of the 11th International Conference*. Lecture Notes in Computer Science, vol. 1633. Springer-Verlag, New York, 222–235.

Brayton, R., Hachtel, G., Sangiovanni-Vincentelli, A., Somenzi, F., Aziz, A., Cheng, S.-T., Edwards, S., Khatri, S., Kukimoto, T., Pardo, A., Qadeer, S., Ranjan, R., Sarwary, S., Shiple, T., Swamy, G., and Villa, T. 1996. VIS: a system for verification and synthesis. In *Computer-Aided Verification, Proceedings of the 8th International Conference*. Lecture Notes in Computer Science, vol. 1102. Springer-Verlag, New York, 428–432.

Bryant, R. 1986. Graph-based algorithms for Boolean-function manipulation. *IEEE Trans. Comput. C-35*, 8.

Büchi, J. 1962. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Method, and Philosophy of Science*. Stanford University Press, Stanford, Calif., 1–12.

BURCH, J., CLARKE, E., MCMILLAN, K., DILL, D., AND HWANG, L. 1992. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput. 98*, 2 (June), 142–170.

CADENCE. SMV. http://www.cadence.com/company/cadence_labs_research.html.

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

CLARKE, E. AND DRAGHICESCU, I. 1988. Expressibility results for linear-time and branching-time logics. In *Proceedings of the Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, J. de Bakker, W. de Roever, and G. Rozenberg, Eds. Lecture Notes in Computer Science, vol. 354. Springer-Verlag, New York, 428–437.

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

CLARKE, E., GRUMBERG, O., AND PELED, D. 1999. *Model Checking*. MIT Press.

CLEAVELAND, R. AND STEFFEN, B. 1991. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. In *Proceedings of the 3rd International Conference on Computer-Aided Verification*. Lecture Notes in Computer Science, vol. 575. Springer-Verlag, New York, 48–58.

EMERSON, E. AND HALPERN, J. 1986. Sometimes and not never revisited: On branching versus linear time. *J. ACM 33*, 1, 151–178.

EMERSON, E. AND LEI, C.-L. 1985. Temporal model checking under generalized fairness constraints. In *Proceedings of the 18th Hawaii International Conference on System Sciences*. Western Periodicals Company, North Hollywood, Calif.

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

HAFER, T. AND THOMAS, W. 1987. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proceedings of the 14th International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, vol. 267. Springer-Verlag, New York, 269–279.

HARDIN, R., HAR'EL, Z., AND KURSHAN, R. 1996. COSPAN. In *Computer Aided Verification, Proceedings of the 8th International Conference*. Lecture Notes in Computer Science, vol. 1102. Springer-Verlag, New York, 423–427.

HARDIN, R., KURSHAN, R., SHUKLA, S., AND VARDI, M. 2001. A new heuristic for bad cycle detection using BDDs. *Form. Meth. Syst. Des. 18*, 131–140.

HOLZMANN, G. 1997. The model checker SPIN. *IEEE Trans. Softw. Eng. 23*, 5 (May), 279–295. (Special issue on Formal Methods in Software Practice.)

JANIN, D. AND WALUKIEWICZ, I. 1995. Automata for the modal μ-calculus and related results. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science*. Lecture Notes in Computer Science. Springer-Verlag, New York, 552–562.

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

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

KRISHNAN, S., PURI, A., AND BRAYTON, R. 1994. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*. Lecture Notes in Computer Science, vol. 834. Springer-Verlag, New York, 378–386.

KRISHNAN, S., PURI, A., AND BRAYTON, R. 1995a. Structural complexity of ω-automata. In *Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, vol. 900. Springer-Verlag, New York.

KRISHNAN, S., PURI, A., BRAYTON, R., AND VARAIYA, P. 1995b. The Rabin index and chain automata, with applications to automata and games. In *Computer Aided Verification, Proceedings of the 7th International Conference* (Liege). 253–266.

KUPFERMAN, O. AND GRUMBERG, O. 1996. Buy one, get one free!!! *J. Logic Comput. 6*, 4, 523–539.

KUPFERMAN, O., SAFRA, S., AND VARDI, M. 1996. Relating word and tree automata. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., 322–333.

KUPFERMAN, O. AND VARDI, M. 1998. Relating linear and branching model checking. In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods*. Chapman & Hall, New York, 304–326.

KUPFERMAN, O. AND VARDI, M. 2001. Weak alternating automata are not that weak. *ACM Trans. Computat. Logic 2*, (July), 408–429.

KUPFERMAN, O., VARDI, M., AND WOLPER, P. 2000. An automata-theoretic approach to branching-time model checking. *J. ACM 47*, 2 (Mar.), 312–360.

KURSHAN, R. 1987. Complementing deterministic Büchi automata in polynomial time. *J. Comput. Syst. Sci. 35*, 59–71.

KURSHAN, R. 1994. *Computer Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, N.J.

LANDWEBER, L. 1969. Decision problems for ω-automata. *Math. Syst. Theory 3*, 376–384.

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

MAIDL, M. 2000. The common fragment of CTL and LTL. In *Proceedings of the 41th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., 643–652.

MCMILLAN, K. 1993. *Symbolic Model Checking*. Kluwer Academic Publishers.

MEYER, A. AND STOCKMEYER, L. 1972. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*. IEEE Computer Society Press, Los Alamitos, Calif., 125–129.

MULLER, D., SAOUDI, A., AND SCHUPP, P. 1986. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proceedings of the 13th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 226. Springer-Verlag, New York.

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

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

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

RABIN, M. AND SCOTT, D. 1959. Finite automata and their decision problems. *IBM J. Res. Devel. 3*, 115–125.

SAFRA, S. 1988. On the complexity of ω-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science* (White Plains, N.Y.). IEEE Computer Society Press, Los Alamitos, Calif., 319–327.

SAFRA, S. 1989. Complexity of automata on infinite objects. Ph.D. dissertation, Weizmann Institute of Science, Rehovot, Israel.

SAFRA, S. 1992. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proceedings of the 24th ACM Symposium on Theory of Computing* (Victoria, B.C., Canada). ACM, New York.

SCHNEIDER, K. 1997. CTL and equivalent sublanguages of CTL*. In *Proceedings of IFIP Conference on Computer Hardware Description Languages and Applications* (Toledo, Ohio). Chapman and Hall, Toledo, 40–59.

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

VARDI, M. AND WOLPER, P. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science*. Cambridge, 332–344.

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

WILKE, T. 1999. CTL$^+$ is exponentially more succinct than CTL. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, C. P. Ragan, V. Raman, and R. Ramanujam, Eds. Lecture Notes in Computer Science, vol. 1738. Springer-Verlag, New York, 110–121.

WOLPER, P. 1982. Synthesis of communicating processes from temporal logic specifications. Ph.D. dissertation, Stanford University, Stanford, Calif.

Received September 2002; revised September 2003; accepted September 2003