Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 32, No. 2, April 1986
Printed in Belgium

Automata-Theoretic Techniques for Modal Logics of Programs*

MOSHE Y. VARDI†

IBM Research Laboratory, San Jose, California

AND

PIERRE WOLPER‡

AT&T Bell Laboratories, Murray Hill, New Jersey

Received September 11, 1984; revised May 23, 1985

We present a new technique for obtaining decision procedures for modal logics of programs. The technique centers around a new class of finite automata on infinite trees for which the emptiness problem can be solved in polynomial time. The decision procedures then consist of constructing an automaton A_f for a given formula f, such that A_f accepts some tree if and only if f is satisfiable. We illustrate our technique by giving exponential decision procedures for several variants of deterministic propositional dynamic logic.

1. INTRODUCTION

Propositional modal logics of programs are formal systems for reasoning about the behavior of program schemes. They are of two different types: dynamic logics, a la Pratt [19], are used for reasoning about the input/output behavior of program schemes, while temporal logics, a la Pnueli [18] are used for reasoning about their ongoing behavior. Most of the propositional program logics studied in the literature are known to have a decidable satisfiability problem. A general technique to show their decidability is by reduction to *SnS*, the second-order theory of n successor functions [7]. Rabin has shown that *SnS* is decidable [24], but the upper bound established by that reduction is, unfortunately, nonelementary [15].

For several of these logics exponential time upper bounds have been established using the so-called *small model property*. This property, established first for propositional dynamic logic [6], says that if a formula of length n is satisfiable, i.e., if it has a model, then it also has a "small model," i.e., a model whose cardinality is at most exponential in n. While this property by itself gives only a nondeterministic

*This article is significantly different from its preliminary version that appeared *in* "Proc. 16th ACM Sympos. on Theory of Computing," Washington, 1984, pp. 446–456.

†Address: IBM Research K55/281, 5600 Cottle Rd., San Jose, CA 95193.
‡Address: AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974.

0022-0000/86 $3.00

While Büchi automata are indeed powerful enough for the reduction from satisfiability to emptiness to work, the reduction turns out to be quite cumbersome. To simplify things we introduce a new type of automata, which we call *subtree automata*. Subtree automata are automata that check that under every node in the tree there exists a certain finite subtree. With subtree automata, the reduction of satisfiability to emptiness is quite straightforward. Moreover, we show that subtree automata can be translated into Büchi automata, with only a quadratic increase in size. Thus emptiness of subtree automata can be checked in polynomial time, and the reduction establishes the desired exponential upper bound for satisfiability.

The resulting technique turns out to be powerful and unifying. It enables us to supply simpler proofs for known results and to obtain many new exponential upper bounds. The power of the technique lies in the fact that it abstracts the logical issues into an automata theoretic framework. Once this abstraction is done, we can prove results for several distinct logics by a single automata theoretic argument. For example, the proof that emptiness of Büchi automata can be checked in polynomial time relies on an unwinding argument. This unwinding corresponds to the unwinding of pseudo-models to models. It is done here, however, in an automata-theoretic framework, with no need to take the intricacies of the logic into account, and it is done once and for all, with no need to repeat it for every logic.

Another advantage is that this technique does not depend on the small model property (or small pseudo-model property). This property is usually established by the *filtration* technique. The essence of this technique is the identification of nodes in a Kripke structure that satisfy the same formulas. While the filtration technique works for several logic, it fails for several others [7, 28, 29]. The tree model property, on the other hand, seems to be much more basic, and it holds in cases where the filtration technique fails [5, 7, 28, 29]. Thus automata based decision procedures have a wider applicability.

We demonstrate our technique with three proofs. We first prove an exponential upper bound for *ADPDL* (deterministic *PDL* of flowcharts). Our proof is significantly simpler than the original proof in [1]. We then prove an exponential upper bound for *loop-ADPDL*, which is the extension of *ADPDL* by the *loop* construct. (Intuitively, *loop*(α) means that the program α may loop.) Again, the proof is significantly simpler than the proof in [23] for the nodeterministic version of *loop-ADPDL*. Finally, we prove an exponential upper bound for *converse-ADPDL*, which is the extension of *ADPDL* with the *converse* construct. (Intuitively, the converse of a program α is a program that runs the computation of α backwards.) The *converse* construct poses special problems for our technique, and unlike the first two proofs the last proof is quite involved.

The automata-theoretic techniques presented here are closely related to, but significantly different from, the techniques in the preliminary version of this paper [32]. We discuss these differences in the Appendix.

2. Automata on Infinite Trees

Before defining the classes of automata on infinite trees we are interested in and examining their emptiness problem, we will, to make our notation clear, define classical sequential automata and some technical notions concerning infinite trees.

2.1. *Sequential Automata*

A sequential automaton is a tuple $A = (\Sigma, S, \rho, s_0, F)$

- Σ is the alphabet.
- S is a set of states.
- $\rho: S \times \Sigma \to 2^S$ is the transition function. For each state and letter it gives the possible successors.
- s_0 is the initial state.
- $F \subseteq S$ is a set of accepting states.

We extend ρ to Σ^* in the following way: $\rho(s, \lambda) = \{s\}$ (λ is the empty string), and $\rho(s, wa) = \{t: t \in \rho(s', a)$ for some $s' \in \rho(s, w)\}$. A *accepts* a word $w \in \Sigma^*$ if $\rho(s_0, w) \cap F \neq \varnothing$.

Given an automation $A = (\Sigma, S, \rho, s_0, F)$, for each $s \in S$, we define A_s, to be the automaton (Σ, S, ρ, s, F), i.e., the automaton A where the state s replaces s_0 as the initial state. Similarly, for states $s, t \in S$, we define A_s^t to be the automaton $(\Sigma, S, \rho, s, \{t\})$.

2.2. *Infinite Trees*

We need to define some technical notions concerning n-ary trees. Let $[n]$ denote the set $\{1,..., n\}$. An n-ary tree T is a labeling of the set $[n]^*$ by letters from an alphabet Σ, that is $T: [n]^* \to \Sigma$. If $x, y \in [n]^*$, then

- If $x = \lambda$ then x is called the *root* of the tree.
- x is the *predecessor* of y and y is the *successor* of x, if $y = xi$ for some $i \in [n]$.
- x *precedes* y and y *succeeds* x, denoted $x \leqslant y$, if there is $z \in [n]^*$ such that $y = xz$.
- x *properly precedes* y and y *properly succeeds* x, denoted $x < y$, if x precedes y and $x \neq y$.

A *path* starting at a node $x \in [n]^*$ is an infinite set $x_0, x_1,...$ such that $x_0 = x$ and x_{i+1} is a successor of x_i for all $i \geqslant 0$. For a tree $T: [n]^* \to \Sigma$ and a path p, $\inf(T, p)$ is the set of labels that appear infinitely often on p. That is,

$$\inf(T, p) = \{\sigma: \text{for all } x \in p \text{ there is } y \in p \text{ such that } x \leqslant y \text{ and } T(y) = \sigma\}.$$

2.3. *Büchi Automata*

A *Büchi automaton* on n-ary trees is a tuple $A = (\Sigma, S, \rho, S_0, F)$, where

- Σ is the alphabet.
- S is a set of states.

- $\rho: S \times \Sigma \to 2^{S^n}$ is the transition function. For each state and letter it gives the possible sets of n successors.
- $S_0 \subset S$ is the set of initial states.
- $F \subseteq S$ is a set of designated states. As we will see later, F defines the acceptance condition.

In the sequel we assume that our automata run on n-ary trees without mentioning it explicitly.

A *run* of an automaton A over a tree $T: [n]^* \to \Sigma$ is an n-ary tree $\phi: [n]^* \to S$, where $\phi(\lambda) \in S_0$ and for every $x \in [n]^*$, we have $(\phi(x1),..., \phi(xn)) \in \rho(\phi(x), T(x))$. A run ϕ of A over T is *accepting* if and only if, for all infinite paths p starting at λ we have $\inf(\phi, p) \cap F \neq \emptyset$. *A accepts T* if it has an accepting run on T. $T(A)$ is the set of trees accepted by A.

This kind of automaton was defined in [25] under the name *special automata*. We call them Büchi automata in honor of Büchi, since the acceptance condition is similar to the one used by Büchi [3] for automata on infinite words.

Other kinds of automata on infinite trees have been defined by changing the acceptance condition [24, 28, 29]. In *Müller automata*, rather then having a set F of designated states, we have a collection $\mathbf{F} \subset 2^S$ of designated sets. A run ϕ of A over T is accepting if and only if, for all infinite paths p starting at λ we have $\inf(\phi, p) \in \mathbf{F}$. Müller used the notion of designated sets to define acceptance of automata on infinite words [16]. Rabin was the first to use it for trees [24].

In *Rabin automata* [26], rather then having designated sets, we have a collection $\mathbf{F} \subset 2^S \times 2^S$ of designated pairs of sets. A run ϕ of A over T is accepting if and only if for all infinite paths p starting at λ we have that $\inf(\phi, p) \cap X = \emptyset$ and $\inf(\phi, p) \cap Y \neq \emptyset$ for some $(X, Y) \in \mathbf{F}$.

In *Streett automata*, called *complemented pair automata* in [28, 29], we again have a collection of designated pairs, but the definition of acceptance is different. A run ϕ of A over T is accepting if and only if for all infinite paths p starting at λ we have that if $\inf(\phi, p) \cap X \neq \emptyset$ then $\inf(\phi, p) \cap Y \neq \emptyset$ for all $(X, Y) \in \mathbf{F}$.

It is easy to see that the definition of acceptance in Büchi automata is a special case of all the other definitions. Indeed, it follows from the results in [25] that Büchi automata are weaker than the other kinds of automata. That is, there exist a set Ψ of n-ary trees, a Müller automaton A_1, a Rabin automation A_2, and a Streett automaton A_3, such that $\Psi = T(A_1) = T(A_2) = T(A_3)$, but such that for no Büchi automaton A we have $\Psi = T(A)$.

The essential difference between Büchi acceptance and the other types of acceptance conditions is that in Büchi acceptance we only care about the states that appear infinitely often, while in the other conditions we also care about the states that do not appear infinitely often. The usefulness of Büchi acceptance in our context comes from the way that *eventualities* behave in modal program logics. An eventuality is a formula that requires that some other formula will eventually hold (e.g., *Fp* in temporal logic). In automata terms, it can be viewed as stating that if one goes through a given state (where the eventually is required) then one will go

through a state where it is satisfied. Given that for an eventuality the union of the set of states where it is required and the set of states where it is satisfied (or no longer required) is the whole set of states, this acceptance condition can be expressed as a Büchi acceptance condition.

2.4. *The Emptiness Problem*

The *emptiness problem* for a class of automata is to determine, given an automaton A in that class, whether there is any tree accepted by A. Algorithms for solving the emptiness problem for the different classes of tree automata were given in [24, 25, 10, 26]. For Rabin automata the best time upper bound is exponential [26], and for Müller and Streett automata it is doubly exponential [10, 28, 29].[2] The difficulty of the problem stems from having to care not only about the states that repeat infinitely often but also about the states that do not repeat infinitely often. On the other hand, for Büchi automata on *binary* trees, Rabin gave a polynomial time algorithm [25]. Here we consider Büchi automata on n-ary trees. We measure the complexity in the *size* of the automata, which is the length of the string describing them in some standard encoding. We first need some technical results.

A *subtree rooted at a node* $x \in [n]^*$ is a finite nonempty subset $W_x \subset [n]^*$ such that:

- if $y \in W_x$, then y succeeds x,
- if $y \in W_x$, and $yi \in W_x$ for some $i \in [n]$, then $yj \in W_x$ for all $j \in [n]$, and
- if $yi \in W_x$ and $yi \neq x$, then $y \in W_x$.

Note that in particular we must have $x \in W_x$. If $y \in W_x$ and there is some $i \in [n]$ such that $yi \in W_x$, then we say that y is an *internal node*, otherwise, y is a *leaf*.

We now prove a lemma that relates labels appearing infinitely often on the paths of a tree to the existence of some subtrees within that tree, As we will see, this will be useful to deal with acceptance conditions of automata on infinite trees.

LEMMA 2.1. *Let* $T: [n]^* \to S$ *be an* n*-ary infinite tree, and let* S' *be a subset of* S. *The following two conditions are equivalent*:

(1) *For every path* p *starting at* λ *we have* $\inf(T, p) \cap S' \neq \emptyset$.

(2) *For every* $x \in [n]^*$ *there exists a finite subtree* $W \subset [n]^*$, $|W| > 1$, *rooted at* x *such that if* $y \in W$ *is a leaf of* W, *then* $T(y) \in S'$.

Proof. (1) $\Rightarrow$ (2). Suppose that for every path p starting at λ we have $\inf(T, p) \cap S' \neq \emptyset$. For a given node x, let X be the set of minimal nodes y properly preceded by x such that $T(y) \in S'$; that is,

$$X = \{y: x < y,\ T(y) \in S', \text{ and if } x < z \leqslant y \text{ and } T(z) \in S' \text{ then } z = y\}.$$

[2] We assume some standard encoding for the automata. The *size* of an automaton is the *length* of its encoding.

By the conditions of the lemma, every path starting at x intersects X. Let W be the set of nodes between x and the nodes in X, that is,

$$W = \{z: x \leqslant z \leqslant y \text{ for some } y \in Y\}.$$

We leave it to the reader to verify that by König's infinity lemma, W is the desired subtree. Clearly, $|W| > 1$.

(2) $\Rightarrow$ (1). Let p be a path starting at λ and let $x \in p$. By assumption there exists a finite subtree $W \subset [n]^*$ rooted at x such that if $y \in W$ is a leaf, then $T(y) \in S'$. Since W is finite, there is a leaf y of W such that $y \in p$. But then $T(y) \in S'$. ∎

THEOREM 2.2. *The emptiness problem for Büchi automata is logspace complete for PTIME.*

Proof. In PTIME. The algorithm is analogous to the algorithm in [25] for Büchi automata on binary trees. Rather then repeat it, we give an informal description.

Since we deal with nondeterministic automata, we can assume that the alphabet Σ consists of a single letter a. Let us define a *good subtree embedded* in an automaton $A = (\Sigma, S, \rho, S_0, F)$ and *rooted* at a state s as a subtree $W \subset [n]^*$, $|W| > 1$, rooted at λ and a mapping $\phi: W \to S$, where $\phi(\lambda) = s$, for every internal node $x \in W$, we have $(\phi(x1),..., \phi(xn)) \in \rho(\phi(x), a)$, and for every leaf $x \in W$ we have $\phi(x) \in F$.

The algorithm for testing emptiness works by repeatedly eliminating states of the automaton that are not roots of good subtrees embedded in the automaton. Note that after a state is eliminated, both the transition function ρ, and the set F of designated states have to be updated accordingly. The algorithm stops when no more states can be eliminated. The automaton accepts some tree iff some initial state is not eliminated.

To test for the existence of good subtrees we use the algorithm in [30] for testing emptiness of automata on *finite* trees, which runs in time polynomial in the size of the automata. Clearly, our algorithm runs in time polynomial in the size of the automata (in fact, it can be made to run in quadratic time). It remains to prove that the initial state is not eliminated iff the automaton accepts some tree.

By Lemma 2.1, if a state is eliminated then it cannot participate in any accepting run. Thus, if all initial states are eliminated, then the automaton does not accept any tree. It remains to show that if some initial state is not eliminated, then the automaton accepts some tree. We prove this by constructing an accepting run.

Since the algorithm has eliminated all eliminable states, we know that for all states s there is a good subtree of positive depth embedded in the automaton, which is labeled by non-eliminable states whose frontier is labeled by states in F. We construct the run in stages, where at each stage we have a finite subtree of an accepting run. At stage 0 we have $\phi(\lambda) \in S_0$. At stage i, $0 < i < \omega$, we append to each leaf of the finite tree constructed in stage $i-1$ a finite tree of positive depth whose frontier is labeled by states in F. By Lemma 2.1, the constructing run is an accepting one.

Hard for PTIME. We show that the problem is hard for *PTIME* by reduction from the path system problem in [13]. An instance of the path system problem consists of a set U of nodes, a set X of initial nodes, a set Y of final nodes, and a ternary relation $R \subset U^3$. The problem is to determine if some node in Y is *accessible*, where the set of accessible nodes is the smallest set that contains X, and includes an element z whenever there are acessible nodes x and y and a triple $(x, y, z) \in R$. Given an instance of the problem, we construct a Büchi automaton $A = (\Sigma, S, \rho, S_0, F)$ as follows: $\Sigma = \{a\}$ consists of a single letter, $S = U$, $S_0 = Y$, $F = X$, and $(x, y) \in \rho(z, a)$ iff either $(x, y, z) \in R$ or $x = y = z \in X$. We leave it to the reader to verify that there is an accessible node in Y iff A accepts some tree. ∎

In the above proof of the correctness of the algorithm, we have *unwound* what remained of the transition function after the elimination of eliminable states into an accepting run. This is the automata-theoretic analogue of the unwinding process that converts pseudo-models to models.

In the process of reducing satisfiability to emptiness we shall find it useful to describe a set of trees by several automata rather than a single one. That is, we describe a set of trees as $T(A_0) \cap \cdots \cap T(A_{k-1})$. We now show that we can describe such a set of trees by a single Büchi automaton with only a polynomial increase in size. Let $|A|$ denote the number of states of the automaton A.

THEOREM 2.3. *Let $A_0, \ldots, A_{k-1}$ be Büchi automata. There is a Büchi automaton A with $k \prod_{i=0}^{k-1} |A_i|$ states such that $T(A) = T(A_0) \cap \cdots \cap T(A_{k-1})$.*

Proof. Rabin [25] has proved the claim for $k = 2$. We extend his proof to an arbitrary number of automata.

Let $A_i = (\Sigma, S^i, \rho^i, S_0^i, F^i)$. Define $A = (\Sigma, S, \rho, S_0, F)$ as follows:

$$S = S^0 \times \cdots \times S^{k-1} \times \{0, \ldots, k-1\},$$

$$S_0 = S_0^0 \times \cdots S_0^{k-1} \times \{0\},$$

$$F = F^0 \times S^1 \times \cdots S^{k-1} \times \{0\},$$

and

$$((s_1^0, \ldots, s_1^{k-1}, j), \ldots, (s_n^0, \ldots, s_n^{k-1}, j)) \in \rho((s^0, \ldots, s^{k-1}, i), \sigma)$$

iff $(s_1^i, \ldots, s_n^i) \in \rho(s^i, \sigma)$, for $0 \leqslant i \leqslant k-1$, and either $s^i \notin F^i$ and $i = j$ or $s^i \in F^i$ and $j = i + 1 \pmod k$.

We leave it to the reader to show that $T(A) = T(A_0) \cap \cdots \cap T(A_{k-1})$. ∎

COROLLARY 2.4. *For every fixed k, we can test in polynomial time, for given Büchi automata $A_0, \ldots, A_{k-1}$, whether $T(A_0) \cap \cdots \cap T(A_{k-1})$ is nonempty.*

2.5. *Subtree Automata*

We are now going to define a new class of automata on infinite n-ary trees, which we call *subtree automata.* Intuitively, a subtree automaton is an automaton that

verifies that if a node in the tree is labeled by a certain symbol, then this node is the root of a subtree accepted by some finite automaton on finite trees. Formally, a subtree automaton A is a tuple $(\Sigma, S, \rho, \xi, F)$, where

- Σ is the alphabet,
- S is the state set,
- $\rho: S \times \Sigma \to 2^{S^*}$ is the transition function,
- $\xi: \Sigma \to S$ is the labeling function, and
- $F \subset S$ is the nonempty set of accepting states.

A tree $T: [n]^* \to \Sigma$ is accepted by A if the following two conditions hold:

- $(\xi(T(x1)),..., \xi(T(xn))) \in \rho(\xi(T(x)), T(x))$ for every $x \in [n]^*$, and
- for every $x \in [n]^*$ there exists a finite subtree $W \subset [n]^*$ rooted at x and a mapping $\phi: W \to S$ such that $\phi(x) = \xi(T(x))$, if $y \in W$ is a leaf then $\phi(y) \in F$, and if $z \in W$ is an internal node then $(\phi(z1),..., \phi(zn)) \in \rho(\phi(z), T(z))$.

The first acceptance condition requires that the labeling ξ of the tree is compatible with the transition function of A. The second condition, requires that below each node x of the tree, there is a subtree accepted by A viewed as an automaton on finite trees with initial state $\xi(T(x))$. We call the first condition the *labeling condition* and we call the second condition the *subtree condition.*

By the results in [25], a subtree automaton, even without the labeling condition, can be converted to a Büchi automaton with an exponential increase in size. We show now that because of the labeling condition we can do this conversion with only a quadratic increase in size. Before proving this we need a technical lemma.

LEMMA 2.5. *Let $A = (\Sigma, S, \rho, \xi, F)$ be a subtree automaton. Then A accepts a tree $T: [n]^* \to \Sigma$ iff*

- $(\xi(T(x1)),..., \xi(T(xn))) \in \rho(\xi(T(x))$ *for every* $x \in [n]^*$, *and*
- *for every $x \in [n]^*$ there exists a finite subtree $W \subset [n]^*$, $|W| > 1$ rooted at x and a mapping $\phi: W \to S$ such that $\phi(x) = \xi(T(x))$, if $y \in W$ is a leaf then $\phi(y) \in F$, and if $z \in W$ is an internal node then $(\phi(z1),..., \phi(zn)) \in \rho(\phi(z), T(z))$.*

Proof. The only difference between the condition in the lemma and the standard condition of acceptance is the requirement that the subtrees consist of more than one node. Thus the "if" direction is trivial. For the "only if" direction assume that A accepts T. The labeling condition clearly holds, so it remains to show the existence of the "right" subtrees.

Let $x \in [n]^*$. Then there exists a finite subtree $W_x \subset [n]^*$ rooted at x and a mapping $\phi_x: W_x \to S$ such that $\phi_x(x) = \xi(T(x))$, if $y \in W_x$ is a leaf then $\phi_x(y) \in F$, and if $z \in W_x$ is an internal node then $(\phi_x(z1),..., \phi_x(zn)) \in \rho(\phi_x(z), T(z))$. If $|W_x| > 1$, we are done, so suppose that $W_x = \{x\}$. For every successor y of x there exists a finite subtree $W_y \subset [n]^*$ rooted at x and a mapping $\phi_y: W_y \to S$ such that $\phi_y(y) =$

$\xi(T(y))$, if $z \in W_y$ is a leaf then $\phi_y(z) \in F$, and if $z \in W_y$ is an internal node then $(\phi_y(z1),..., \phi_y(zn)) \in \rho(\phi_y(z), T(z))$. Define

$$W = \bigcup \{W_z : z \in \{x, x1,..., xn\}\},$$

and

$$\phi = \bigcup \{\phi_z : z \in \{x, x1,..., xn\}\}.$$

Clearly, $|W| > 1$, and it is easy to verify that ϕ satisfies the desired properties.

THEOREM 2.6. *Every subtree automaton with m states is equivalent to a Büchi automaton with m^2 states.*

Proof. Let $A = (S, \Sigma, \rho, \xi, F)$ be a subtree automaton. We now define two new transition functions $\rho_1, \rho_2 : S \times \Sigma \to 2^{S^n}$:

- $\rho_1(s, a) = \rho(s, a)$ if $s = \xi(a)$, and $\rho_1(s, a) = \varnothing$ otherwise.
- $\rho_2(s, a) = \rho(\xi(a), a)$ if $s \in F$, and $\rho_2(s, a) = \rho(s, a)$ otherwise.

These transition functions let us define two Büchi automata: $B_1 = (\Sigma, S, \rho_1, S, S)$ and $B_2 = (\Sigma, S, \rho_2, S, F)$. Basically, B_1 will take care of checking the labelling condition and B_2 of checking the subtree condition. Let us thus show that a tree $T : [n]^* \to S$ is accepted by A iff it is accepted by both B_1 and B_2.

Suppose first that T is accepted by B_1 and B_2. This means there are accepting runs $\psi_1, \psi_2 : [n]^* \to S$ of B_1 and B_2 (resp.) on T. We verify first the labeling property holds. Clearly, $\rho_1(\psi_1(x), T(x)) = \varnothing$ for every $x \in [n]^*$. Thus, $\psi_1(x) = \xi(T(x))$. Consequently, for every $x \in [n]^*$,

$$(\xi(T(x1),..., \xi(T(xn))) = (\psi_1(x1),..., \psi_1(xn)) \in \rho_1(\psi_1(x), T(x)) = \rho(\xi(T(x)), T(x)).$$

It remains to verify the subtree condition.

Let $x \in [n]^*$, by Lemma 2.1, there exists a finite subtree $W_x \subset [n]^*$ rooted at x such that if $y \in W_x$ is a leaf then $\psi_2(y) \in F$. Let now z be a leaf of W_x. By Lemma 2.1, there exists a finite subtree $W_z \subset [n]^*$ rooted at z such that if $y \in W_z$ is a leaf then $\psi_2(y) \in F$. Assume without loss of generality that for all internal nodes $y \in W_z - \{z\}$ we have that $\psi_2(y) \notin F$. Let

$$W = W_x \cup \{y \in W_z : z \text{ is a leaf of } W_x\}.$$

We claim that W satisfies the subtree condition for the node x. Indeed, a suitable mapping $\phi : W \to S$ can be defined in the following way: $\phi(y) = \xi(T(y))$ for $y \in W_x$, and $\phi(y) = \psi_2(y)$ for $y \in W_z - \{z\}$, where z is a leaf of W_x.

Clearly $\phi(y) \in F$ for any leaf y of W. We show that for all internal nodes $y \in W$ we have $(\phi(y1),..., \phi(yn)) \in \rho(\phi(y), T(y))$.

Let y be an internal node of W_x. Then $\phi(y) = \xi(T(y))$, and by the labeling condition $(\phi(y1),..., \phi(yn)) \in \rho(\phi(y), T(y))$.

Let z be a leaf of W_x. Then $\psi_2(z) \in F$. Consequently,

$$(\phi(z1),..., \phi(zn)) = (\psi_2(z1),..., \psi_2(zn)) \in \rho_2(\psi_2(z), T(z))$$
$$= \rho(\xi(T(z)), T(z)) = \rho(\phi(z), T(z)).$$

Let y be an internal node, different form z, of W_z. Then $\phi(y) = \psi_2(y) \notin F$. Consequently,

$$(\phi(z1),..., \phi(zn)) = (\psi_2(z1),..., \psi_2(zn)) \in \rho_2(\psi_2(z), T(z)) = \rho(\phi(y), T(y)).$$

We now have to show that if T is accepted by A then it is accepted by both B_1 and B_2. We first define a mapping $\psi_1 : [n]^* \to S$ by $\psi_1(x) = \xi(T(x))$. By the labeling condition, ψ_1 is an accepting run of B_1 on T.

We now define a mapping $\psi_2 : [n]^* \to S$ by defining it on a sequence $W_1, W_2,...,$ of subtrees rooted in λ in such that a way that if x is a leaf of one of these subtrees then $\psi_2(x) \in F$. The first subtree is $W_1 = \{\lambda\}$, and we define $\psi_2(\lambda) = s$, where s is an arbitrary member of F. Suppose that ψ_2 is defined on a subtree W_n and let $x \in W_n$ be a leaf. By induction, $\psi_2(x) \in F$. By Lemma 2.5, there exists a finite subtree $W_x \subset [n]^*$, $|W_x| > 1$, rooted at x and a mapping ϕ: $W_x \to S$ such that $\phi(x) = \xi(T(x))$, $\phi(y) \in F$ for each $y \in W_x$ that is a leaf, and $(\phi(z1),..., \phi(zn)) \in \rho(\phi(z), T(z))$ for each $z \in W$ that is an internal node. We can assume that if y is an internal node of W_x different from x then $\phi(y) \notin F$. For a node $y \in W_x - \{x\}$, define $\psi_2(y) = \phi(y)$. We show now that for every internal node $y \in W_x$ we have that $(\psi_2(y1),..., \psi_2(yn)) \in \rho_2(\psi_2(y), T(y))$. Indeed, if $y = x$, then $\phi(y) = \xi(T(y))$ and $\psi_2(y) \in F$. Consequently,

$$(\psi_2(y1),..., \psi_2(yn)) = (\phi(y1),..., \phi(yn)) \in \rho(\xi(T(y)), T(y)) = \rho_2(\psi_2(y), T(y)).$$

If, on the other hand, $y \neq x$, then $\psi_2(y) = \phi(y) \notin F$. Consequently,

$$(\psi_2(y1),..., \psi_2(yn)) = (\phi(y1),..., \phi(yn)) \in \rho(\phi(y), T(y)) = \rho_2(\psi_2(y), T(y).$$

Let

$$W_{n+1} = W_n \cup \{y \in W_x : x \text{ is a leaf of } W_n\}.$$

Clearly, $\bigcup_{i=1}^{\infty} W_i = [n]^*$, so ψ_2 is a run of B_2 on T. Furthermore, for every $x \in [n]^*$ there exists a finite subtree $W \subset [n]^*$, $|W| > 1$, rooted at x such that if $y \in W$ is a leaf then $\psi_2(y) \in F$. Thus by Lemma 2.1, for every path p starting at λ, $\inf(\psi_2, p) \cap F \neq \emptyset$. Thus, ψ_2 is an accepting run.

We have shown that T is accepted by both B_1 and B_2. Finally, by Theorem 2.3, we can construct an automaton B such that a tree T is accepted by B iff it is accepted by both B_1 and B_2. It follows that the subtree automaton A is equivalent to the Büchi automaton B. ∎

Subtree automata are more adequate than Büchi automata for the purpose of reducing satisfiability to emptiness. Nevertheless, to facilitate our task in the rest of

the paper, we now specialize the notion of subtree automata even further. The trees that we shall deal with are going to be labeled by sets of formulas, and the states of the automata that will accept these trees are also going to be sets of formulas. Thus, we consider automata where the alphabet and the set of states are the same set and have the structure of a power-set. We will call these automata *set-subtree automata.*

Formally, a set-subtree automaton A is a pair (Ψ, ρ), where

- Ψ is a finite set of basic symbols (in fact these symbols will be just formulas of the logic). The power set 2^{Ψ} serves both as the alphabet Σ and as the state set S. The empty set serves as a single accepting state. We will denote elements of 2^{Ψ} by $\mathbf{a}, \mathbf{b},\ldots$, when viewed as letters from the alphabet Σ, and by $\mathbf{s}, \mathbf{s}_1,\ldots$, when viewed as elements of the state set S.
- $\rho: \mathbf{S} \times \Sigma \to 2^{\mathbf{S}^n}$ is a transition function such that
  (1) $\rho(\mathbf{s}, \mathbf{a}) \neq \varnothing$ iff $\mathbf{s} \subset \mathbf{a}$,
  (2) $(\varnothing,\ldots, \varnothing) \in \rho(\varnothing, \mathbf{a})$,
  (3) if $\mathbf{s} \subset \mathbf{s}'$, $\mathbf{s}_1 \subset \mathbf{s}'_1,\ldots, \mathbf{s}_n \subset \mathbf{s}'_n$, and $(\mathbf{s}_1,\ldots, \mathbf{s}_n) \in \rho(\mathbf{s}', \mathbf{a})$, then $(\mathbf{s}'_1,\ldots, \mathbf{s}'_n) \in \rho(\mathbf{s}, \mathbf{a})$.
  (4) if $(\mathbf{s}_{11},\ldots, \mathbf{s}_{1n}) \in \rho(\mathbf{s}_1, \mathbf{a})$ and $(\mathbf{s}_{21},\ldots, \mathbf{s}_{2n}) \in \rho(\mathbf{s}_2, \mathbf{a})$, then $(\mathbf{s}_{11} \cup \mathbf{s}_{21},\ldots, \mathbf{s}_{1n} \cup \mathbf{s}_{2n}) \in \rho(\mathbf{s}_1 \cup \mathbf{s}_2, \mathbf{a})$.

A tree T: $[n]^* \to \Sigma$ is accepted by A if for every $x \in [n]^*$ and every $f \in T(x)$ there exists a finite subtree $W \subset [n]^*$ rooted at x and a mapping ϕ: $W \to \mathbf{S}$ such that

- $\phi(x) = \{f\}$,
- if $y \in W$ is a leaf then $\phi(y) = \varnothing$, and
- if $z \in W$ is an internal node then $(\phi(z1),\ldots, \phi(zn)) \in \rho(\phi(z), T(z))$.

The acceptance condition requires that for each node x of the tree and for each each formula f in $T(x)$, the "right" formulas appear in the labels of the nodes under x. Intuitively, the transition of the automaton are meant to capture the fact that if a certain formula appears in the label of a node x, then certain formulas must appear in the labels of the successors of x. The four conditions imposed on the transition relation ρ can be explained as follows:

(1) The formulas in the state of the automaton are formulas that the automaton is trying to verify. A minimal requirement is that these formulas appear in the label of the scanned node of the tree. As we will see, this condition is related to the labeling condition defined for subtree automata.

(2–3) These are what we call *monotonicity* conditions. A transition of the automaton is a minimum requirement on the formulas that holds of $x1,\ldots, xn$ given the formulas that the automaton is trying to verify at x. Clearly if there is nothing to verify at x, then nothing is required at $x1,\ldots, xn$ (condition 2)). Also, the transition is still legal if we try to verify fewer formulas at x or more formulas at $x1,\ldots, xn$.

(4) This is an *additivity* condition. It says that there is no interaction between different formulas that the automaton is trying to verify at node x. Thus the union of two transitions is a legal transition.

The acceptance condition requires that for each node of the tree, if we start the automaton in each of the singleton sets corresponding to the members of the label of that node, it accepts a finite subtree. We will now prove that, given conditions (2), (3), and (4), it is equivalent to require that the automaton accepts when started in the state identical to the label of the node.

LEMMA 2.7. *Let $A=(\Psi, \rho)$ be a set-subtree automaton, let $T: [n]^* \to 2^\Psi$ be a tree, and let $x \in [n]^*$. The following are equivalent:*

(1) *There exist a finite subtree $W \subset [n]^*$ rooted at x and a mapping $\phi: W \to \mathbf{S}$ such that $\phi(x)=T(x)$, if $y \in W$ is a leaf then $\phi(y)=\varnothing$, and if $z \in W$ is an internal node then $(\phi(z1),\ldots,\phi(zn)) \in \rho(\phi(z), T(z))$.*

(2) *For every $f \in T(x)$ there exists a finite subtree $W_f \subset [n]^*$ rooted at x and a mapping $\phi_f: W_f \to \mathbf{S}$ such that $\phi_f(x)=\{f\}$, if $y \in W_f$ is a leaf then $\phi_f(y)=\varnothing$, and if $z \in W_f$ is an internal node then $(\phi_f(z1),\ldots,\phi_f(zn)) \in \rho(\phi_f(z), T(z))$.*

Proof. (1) $\Rightarrow$ (2). Let $f \in T(x)$. We take $W_f = W$, and define ϕ_f as follows: $\phi_f(x)=\{f\}$ and $\phi_f(y)=\phi(y)$ for $y \neq x$. We only have to show that $(\phi_f(x1),\ldots, \phi_f(xn)) \in \rho(\phi_f(x), T(x))$. But this follows, by the monotonicity condition (3) in the definition of set-subtree automata, given that $\phi_f(x) \subset \phi(x)$.

(2) $\Rightarrow$ (1). If $T(x)=\varnothing$ take $W=\{x\}$, otherwise take $W=\bigcup_{f \in T(x)} W_f$. It is easy to verify that W is a subtree rooted at x. Also, if y is a leaf of W, then for each $f \in T(x)$, either y is a leaf of W_f or $y \notin W_f$. For every $f \in T(x)$ extend ϕ_f to W by defining $\phi_f(y)=\varnothing$ for $y \in W - W_f$. If $T(x)=\varnothing$ we define $\phi(x)=\varnothing$, otherwise we define $\phi(y)=\bigcup_{f \in T(x)} \phi_f(y)$ for each $y \in W$. By the above observation concerning leaves, if y is a leaf of W then $\phi(y)=\varnothing$. Furthermore, by condition (4) in the definition of the set-subtree automata, if $z \in W$ is an internal node then $(\phi(z1),\ldots,\phi(zn)) \in \rho(\phi(z), T(z))$. ∎

We can now prove that set-subtree automata can be converted to subtree automata without any increase in size. Thus, by Theorem 2.6, a set-subtree automaton can be converted to an equivalent Büchi automaton with only a quadratic increase in size.

THEOREM 2.8. *The set-subtree automaton $A=(\Psi, \rho)$ is equivalent to the subtree automaton $A'=(2^\Psi, 2^\Psi, \rho, \xi, \{\varnothing\})$, where ξ is the identity function.*

Proof. By Lemma 2.7, it is immediate that if a tree T is accepted by the automaton A', then it is also accepted by A. Also by Lemma 2.7, if the tree T is accepted by the set-subtree automaton A, the subtree condition of the subtree automaton A' is satisfied. It remains to show that ξ satisfied the labeling condition. In other words, since ξ is the identity mapping, we have to show that for every $x \in [n]^*$, $(T(x1),\ldots, T(xn)) \in \rho(T(x), T(x))$. Since A accepts T, there exists a finite subtree $W \subset [n]^*$ rooted at x and a mapping $\phi: W \to \mathbf{S}$ such that $\phi(x)=T(x)$, if $y \in W$ is a leaf then $\phi(y)=\varnothing$, and if $z \in W$ is an internal node then

$(\phi(z1),..., \phi(zn)) \in \rho(\phi(z), T(z))$. If $W = \{x\}$, then $T(x) = \varnothing$, and by the monotonicity conditions we have $(T(x1),..., T(xn)) \in \rho(T(x), T(x))$. Otherwise, $\{x1,..., xn\} \subset W$, so $(\phi(x1),..., \phi(xn)) \in \rho(T(x), T(x))$. Consider now each xi, $i \in [n]$. If $\phi(xi) = \varnothing$, then clearly $\phi(xi) \subset T(xi)$. Otherwise, $\rho(\phi(xi), T(xi)) \neq \varnothing$, so, by condition (1) in the definition of set-subtree automata, $\phi(xi) \subset T(xi)$. Thus by the monotonicity condition $(T(x1),..., T(xn)) \in \rho(T(x), T(x))$. ∎

3. Deterministic Propositional Dynamic Logic

We assume familiarity with *dynamic logic* [19] and with *propositional dynamic logic* (*PDL*) [6].

Deterministic propositional dynamic logic (*DPDL*) is a propositional dynamic logic with deterministic atomic programs. It was studied in [1], where an exponential decision procedure was given. The proof of the decision procedure given there is quite complicated. Here, we show how it can be substantially simplified using the automata theoretic result established in the previous section. We will consider a variant of *DPDL*, in which programs are described by finite automata rather than by regular expressions (c.f. [12, 22]). This variant is called *ADPDL*. *ADPDL* is more succinct than *DPDL* and also has the advantage of fitting nicely with our automata theoretic techniques. As the translation from regular expressions to automata is linear, our results for *ADPDL* apply easily to *DPDL*.

Formulas of *ADPDL* are built from a set of atomic propositions *Prop* and a set *Prog* of atomic programs. The sets of *formulas*, *tests*, and *programs* are defined inductively as follows:

- every proposition $p \in Prop$ is a formula.
- if f_1 and f_2 are formulas, then $\neg f_1$ and $f_1 \wedge f_2$ are formulas.
- If f is a formula, then $f?$ is a test.
- if α is a program and f is a formula, then $\langle\alpha\rangle f$ is a formula
- If α is a sequential automaton over an alphabet Σ, where Σ is a finite set of atomic programs and tests, then α is a program.

ADPDL formulas are interpreted over structures $M = (W, R, \Pi)$, where W is a set of states, R: $Prog \to 2^{W \times W}$ is a deterministic transition relation (for each state u and atomic program a there is at most one pair $(u, u') \in R(a)$), and Π: $W \to 2^{Prop}$ assigns truth values to the proposition in *Prop* for each state in W. We now extend R to all programs and define satisfaction of a formula f in a state u of a structure M, denoted $M, u \models f$, inductively:

- $R(f?) = \{(u, u)$: $M, u \models f\}$.
- $R(\alpha) = \{(u, u')$: there exists a word $w = w_1 \cdots w_n$ accepted by α and states $u_0, u_1,..., u_n$ of W such that $u = u_0$, $u' = u_n$ and for all $1 \leqslant i \leqslant n$ we have $(u_{i-1}, u_i) \in R(w_i)\}$.
- For a proposition $p \in Prop$, $M, u \models p$ iff $p \in \Pi(u)$.
- $M, u \models f_1 \wedge f_2$ iff $M, u \models f_1$ and $M, u \models f_2$.

- $M, u \models \neg f_1$ iff not $M, u \models f_1$.
- $M, u \models \langle \alpha \rangle f$ iff there exists a state u' such that $(u, u') \in R(\alpha)$ and $M, u' \models f$.

Note that only atomic programs are required to be deterministic, while non-atomic programs can be nondeterministic.

A formula f is *satisfiable* if there is a structure M and a state u in the structure such that $M, u \models f$. The *satisfiability problem* is to determine, given a formula f, whether f is satisfiable. Before giving the decision procedure for satsifiability of *ADPDL* formulas, we need to define a notion of closure of *ADPDL* formulas similar to the closure defined for *PDL* in [6]. From now on we identify a formula g with $\neg\neg g$. The closure of a formula f, denoted $\mathrm{cl}(f)$, is defined as follows:

- $f \in \mathrm{cl}(f)$
- If $g_1 \wedge g_2 \in \mathrm{cl}(f)$ then $g_1, g_2 \in \mathrm{cl}(f)$.
- If $\neg g \in \mathrm{cl}(f)$ then $g \in \mathrm{cl}(f)$.
- If $g \in \mathrm{cl}(f)$ then $\neg g \in \mathrm{cl}(f)$.
- If $\langle \alpha \rangle g \in \mathrm{cl}(f)$ then $g \in \mathrm{cl}(f)$.
- If $\langle \alpha \rangle g \in \mathrm{cl}(f)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then $g' \in \mathrm{cl}(f)$ for all $g'? \in \Sigma$.
- If $\langle \alpha \rangle g \in \mathrm{cl}(f)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then $\langle \alpha_s \rangle g \in \mathrm{cl}(f)$ for all $s \in S$.[3]

It is not hard to verify that the size of $\mathrm{cl}(f)$ is linear in the length of f.

For our techniques to be usable, we first have to prove that *ADPDL* has the *tree model property*. A *tree structure* for a formula f is a structure $M = (W, R, \Pi)$ such that:

(1) $W \subseteq [n]^*$, where n is linear in the length of f and $W \neq \emptyset$.

(2) $xi \in W$ only if $x \in W$.

(3) $(x, y) \in R(a)$ for an atomic program a only if x is the predecessor or the successor of y and $(x, y) \notin R(b)$ for any other atomic program b.

If in addition we have that $(x, y) \in R(a)$ for an atomic program a only if x is the predecessor of y, then M is a *one-way* tree structure. The reason we consider two-way tree structures is that they will be necessary when we extend the logic with the *converse* construct. A tree structure $M = (W, R, \Pi)$ is a *tree model* for f if $M, \lambda \models f$ (note that since $W \neq \emptyset$, $\lambda \in W$).

We now show that *ADPDL* has the one-way tree model property, i.e., if an *ADPDL* formula f is satisfiable, then it has a one-way tree model

PROPOSITION 3.1. *Let f be a satisfiable ADPDL formula with atomic programs $a_1, \ldots, a_n$. Then f has an n-ary one-way tree model.*

Proof. Suppose that $M, u \models f$, for some structure $M = (W, R, \Pi)$ and some state $u \in W$. To show that f has a one-way tree model, we first define a partial mapping ϕ: $[n]^* \to W$ by induction on the length of the words in $[n]^*$. To start, we take $\phi(\lambda) = u$. Suppose now that ϕ is known for every $x \in [n]^k$, and let $xi \in [n]^{k+1}$. If

[3] α_s is defined in Section 2.1.

$\phi(x)$ is undefined, then so is $\phi(xi)$. If $\phi(x)=s\in W$ but s has no a_i-successor in M (i.e., there is no state $t\in W$ such that $(s,t)\in R(a_i)$), then again $\phi(xi)$ is left undefined. If $\phi(x)=s$ and t is the a_i-successor of s (if there is such a successor then it must be unique), then $\phi(xi)=t$.

We now define a structure $M'=(W', R', \Pi')$ as follows. $W'=\{x: \phi(x)$ is defined$\}$, $R(a_i)=\{(x, xi): xi\in W'\}$, and $\Pi'(x)=\Pi(\phi(x))$. We claim now that M' is a one-way tree structure and that if $x\in W'$ and g is any *ADPDL* formula with atomic programs among $a_1,\ldots, a_n$, then $M', x\models g$ iff $M, \phi(x)\models g$. The proof is straightforward and is left to the reader. In particular we have that $M', \lambda\models f$. ∎

Intuitively, what we have done is to unravel M into a tree with u as its root. Furthermore, as all atomic programs are deterministic, the branching factor of the tree is at most the number of atomic programs that occur in f. Note that the tree model can have infinitely many states, even if the original model was finite.

To establish a decision procedure for *ADPDL*, we reduce the satisfiability problem to the emptiness problem for Büchi automata (via set-subtree automata). To this end we associate an infinite n-ary tree over $2^{\mathrm{cl}(f)\cup\{\perp\}}$ with the tree model $M'=(W', R', \Pi')$ constructed above in a natural way: every node in W' is labeled by the formulas in $\mathrm{cl}(f)$ that are satisfied at that node, and the other nodes are labeled by the special symbol $\perp$. Trees that correspond to tree models satisfy some special properties.

A *Hintikka tree* for an *ADPDL* formula f with atomic programs $a_1,\ldots, a_n$ is an n-ary tree T: $[n]^*\to 2^{\mathrm{cl}(f)\cup\{\perp\}}$ that satisfies the following conditions:

(1) $f\in T(\lambda)$,

and, for all elements x of $[n]^*$:

(2) either $T(x)=\{\perp\}$ or $\perp\notin T(x)$ and $g\in T(x)$ iff $\neg g\notin T(x)$,

(3) $g_1\wedge g_2\in T(x)$ iff $g_1\in T(x)$ and $g_2\in T(x)$,

(4) if $\langle\alpha\rangle g\in T(x)$, where $\alpha=(\Sigma, S, \rho, s_0, F)$, then there exists a word $w=w_1,\ldots, w_k$ over Σ, $k\geqslant 0$, states $s_1,\ldots, s_k$ of S, and nodes $u_0,\ldots, u_k$ of $[n]^*$ such that:

(a) $u_0=x$,

(b) $g\in T(u_k)$ and $s_k\in F$,

and, for all $1\leqslant i\leqslant k$,

(c) $s_i\in\rho(s_{i-1}, w_i)$,

(d) if w_i is f?, then $f\in T(u_{i-1})$ and $u_i=u_{i-1}$, and

(e) if w_i is a_j, then $u_i=u_{i-1}j$.

(Note that if $k=0$ in condition (4), then $s_0\in F$ and $g\in T(x)$.)

(5) if $\neg\langle\alpha\rangle g\in T(x)$, where $\alpha=(\Sigma, S, \rho, s_0, F)$, then $s_0\in F$ entails $\neg g\in T(x)$, and for all $s\in S$ and $w\in\Sigma$ such that $s\in\rho(s_0, w)$:

(a) if w is g'?, then either $\neg g'\in T(x)$ or $\neg\langle\alpha_s\rangle g\in T(x)$.

(b) if w is a_j then $\neg\langle\alpha_s\rangle g\in T(xj)$ or $T(xj)=\{\perp\}$.

PROPOSITION 3.2. *An ADPDL formula f has a one-way tree model iff it has a Hintikka tree.*

Proof. Only if. Let f be an *ADPDL* formula with atomic programs $a_1,\ldots, a_n$, and let $M=(W, R, \Pi)$ be a one-way tree model of f. We define a Hintikka tree T for f as follows: for an element $x\in[n]^*-W$, $T(x)=\{\bot\}$ and for an element $x\in W$, $T(x)=\{g\in \mathrm{cl}(f)\colon M, x\models g\}$. We now have to show that T is indeed a Hintikka tree. By definition of a tree model $M, \lambda\models f$; this implies condition (1). That conditions (2), (3), (4), and (5) hold follows immediately from the semantic definition of *ADPDL*.

If. Let f be an *ADPDL* formula with atomic programs $a_1,\ldots, a_n$, and let T be a Hintikka tree for f. We construct a one-way tree model for f as follows. The structure is $M=(W, R, \Pi)$, where $W=\{x\in[n]^*\colon T(x)\neq\{\bot\}\}$, $R(a_i)=\{(x, xi)\colon i\in[n]$ and $xi\in W\}$, and for all $x\in W$, $\Pi(x)=\{p\in Prop\colon p\in T(x)\}$. It now remains to show that $M, \lambda\models f$. For this, we show by induction that for all $g\in\mathrm{cl}(f)$ and $x\in[n]^*$, we have that $g\in T(x)$ iff $M, x\models g$. For the base case ($g\in Prop$), this is immediate by construction. The inductive step for formulas of the form $g_1\wedge g_2$, $\neg g_1$ and $\langle\alpha\rangle g_1$ follows directly from the Hintikka conditions (2), (3) and (4), (5) respectively. ∎

The next step is to build a Büchi automaton on n-ary trees over the alphabet $2^{\mathrm{cl}(f)\cup\{\bot\}}$ that accepts precisely the Hintikka trees for f. Rather than do that, we build two automata, A_L and $A_{\langle\rangle}$, such that $T(A_L)\cap T(A_{\langle\rangle})$ is the set of Hintikka trees for f. The first automaton A_L, called the *local* automaton, checks the tree locally, i.e., it checks Hintikka conditions (1)–(3) and (5). This automaton is a Büchi automaton. The second automaton $A_{\langle\rangle}$, called the $\langle\ \rangle$-automaton, is a set-subtree automaton that checks condition (4). This automaton ensures that for all *eventualities* (i.e., formulas of the form $\langle\alpha\rangle f$, for some program α) there is some finite word for which condition (4) is satisfied. Finally, we convert the $\langle\ \rangle$-automaton to a Büchi automaton and combine it with the local automaton.

The Local Automaton

The local automaton is $A_L=(2^{\mathrm{cl}(f)\cup\{\bot\}}, 2^{\mathrm{cl}(f)\cup\{\bot\}}, \rho_L, N_f, 2^{\mathrm{cl}(f)\cup\{\bot\}})$. The state set is the collection of all sets of formulas in $\mathrm{cl}(f)\cup\{\bot\}$. For the transition relation ρ_L, we have that $(\mathbf{s}_1,\ldots, \mathbf{s}_n)\in\rho_L(\mathbf{s}, \mathbf{a})$ iff $\mathbf{a}=\mathbf{s}$ and:

- either $\mathbf{s}=\{\bot\}$ or $\bot\notin\mathbf{s}$ and $g\in\mathbf{s}$ iff $\neg g\notin\mathbf{s}$,
- $g_1\wedge g_2\in\mathbf{s}$ iff $g_1\in\mathbf{s}$ and $g_2\in\mathbf{s}$,
- if $\neg\langle\alpha\rangle g\in\mathbf{s}$, where $\alpha=(\Sigma, S, \rho, s_0, F)$, then $s_0\in F$ entails $\neg g\in\mathbf{s}$, and for all $s\in S$ and $w\in\Sigma$ such that $s\in\rho(s_0, w)$:
  - if w is $g'?$, then either $\neg g'\in\mathbf{s}$ or $\neg\langle\alpha_s\rangle g\in\mathbf{s}$, and
  - if w is a_j then $\neg\langle\alpha_s\rangle g\in\mathbf{s}_j$ or $\mathbf{s}_j=\{\bot\}$.

The set of starting states N_f consists of all sets $\mathbf{s}$ such that $f\in\mathbf{s}$. Clearly, A_L accepts precisely the trees that satisfy Hintikka conditions (1)–(3) and (5).

The $\langle\ \rangle$-Automaton

Before describing the construction of $A_{\langle\ \rangle}$, we express Hintikka condition (4) in a form that will be easier to handle.

LEMMA 3.3. *Let f be an ADPDL formula with atomic programs $a_1,\ldots, a_n$, and let T: $[n]^* \to 2^{\mathrm{cl}(f)\cup\{\perp\}}$ be an n-ary tree. Then satisfies Hintikka condition (4) iff for all $x \in [n]^*$ we have that if $\langle\alpha\rangle\, g \in T(x)$, where $\alpha = (\Sigma, S, \rho, s, F)$, then there are nodes $u_0,\ldots, u_k$ of $[n]^*$, states $s_0, t_0,\ldots, s_k, t_k$ of S, and atomic programs $a_{j1},\ldots, a_{jk}$ such that*

(a) $u_0 = x$, $s_0 = s$, $t_k \in F$, $u_i = u_{i-1} j_i$, *and* $s_i \in \rho(t_{i-1}, a_{ji})$, *for* $1 \leqslant i \leqslant k$,

(b) *for* $0 \leqslant i \leqslant k$ *there exists a word* $w = g_1? \cdots g_m?$, $m \geqslant 0$, such that $\{g_1,\ldots, g_m\} \subset T(u_i)$ *and* $t_i \in \rho(s_i, w)$,

(c) $g \in T(u_k)$.

Proof. Left to the reader. ∎

We can now describe the $\langle\ \rangle$-automaton. It is a set-subtree automaton $A_{\langle\ \rangle} = (\mathrm{cl}(f) \cup \{\perp\},\ \rho_{\langle\ \rangle})$. For the transition relation $\rho_{\langle\ \rangle}$, we have that $(\mathbf{s}_1,\ldots, \mathbf{s}_n) \in \rho_{\langle\ \rangle}(\mathbf{s}, \mathbf{a})$ iff:

- $\mathbf{s} \subset \mathbf{a}$, and
- If $\langle\alpha\rangle\, g \in \mathbf{s}$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then there is a word $w = g_1? \cdots g_m?$, $m \geqslant 0$, and a state $s \in S$ such that $s \in \rho(s_0, w)$, $\{g_1,\ldots, g_m\} \subset \mathbf{a}$, and either $s \in F$ and $g \in \mathbf{a}$ or there is an atomic program a_j and a state $s' \in \rho(s, a_j)$ such that $\langle\alpha_{s'}\rangle\, g \in \mathbf{s}_j$.

It is immediate to check that conditions (1)–(4) of the definition of set-subtree automata are satisfied for $A_{\langle\ \rangle}$. Furthermore, by Lemma 3.3, $A_{\langle\ \rangle}$ accepts precisely the trees that satisfy Hintikka condition (4). Thus we have

PROPOSITION 3.4. *Let f be an ADPDL formula with atomic programs $a_1,\ldots, a_n$, and let T: $[n]^* \to 2^{\mathrm{cl}(f)\cup\{\perp\}}$ be an n-ary tree. Then T is a Hintikka tree for f iff $T \in T(A_L) \cap T(A_{\langle\ \rangle})$.*

At this point, we can give an algorithm and complexity bounds for the satisfiability problem for *ADPDL*. Given a formula f, we construct the automata A_L and $A_{\langle\ \rangle}$. By Propositions 3.1, 3.2, and 3.4, f is satisfiable iff $T(A_L) \cap T(A_{\langle\ \rangle}) \neq \varnothing$. The size of these automata is exponential in the length of f. By the results in Section 2, we can construct a Büchi automaton A, whose size is exponential in the length of f, such that $T(A) = T(A_L) \cap T(A_{\langle\ \rangle})$. Note that A can be constructed in time exponential in the length of f. Since we can check emptiness of A in time polynomial in the size of A, we have proven

THEOREM 3.5. *The satisfiability problem for ADPDL can be solved in exponential time.*

The size of the automata we construct in the process of testing satisfiability is $O(c^{n^2})$ for some constant c, where n is the length of the given formula. Nevertheless, there is a way to implement the algorithm to run in time $O(c^n)$.

Parikh has shown [17] that the satisfiability problem for *PDL* is logspace reducible to the satisfiability problem for *DPDL*, since a nondeterministic atomic program a can be encoded as the composite program $b; c^*$, where b and c are deterministic atomic programs. Thus we have also reestablished an exponential upper bound for the satisfiability problem for *PDL*. Furthermore, Fisher and Ladner have proven an exponential lower bound for *PDL* [6]. Since Parikh's reduction is a logspace reduction, the same lower bound holds for *ADPDL*.

4. Deterministic Propositional Dynamic Logic with Looping

In [11], the construct *loop* is added to *PDL*. Intuitively, the formula $loop(\alpha)$ holds in a state if there is an infinite computation of α from that state. The *loop* construct should be distinguished from the *repeat* construct, denoted Δ in [28, 29]. The formula $repeat(\alpha)$ holds in a state if α can be repeated infinitely often from that state. In [11], it is shown that the *repeat* construct is strictly more expressive than the loop construct when incorporated into *PDL*. *Loop-PDL* was shown in [23] to have an exponential decision procedure, while the best known upper bound for *repeat-PDL* is nondeterministic exponential [31]. Here, we consider *DPDL*, or rather, as in the previous section *ADPDL*, augmented with the *loop* construct. We will show that our automata-theoretic techniques enable us to very easily obtain an exponential decision procedure for *loop-ADPDL*.

Syntactically, the definition of *loop-ADPDL* is identical to that of *ADPDL* except for the addition of the following clause:

- If $\alpha = (\Sigma, S, \rho, s, F)$ is a program, then $loop(\alpha)$ is a formula.

Semantically, *loop-ADPDL* is also defined exacty as *ADPDL*, with the addition of

- $M, u \models loop(\alpha)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, iff there exists an infinite word $w = w_1 w_2 \cdots$, over Σ, an infinite sequence $s_0, s_1, \ldots$, of states of S, and an infinite sequence $u_0, u_1, \ldots$, of nodes of W such that:
  - $u_0 = u$, $s_0 = s$, and
  - for all $i \geqslant 1$, $s_i \in \rho(s_{i-1}, w_i)$ and $(u_{i-1}, u_i) \in R(w_i)$.

Again, the closure of a *loop-ADPDL* formula is defined exactly as for *ADPDL* with the addition of the two following clauses:

- If $loop(\alpha) \in \mathrm{cl}(f)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then $g' \in \mathrm{cl}(f)$ for all $g'? \in \Sigma$.
- If $loop(\alpha) \in \mathrm{cl}(f)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then $loop(\alpha_s) \in \mathrm{cl}(f)$ for all $s \in S$.

The proof that *loop-ADPDL* has the one-way tree model property is almost identical to the proof of Proposition 3.1.

Proposition 4.1. *Let f be a satisfiable loop-ADPDL formula with atomic programs $a_1, \ldots, a_n$. Then f has an n-ary one way tree model.*

Before defining Hintikka trees for *loop-ADPDL*, we will prove two results about loop formulas

PROPOSITION 4.2. *Let $M=(W, R, \Pi)$ be a tree structure, let $x \in W$, and let $\alpha=(\Sigma, S, \rho, s_0, F)$ be a program. Then $M, x \models loop(\alpha)$ if and only if there exists an $s \in S$ and a $w \in \Sigma$ such that $s \in \rho(s_0, w)$ and*

- *if w is $g?$, then $M, x \models g$ and $M, x \models loop(\alpha_s)$.*
- *if w is a_j then $M, xj \models loop(\alpha_s)$.*

Proof. Left to the reader. ∎

PROPOSITION 4.3. *Let $M=(W, R, \Pi)$ be a tree structure, let $x \in W$, and let $\alpha=(\Sigma, S, \rho, s_0, F)$ be a program. Then $M, x \models \neg loop(\alpha)$ if an only if there is a finite subset $W' \subset W$ with $x \in W'$ and a mapping $\phi: W' \to 2^{\{\neg loop(\alpha_p): p \in S\}}$, such that $\neg loop(\alpha) \in \phi(x)$, and if $y \in W'$ and $\neg loop(\alpha_p) \in \phi(y)$, then*

- *there is no word $w=g_1?\, g_2? \cdots g_m?$, $m \geqslant 1$, such that $M, y \models g_i$, $1 \leqslant i \leqslant m$, and $p \in \rho(p, w)$,*
- *for all $s \in S$ and $w \in \Sigma$ such that $s \in \rho(p, w)$:*
  - (a) *if w is $g?$, then either $M, y \models \neg g$ or $\neg loop(\alpha_s) \in \phi(y)$,*
  - (b) *if w is a_j then either $yj \in W'$ and $\neg loop(\alpha_s) \in \phi(yj)$ or $yj \notin W$.*

Proof. If. We claim that $M, y \models \neg loop(\alpha_p)$ for all $y \in W'$ and $\neg loop(\alpha_p) \in \phi(y)$. In particular $M, x \models \neg loop(\alpha)$.

We now prove the claim. Let $y \in W'$ be such that it has no successors in W' and let $\neg loop(\alpha_p) \in \phi(y)$. Suppose that $M, y \models loop(\alpha_p)$. Then there exists an infinite word $w=w_1 w_2 \cdots$ over Σ, an infinite sequence $s_0, s_1, \ldots$ of states of S, and an infinite sequence $y_0, y_1, \ldots$ of nodes of W such that:

- $y_0=y$, $s_0=p$ and
- for all $i \geqslant 1$, $s_i \in \rho(s_{i-1}, w_i)$ and $(y_{i-1}, y_i) \in R(w_i)$.

Let $k \geqslant 1$ be the minimal one such that w_k is not a test. Then for $1 \leqslant i < k$, we have $\neg loop(\alpha_{s_i}) \in \phi(y)$. In particular, $\neg loop(\alpha_{s_{k-1}}) \in \phi(y)$. Since w_k is not a test, it must be some program a_j. But then we must have $yj \in W'$, in contradiction to the assumption that y has no successors in W'. It follows that all the w_i's are test, i.e., $w=g_1?\, g_2? \cdots$, $M, y \models g_i$ for $i \geqslant 1$, and $\neg loop(\alpha_{s_i}) \in \phi(y)$ for $i \geqslant 1$. Since S is finite, there are $k > j \geqslant 1$ such that $s_j=s_k$. But then $\neg loop(\alpha_{s_j}) \in \phi(y)$, $s_j \in \rho(s_j, g_j? \cdots g_{k-1}?)$, and $M, y \models g_i$ for $j \leqslant i \leqslant k-1$ - a contradiction. It follows that $M, , y \models \neg loop(\alpha_p)$.

Suppose now that we have already proven the claim for all successors of a node $y \in W'$. It is easy to verify that the claim holds for y. Since W' is finite, the claim holds for all $y \in W'$.

Only if. We define W' and ϕ inductively in such a way that $M, y \models \phi(y)$ for all $y \in W'$. Initially, we have $W'=\{x\}$ and $\phi(x)=\{\neg loop(\alpha)\}$. By assumption $M, x \models \phi(x)$. Let now $y \in W'$ and $\neg loop(\alpha_p) \in \phi(y)$. If there is a word $w=g_1?\, g_2? \cdots g_m?$, $m \geqslant 1$, such that $M, y \models g_i$, $1 \leqslant i \leqslant m$, and $p \in \rho(p, w)$, then $M, y \models loop(\alpha_p)$, so this cannot be the case. Suppose now that $s \in S$, $w \in \Sigma$ and $s \in \rho(p, w)$. If w is $g?$ and $M, y \models g$ then it must be the case that $M, y \models \neg loop(\alpha_s)$,

so we put $\neg loop(\alpha_s)$ in $\phi(y)$, If w is a_j and $yj \in W$, then it must be the case that $M, yj \models \neg loop(\alpha_s)$, so we add yj to W', and put $\neg loop(\alpha_s)$ in $\phi(yj)$. It is easy to see that if this process did not terminate then we would have that $M, x \models loop(\alpha)$, therefore the process must terminate and W' is finite. ∎

Our next step is to define Hintikka trees for *loop-ADPDL*. A *Hintikka tree* for an *ADPDL* formula f with atomic programs $a_1, \ldots, a_n$ is an n-ary tree $T: [n]^* \to 2^{cl(f) \cup \{\bot\}}$ that satisfies the following conditions:

(1) $f \in T(\lambda)$.

and, for all elements x of $[n]^*$:

(2) either $T(x) = \{\bot\}$ or $\bot \notin T(x)$ and $g \in T(x)$ iff $\neg g \notin T(x)$.

(3) $g_1 \wedge g_2 \in T(x)$ iff $g_1 \in T(x)$ and $g_2 \in T(x)$.

(4) If $\langle \alpha \rangle g \in T(x)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then there exists a word $w = w_1, \ldots, w_k$ over Σ, $k \geqslant 0$, states $s_1, \ldots, s_k$ of S, and nodes $u_0, \ldots, u_k$ of $[n]^*$ such that:

(a) $u_0 = x$,

(b) $g \in T(u_k)$ and $s_k \in F$,

and, for all $1 \leqslant i \leqslant k$,

(c) $s_i \in \rho(s_{i-1}, w_i)$;

(d) if w_i is g'? then $g' \in T(u_{i-1})$ and $u_i = u_{i-1}$; and

(e) if w_i is a_j then $u_i = u_{i-1} j$.

(Note that if $k = 0$ in condition (4), then $s_0 \in F$ and $g \in T(x)$.)

(5) If $\neg \langle \alpha \rangle g \in T(x)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then $s_0 \in F$ entails $\neg g \in T(x)$, and for all $s \in S$ and $w \in \Sigma$ such that $s \in \rho(s_0, w)$:

(a) if w is g'? then either $\neg g' \in T(x)$ or $\neg \langle \alpha_s \rangle g \in T(x)$;

(b) if w is of the form a_j then $\neg \langle \alpha_s \rangle g \in T(xj)$ or $T(xj) = \{\bot\}$.

(6) If $loop(\alpha) \in T(x)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then there exists an $s' \in S$ and a $w \in \Sigma$ such that $s' \in \rho(s, w)$ and

(a) if w is g? then $g \in T(x)$ and $loop(\alpha_s) \in T(x)$;

(b) if w is a_j then $loop(\alpha_s) \in T(xj)$.

(7) If $\neg loop(\alpha) \in T(x)$ then, there is a finite subset $W' \subset [n]^*$ with $x \in W'$ and a mapping ϕ: $W' \to 2^{cl(f)}$ such that $\neg loop(\alpha) \in \phi(x)$, and if $y \in W'$ and $\neg loop(\alpha_p) \in \phi(y)$, then

- there is no word $w = g_1? g_2? \cdots g_m?$, $m \geqslant 1$, such that $g_i \in T(y)$, $1 \leqslant i \leqslant m$, and $p \in \rho(p, w)$,
- for all $s \in S$ and $w \in \Sigma$ such that $s \in \rho(p, w)$:
  - (a) if w is g? then either $\neg g \in T(y)$ or $\neg loop(\alpha_s) \in \phi(y)$,
  - (b) if w is a_j then either $yj \in W'$ and $\neg loop(\alpha_s) \in \phi(yj)$ or $\bot \in T(yj)$.

PROPOSITION 4.4 *A loop-ADPDL formula f has a one-way tree model iff it has a Hintikka tree.*

Proof. The proof is similar to the proof of Lemma 3.2. The only difference are the cases corresponding to Hintikka conditions (6) and (7) which are easily handled given Propositions 4.2 and 4.3. ∎

The next step is to build a Büchi automaton on n-ary trees over the alphabet $2^{\mathrm{cl}(f)\cup\{\perp\}}$ that accepts precisely the Hintikka trees for f. Rather than do that, we build three automata: the local automaton A_L, the $\langle\ \rangle$-automaton $A_{\langle\ \rangle}$, and the loop automaton A_{loop}, such that $T(A_L)\cap T(A_{\langle\ \rangle})\cap T(A_{loop})$ is the set of Hintikka trees for f. The local automaton checks Hintikka conditions (1)–(3), (5), and (6). It is built analogously to the local automaton for *ADPDL*. The $\langle\ \rangle$-automaton checks Hintikka condition (4) and is identical to the $\langle\ \rangle$-automaton for *ADPDL*. The loop-automaton checks Hintikka condition (7). It is a set-subtree automaton. Finally, we convert the $\langle\ \rangle$-automaton and the loop automaton to Büchi automata and combine them with the local automaton.

The Loop Automaton

The loop automaton is a set-subtree automaton $A_{loop}=(\mathrm{cl}(f)\cup\{\perp\}, \rho_{loop})$. For the transition relation ρ_{loop}, we have that $(\mathbf{s}_1,\ldots,\mathbf{s}_n)\in\rho_{loop}(\mathbf{s},\mathbf{a})$ iff:

- $\mathbf{s}\subset\mathbf{a}$, and
- if $\neg loop(\alpha)\in\mathbf{s}$, where $\alpha=(\Sigma, S, \rho, s_0, F)$ then there is no word $w=g_1?\,g_2?\cdots g_m?$, $m\geqslant 1$, such that $g_i\in\mathbf{a}$, $1\leqslant i\leqslant m$, and $s_0\in\rho(s_0, w)$, and for all $s\in S$ and $w\in\Sigma$ such that $s\in\rho(p, w)$:
  - (a) if w is $g?$ then either $\neg g\in\mathbf{a}$ or $\neg loop(\alpha_s)\in\mathbf{s}$,
  - (b) if w is a_j then either $\neg loop(\alpha_s)\in\mathbf{s}_j(yj)$ or $\perp\in\mathbf{s}_j$.

It is immediate to check that conditions (1)–(4) of the definition of set-subtree automata are satisfied for A_{loop}. Furthermore, it is easy to check that A_{loop} accepts precisely the trees that satisfy Hintikka condition (7) (note that if $\mathbf{s}=\{\perp\}$, then $(\varnothing,\ldots,\varnothing)\in\rho_{loop}(\mathbf{s},\{\perp\})$. Thus we have

PROPOSITION 4.5. *Let f be a loop-ADPDL formula with atomic programs* $a_1,\ldots,a_n$, *and let* T: $[n]^*\to 2^{\mathrm{cl}(f)\cup\{\perp\}}$ be an n-ary tree. Then T is a Hintikka tree for f iff $T\in T(A_L)\cap T(A_{\langle\ \rangle})\cap T(A_{loop})$. ∎

As for *ADPDL*, we thus have

THEOREM 4.6. *The satisfiability problem for loop-ADPDL can be solved in exponential time.*

Again, the satisfiability problem for *loop-PDL* is reducible to the satisfiability problem for *loop-ADPDL*.[4] Thus we have also reestablished an exponential upper

[4]The reduction is similar to the one in [17] but the proof that the reduction is correct is somewhat more involved in the presence of *loop*.

bound for the satisfiability problem for *loop-PDL*.[5] The reader is urged, however, to compare our proof to the proof in [23]. Note that since *loop-ADPDL* extends *ADPDL*, it has the same exponential lower bound as *ADPDL*.

5. Deterministic Propositional Dynamic Logic with Converse

Pratt's original formulation of dynamic logic included the construct *converse* [19]. For every atomic program $a \in Prog$, there is another atomic program in *Prog*, denoted a^- (the converse of a), whose semantics is running a backwards, i.e., undoing the computation performed by a. Formally, if $M=(W, R, \Pi)$ is a structure, then $R(a^-)=\{(v, u)\colon (u, v)\in R(a)\}$. We distinguish between *positive* atomic programs a, and *negative* atomic programs a^-, and we identify a^{--} with a. We use b as a generic name for either positive or negative atomic programs.

Converse-PDL, the extension of *PDL* to include the *converse* construct, satisfies the same finite model property as *PDL*, and the known decision procedures for *PDL* extend without difficulty to *converse-PDL* [6, 20]. The situation is different with *converse-DPDL*, the extension of *DPDL* to include *converse* construct. In *converse-DPDL* the positive atomic programs are deterministic while the negative atomic programs may be nondeterministic.

Proposition 5.1 [9]. *Converse-DPDL does not have the finite model property.*

Proof. Consider the formula $P \wedge [a^{-}{}^{*}]\langle a^{-}\rangle \neg P$. It is easy to verify that this formula is satisfiable in an infinite model, but is not satisfiable in any finite model. ∎

Since the finite model property fails for *converse-DPDL*, the decision procedure for *DPDL* given in [1] does not apply to *converse-DPDL*. Nevertheless, converse-*DPDL* has the tree model property. While *DPDL* has the one-way tree model property, the tree models that we construct for *converse-DPDL* are not one-way. In the tree models constructed in the previous sections, programs always connected nodes to their successors in the tree. Here we shall also have programs connecting nodes to their predecessors in the tree. Again we consider the variant *converse-ADPDL*, in which program are described by finite automata rather than by regular expressions.

Proposition 5.2. *Let f be a satisfiable converse-ADPDL formula. Then f has an n-ary tree model, where $n \leqslant |\mathrm{cl}(f)|$.*[6]

Proof. Suppose that $M, u \models f$, for some structure $M=(W, R, \Pi)$ and $u \in W$. Before going further, let us give some definitions.

[5] Since the termporal logics *UB* [2] and *CTL* [4] are expressible in *loop-PDL* [27], this upper bounds holds also for these logics.

[6] $|\mathrm{cl}(f)|$ is defined for *converse-ADPDL* exactly as it is for *ADPDL*.

An *execution sequence* is a word over an alphabet of atomic programs and tests. Consider an eventuality formula $\langle\alpha\rangle g$, where $\alpha = (\Sigma, S, \rho, s_0, F)$. If $M, u \models \langle\alpha\rangle g$, then there is an execution sequence $w = w_1 \cdots w_q$, states $s_0, \ldots, s_q$ of S and nodes $u_0, u_1, \ldots, u_q$ in W such that $u = u_0$, for all $1 \leqslant i \leqslant q$ we have $(u_{i-1}, u_i) \in R(w_i)$, $(s_{i-1}, s_i) \in \rho(w_i)$, $M, u_i \models \langle\alpha_{s_i}\rangle g$, $s_q \in F$, and $M, u_q \models g$. In this case we say that w *accomplishes* $\langle\alpha\rangle g$ at u. Let $\ll$ be some fixed linear order on execution sequences, such that if $|w| < |w'|$ then $w \ll w'$ and if $w' \ll w''$ then $ww' \ll ww''$. Note that this definition implies that if the minimum execution sequence accomplishing and eventually $\langle\alpha\rangle g$ in state u is $w_1, \ldots, w_q$ then $w_2, \ldots, w_q$ is the minimum execution sequence accomplishing $\langle\alpha_{s_1}\rangle g$ in u_1. Let $e_1, \ldots, e_n$ be an enumeration of all eventually formulas in $\mathrm{cl}(f)$. Clearly $n \leqslant |\mathrm{cl}(f)|$, so it is linear in the length of f.

In the previous section we showed that by simply unraveling M with u as a root we get a tree model for f. This is not sufficient in the presence of *converse*. Rather we have to unravel M, while ensuring that all eventualities are satisfied. Thus we construct an n-ary tree model (n is the number of eventualities), where the ith successor of a node is intended to satisfy the eventuality e_i. The technique is related to the *selective filtration* technique in [7].

We define a partial mapping $\phi: [n]^* \to W$ by induction. The tree model will be the structure $M' = (W', R', \Pi')$, where $W' = \{x: \phi(x) \text{ is defined}\}$, and $\Pi'(x) = \Pi(\phi(x))$. The relation R' will be defined by induction simultaneously with ϕ. First, we take $\phi(\lambda) = u$ and $R' = \varnothing$. Suppose now that we have already considered every member of $[n]^k$, and we have already considered $xi1, \ldots, xi(j-1)$, where $xi \in [n]^k$ and $1 \leqslant j \leqslant n$. Let e_j be $\langle\alpha\rangle g$. If $M, \phi(xi) \models \langle\alpha\rangle g$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then let $w = w_1 \cdots w_q$ be an execution sequence that is minimal according to $\ll$ and accomplishes $\langle\alpha\rangle g$ at $\phi(xi)$ (note that there is a unique such w and that it is of minimal length). If w_1 is a_l, then $\phi(xi)$ has a unique a_l-successor, that is, there is a unique $t \in W$ such that $(\phi(xi), t) \in R(a_l)$. If $(xi, x) \notin R'(a_l)$ and $(xi, xih) \notin R'(a_l)$ for $1 \leqslant h \leqslant j-1$, then we define $\phi(xij) = t$ and we put (xi, xij) in $R'(a_l)$ (these conditions are necessary to ensure that a_l is deterministic). If w_1 is a_l^-, then there are $s \in S$ and $t \in W$ such that $s \in \rho(s_0, w_1)$, $(t, \phi(xi)) \in R(a_l)$. and $w_2 \cdots w_q$ accomplishes $\langle\alpha_s\rangle g$ at t. We define $\phi(xij) = t$, and put (xij, xi) in $R'(a_l)$. If w_1 is a test, we leave $\phi(xij)$ undefined.

We now claim that if $x \in W'$ and g is any formula in $\mathrm{cl}(f)$, then $M', x \models g$ iff $M, \phi(x) \models g$. In particular we have $M', \lambda \models f$. The claim is proven by induction on the structure of the formulas. The claim is clearly true for atomic propositions, and it is straightforward to carry the induction for formulas of the form $g_1 \wedge g_2$ or $\neg g$. It remains to consider formulas of the form $\langle\alpha\rangle g$.

Suppose first that $M', x \models \langle\alpha\rangle g$. Let $w = w_1 \cdots w_q$ accomplish $\langle\alpha\rangle g$ at x. Then there are nodes $x_0, x_1, \ldots, x_q$ in W' such that $x = x_0$, for all $1 \leqslant i \leqslant q$ we have $(x_{i-1}, x_i) \in R'(w_i)$, and $M, x_q \models g$. By construction, for all $0 \leqslant i \leqslant q$, we have $(\phi(x_{i-1}), \phi(xi)) \in R(w_i)\}$, and by the induction hypothesis, $M, \phi(x_q) \models g$. Thus $M, \phi(x) \models \langle\alpha\rangle g$.

Suppose now that $M, \phi(x) \models \langle\alpha\rangle g$, where $\alpha = (\Sigma, S, \rho, s_0, F)$. Let $w = w_1 \cdots w_q$ be an execution sequence that is minimal according to $\ll$ and accomplishes $\langle\alpha\rangle g$

at $\phi(x)$. We prove that $M', x \models \langle \alpha \rangle g$, by induction on q. If $q = 0$, then $s_0 \in F$ and $M, \phi(x) \models g$. But then, by the induction on the structure of the formulas, we have $M', x \models g$ and consequently $M', x \models \langle \alpha \rangle g$. If $q \geq 1$, then there are $s \in S$ and $t \in W$ such that $s \in \rho(s_0, w_1)$, $(\phi(x), t) \in R(w_1)$, and $w_2 \cdots w_q$ accomplishes $\langle \alpha_s \rangle g$. in t. If w_1 is a test g'?, then $t = \phi(x)$, by the induction on the structure of the formula we have that $M', x \models g'$, and by induction on q we have that $M', x \models \langle \alpha_s \rangle g$. Consequently we have $M', x \models \langle \alpha \rangle g$. If w_1 is either a_j or a_j^-, then the construction guarantees that there is some $y \in W'$ such that $(x, y) \in R'(w_1)$, $(\phi(x), \phi(y)) \in R(w_1)$, and $w_2 \cdots w_q$ accomplishes $\langle \alpha_s \rangle g$ at $\phi(y)$. Furthermore, $w_2 \cdots w_q$ is minimal according to $\ll$. By the induction on q we have $M', y \models \langle \alpha_s \rangle g$, and consequently we have $M', x \models \langle \alpha \rangle g$. ∎

In the tree models for *ADPDL* eventualities are accomplished by "downward" paths. That is, if $\langle \alpha \rangle g$ is satisfied in a state x, then the sequence of states that leads to a state that satisfies g is of the form $x, xi_1, xi_1i_2, xi_1i_2i_3, \ldots$. Thus the automata that checked for satisfaction of eventualities only needed to go down the tree (we view the trees as growing downwards). In the presence of the *converse* construct, however, eventualities may require "two-way paths." Indeed, in [29] *two-way automata* are defined in order to deal with *converse*. Unfortunately, the way the emptiness problem is solved for these automata is to convert them to one-way automata with a fourfold exponential increase in the number of states. To avoid this difficulty we extend the logic by adding formulas that deal with "cycling" computations. If α is a program, then *cycle*(α) is a formula. Let $M = (W, R, \Pi)$ and $u \in W$, then $M, u \models$ *cycle*(α) if $(u, u) \in R(\alpha)$. That is, *cycle*(α) holds in the state u if there is a computation of α that starts and terminates at u. Note that we do not consider cycle formulas as formulas of *converse-ADPDL*, but they will be helpful in the decision procedure, because they enable us to check eventualities using "one-way" automata.

Let $M = (W, R, \Pi)$ be a tree structure, and let α be a program. If $M, x \models$ *cycle*(α), for $x \in W$, then there is an execution sequence $w = w_1 \cdots w_m$ accepted by α and nodes $x_0, x_1, \ldots, x_m$ of W such that $x = x_0$, $x = x_m$, and for all $0 \leq i \leq m - 1$, we have $(x_i, x_{i+1}) \in R(w_{i+1})$. If x_1 and x_{m-1} are successors of x then we say that w accomplishes *cycle*(α) at x *downwardly*. If x_1 and x_{m-1} are the predecessor of x then we say that w accomplishes *cycle*(α) at x *upwardly*.

The distinction between upward accomplishment and downward accomplishment turns out to be very useful. We therefore introduce two new types of formulas, whose semantics is defined only on tree structures. If α is a program, then both *cycle*$_d(\alpha)$ and *cycle*$_u(\alpha)$ are formulas. We call these formulas *directed cycle formulas*. Formulas of the first type are called *downward cycle formulas*, and formulas of the second type are called *upward cycle formulas*. We now define the semantics of these formulas.

Let $M = (W, R, \Pi)$ be a tree structure, and let $x \in W$. $M, x \models$ *cycle*$_d(\alpha)$ if there is an execution sequence $w = w_1 \cdots w_m$, $m \geq 1$, accepted by α and nodes $x_0, x_1, \ldots, x_m$ of W such that

- $x = x_0$ and $x = x_m$,
- $(x_i, x_{i+1}) \in R(w_{i+1})$ for all $0 \leqslant i \leqslant m-1$, and either
  - $m = 1$ (so w is a test), or
  - $m > 1$ and x_i properly succeeds x for $1 \leqslant i \leqslant m-1$.

That is, $cycle_d(\alpha)$ is satisfied at x if there is an accepting computation that consists only of a test or if it is accomplished downwardly by a computation that does not go through x except at the beginning and at the end.

Let $M = (W, R, \Pi)$ be a tree structure, and let $x \in W$. $M, x \models cycle_u(\alpha)$ if there is an execution sequence $w = w_1 \cdots w_m$, $m \geqslant 1$, accepted by α and nodes $x_0, x_1, \ldots, x_m$ of W such that

- $x = x_0$ and $x = x_m$,
- $(x_i, x_{i+1}) \in R(w_{i+1})$ for all $0 \leqslant i \leqslant m-1$, and
- $x_1 = x_{m-1}$ is the predecessor of x.

That is, $cycle_u(\alpha)$ is satisfied at x if it is accomplished upwardly. Note that $cycle_u(\alpha)$ can be satisfied at a node x even if the computation goes through x at some other points than its beginning and end. This implies that the definitions of downward cycle formulas and upward cycle formulas are not symmetric.

The relationship between the various *cycle* formulas is expressed in the following proposition. As we shall see later, when dealing with cycle formulas it suffices to consider programs of the form $(\Sigma, S, \rho, s_0, \{t\})$ (i.e., a single accepting state).

PROPOSITION 5.3. *Let $M = (W, R, \Pi)$ be a tree structure, let $x \in W$, and let $\alpha = (\Sigma, S, \rho, s, \{t\})$ be a program. Then $M, x \models cycle(\alpha)$ if and only if there are states $s_{j_1}, \ldots, s_{j_k}$ in S, where $1 \leqslant k \leqslant |S|$, such that $s_{j_1} = s$, $s_{j_k} = t$, and for $1 \leqslant i \leqslant k-1$, if $p = s_{j_i}$ and $q = s_{j_{i+1}}$ then $M, x \models cycle_u(\alpha_p^q)$ or $M, x \models cycle_d(\alpha_p^q)$.*[7]

Proof. The "if" direction is immediate. For the "only if" direction, assume that $M, x \models cycle(\alpha)$. Then there is an execution sequence $w = w_1 \cdots w_m$, states $s_0, \ldots, s_m$ in S, and nodes $x_0, x_1, \ldots, x_m$ in W such that $x = x_0$, $x = x_m$, $s_0 = s$, $s_m = t$, and for all $0 \leqslant i \leqslant m-1$, we have that $s_{i+1} \in \rho(s_i, w_{i+1})$ and $(x_i, x_{i+1}) \in R(w_{i+1})$. Let $j_1, \ldots, j_k$ be the enumeration of all the points j_i such that $x_{j_i} = x$. It is easy to show that for all $1 \leqslant i \leqslant k-1$, if $p = s_{j_i}$ and $q = s_{j_{i+1}}$ then either $M, x \models cycle_d(\alpha_p^q)$ or $M, x \models cycle_u(\alpha_p^q)$. (Note that in the degenerate case we have $k = 1$ and $s = t$.)

It remains to show that we can assume that $k \leqslant |S|$. Consider the directed graph $G = (S, E)$, with the states in S as nodes and an edge from p to q iff $M, x \models cycle_d(\alpha_p^q)$ or $M, x \models cycle_u(\alpha_p^q)$. We have shown that $M, x \models cycle(\alpha)$ iff either $s = t$ or there is a directed path in G from s to t. Clearly, if there is such a path, then there is such a path whose length is at most $|S|$. The claim follows. ∎

The crucial property of eventuality formulas that was used in constructing the automata in Section 3 is that they *propagate*. This means, that the truth of an even-

[7] α_p^q is defined in Section 2.1.

tuality in a state of a structure only depends on the truth of formulas in that state and on the truth of an eventualily in a successor state. Directed cycle formulas also propagate, but in a somewhat more complicated manner.

PROPOSITION 5.4. *Let $M=(W, R, \Pi)$ be a tree structure, let $x \in W$, and let $\alpha=(\Sigma, S, \rho, s, \{t\})$ be a program. Then $M, x \models cycle_d(\alpha)$ if and only if either there is a test $g?$ such that $M, x \models g$ and $t \in \rho(s, g?)$, or there are states $s_{j_1}, \ldots, s_{j_k}$ in S, $1 \leqslant k \leqslant |S|$, an atomic program b, and a successor y of x such that*

- $(x, y) \in R(b)$,
- *for all $1 \leqslant i \leqslant k-1$, if $p=s_{j_i}$ and $q=s_{j_{i+1}}$ then $M, y \models cycle_d(\alpha_p^q)$.*
- $s_{j_1} \in \rho(s, b)$ and $t \in \rho(s_{j_k}, b^-)$.

Proof. The "if" direction is immediate. For the "ony if" direction assume that $M, x \models cycle_d(\alpha)$. Then there is an execution sequence $w=w_1 \cdots, w_m$, $m \geqslant 1$, states $s_0, \ldots, s_m$ in S, and nodes $x_0, x_1, \ldots, x_m$ of W such that $x=x_0$, $x=x_m$, $s_0=s$, $s_m=t$, and for all $0 \leqslant i \leqslant m-1$ we have that $(x_i, x_{i+1}) \in R(w_{i+1})$ and $s_{i+1} \in \rho(s_i, w_{i+1})$. Moreover, we have that either $m=1$, or $m>1$ and x_i properly succeeds x for $1 \leqslant i \leqslant m-1$.

In the first case, w_1 must be a test $g?$ such that $M, x \models g$ and $t \in \rho(s, g?)$. In the latter case, since x_1 properly succeeds x, it must be a successor of x and w_1 must be some atomic program b such that $(x, x_1) \in R(b)$. We have to show that each time the path $x_0, \ldots, x_m$ leaves x_1, it leaves it downwardly. Let $j_1, \ldots, j_k$ be an enumeration of all the points j_i such that $x_{j_i}=x_1$. Clearly, $j_1=1$ and $s_{j_1} \in \rho(s, b)$. Also, since $x_m=x$, and $x_j \neq x$ for $1 \leqslant j \leqslant m-1$, $j_k=m-1$ and $t \in \rho(s_{j_k}, b^-)$.

We show now that if $1 \leqslant i \leqslant k-1$, $p=s_{j_i}$ and $q=s_{j_{i+1}}$, then $M, x_1 \models cycle_d(\alpha_p^q)$. There are two cases. If $j_{i+1}=j_i+1$ then $w_{j_{i+1}}$ must be a test. Consequently, $M, x_1 \models cycle_d(\alpha_p^q)$. If, on the other hand, $j_i+1<j_{i+1}$, then x_h properly succeeds x_1 for $j_i<h<j_{i+1}$, so $M, x_1 \models cycle_d(\alpha_p^q)$.

The argument showing that we can assume that $k \leqslant |S|$ is as in the proof of Proposition 5.3. ∎

COROLLARY 5.5. *Let $M=(W, R, \Pi)$ be a tree structure, let $x \in W$, and let $\alpha=(\Sigma, S, \rho, s, \{t\})$ be a program. Then $M, x \models cycle_d(\alpha)$ if and only if there is a finite subset $W' \subset W$ with $x \in W'$ and a mapping $\phi: W' \to 2^{\{cycle_d(\alpha_p^q): p, q \in S\}}$, such that $cycle_d(\alpha) \in \phi(x)$, and if $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$, then either there is a test $g?$ such that $M, y \models g$ and $q \in \rho(p, g?)$, or there are states $s_{j_1}, \ldots, s_{j_k}$ in S, $1 \leqslant k \leqslant |S|$, an atomic program b and a successor $z \in W'$ of y such that*

- $(y, z) \in R(b)$,
- *$cycle_d(\alpha_u^v) \in \phi(z)$ for all u, v such that $u=s_{j_i}$, $v=s_{j_{i+1}}$ and $1 \leqslant i \leqslant k-1$,*
- $s_{j_1} \in \rho(p, b)$ *and* $q \in \rho(s_{j_k}, b^-)$.

Proof. *If.* We claim that $M, y \models cycle_d(\alpha_p^q)$ for all $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$. In particular $M, x \models cycle_d(\alpha)$.

We now prove the claim. Let $y \in W'$ be such that it has no successors in W' and

let $cycle_d(\alpha_p^q) \in \phi(y)$. Then there is a test $g?$ such that $M, y \models g$ and $q \in \rho(p, g?)$, so by Proposition 5.4, $M, y \models cycle_d(\alpha_p^q)$. Suppose now that we have already proven the claim for all successors of a node $y \in W'$. It is easy to verify that the claim holds for y. Since W' is finite, the claim holds for all $y \in W'$.

Only if. We define W' and ϕ inductively in such a way that $M, y \models \phi(y)$ for all $y \in W'$. Initially, we put $x \in W'$ and $\phi(x) = \{cycle_d(\alpha)\}$. By assumption $M, x \models \phi(x)$. Let now $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$. Then there is an execution sequence $w = w_1 \cdots w_m$, $m \geqslant 1$, accepted by α_p^q and nodes $y_0, y_1, \ldots, y_m$ of W such that $y = y_0$, $y = y_m$ $(y_i, y_{i+1}) \in R(w_{i+1})$ for all $0 \leqslant i \leqslant m-1$, and either $m = 1$ or $m > 1$ and y_i properly succeeds y for $1 \leqslant i \leqslant m-1$. We say that $cycle_d(\alpha_p^q)$ has *rank* m at y.

In the first case w_1 must be a test $g?$ such that $M, y \models g$ and $t \in \rho(s, g?)$. In that case we do not need to extend W'. In the second case there are states $s_{j_1}, \ldots, s_{j_k}$ in S, $1 \leqslant k \leqslant |S|$, an atomic program b and a successor z of y such that

- $(y, z) \in R(b)$,
- $M, z \models cycle_d(\alpha_u^v)$, for all u, v such that $u = s_{j_i}$, $v = s_{j_{i+1}}$ and $1 \leqslant i \leqslant k-1$,
- $s_{j_1} \in \rho(p, b)$ and $q \in \rho(s_{j_k}, b^-)$.

Furthermore, the proof of Proposition 5.4 shows that the rank of $cycle_d(\alpha_u^v)$, where $u = s_{j_i}$, $v = s_{j_{i+1}}$, and $1 \leqslant i \leqslant k-1$, in z is smaller than m. We add z to W' and add $cycle_d(\alpha_u^v)$, where $u = s_{j_i}$, $v = s_{j_{i+1}}$, and $1 \leqslant i \leqslant k-1$, to $\phi(z)$.

Since, only formulas whose rank is greater than one cause addition of new nodes to W', the above process terminates, and the result satisfies the conditions of the corollary. ∎

Corollary 5.5 is very significant, since it reduces the satisfaction of downward cycle formulas to satisfaction of subformulas in a way that can be easily checked by an automaton.

We now state the propagation property of upward cycle formulas. The proof is straightforward and left to the reader.

PROPOSITION 5.6. *Let $M = (W, R, \Pi)$ be a tree structure, let $x \in W$ with predecessor y, and let $\alpha = (\Sigma, S, \rho, s, \{t\})$ be a program. Then $M, x \models cycle_u(\alpha)$ if and only if there are states $p, q \in S$ and an atomic program b such that*

- $(x, y) \in R(b)$,
- $M, y \models cycle(\alpha_p^q)$,
- $p \in \rho(s, b)$ *and* $t \in \rho(q, b^-)$.

Note that upward cycle formulas propagate undirected cycle formulas. Upward propagation, however, differs from downward propagation in a crucial way: it stops at the root of the tree, since by Proposition 5.6 no upward cycle formulas is satisfied at λ.

Having characterized satisfaction of cycle formulas, we can go back to examining eventualities. As with cycle formulas, we distinguish between downward and upward accomplishment of eventualities. We therefore introduce two new types of formulas, whose semantics is defined only on tree structures. If α is a program and g

is a formula, then both $\langle\alpha\rangle_d g$ and $\langle\alpha\rangle_u g$ are formulas. We call these formulas *directed eventualities*. Formulas of the former type are called *downward eventualities*, and formulas of the latter type are called *upward eventualities*. We now define the semantics of directed eventualities.

Let $M=(W, R, \Pi)$ be a tree structure, and let $x\in W$. We have that $M, x\models\langle\alpha\rangle_d g$ if there is an execution sequence $w=w_1\cdots w_m$, $m\geqslant 0$, accepted by α and nodes $x_0, x_1,..., x_m$ of W such that:

- $x=x_0$,
- $(x_i, x_{i+1})\in R(w_{i+1})$ for all $0\leqslant i\leqslant m-1$,
- $M, x_m\models g$, and
- there is $0\leqslant k\leqslant m$ such that $x_k=x$ and x_i properly succeeds x for $k+1\leqslant i\leqslant m$ (this is vacuously true if $k=m$).

Let $M=(W, R, \Pi)$ be a tree structure, and let $x\in W$. We have that $M, x\models\langle\alpha\rangle_u g$ if there is an execution sequence $w=w_1\cdots w_m$, $m\geqslant 1$, accepted by α and nodes $x_0, x_1,..., x_m$ of W such that:

- $x=x_0$,
- $(x_i, x_{i+1})\in R(w_{i+1})$ for all $0\leqslant i\leqslant m-1$,
- $M, x_m\models g$, and
- there is $0\leqslant k\leqslant m$ such that x_k is the predecessor of x.

Note that an upward eventuality actually requires that the computation eventually goes upward, while a downward eventuality does not require that the computation eventually goes downward. Also note that an eventuality can be satisfied both upwards and downwards. The relationship between the various types of ventualities is expressed in the next proposition.

PROPOSITION 5.7. *Let $M=(W, R, \Pi)$ be a tree structure, let $x\in W$, let α be a program, and let g be a formula. Then $M, x\models\langle\alpha\rangle g$ if and only if either $M, x\models\langle\alpha\rangle_d g$ or $M, x\models\langle\alpha\rangle_u g$.*

Proof. The "if" direction is immediate. For the "only if" direction, assume that $M, x\models\langle\alpha\rangle g$. Then there is an execution sequence $w=w_1\cdots w_m$, $m\geqslant 0$, accepted by α and nodes $x_0, x_1,..., x_m$ in W such that $x=x_0$, $M, x_m\models g$, and $(x_i, x_{i+1})\in R(w_{i+1})$ for all $0\leqslant i\leqslant m-1$. Let j be the maximal index such that $x_j=x$. If $j=m$ then $M, x\models\langle\alpha\rangle_d g$. If $j<m$, then either x_{j+1} is a successor of x, in which then $M, x\models\langle\alpha\rangle_d g$, or x_{j+1} is the predecessor of x, in which case $M, x\models\langle\alpha\rangle_u g$. ∎

We now characterize the propagation of directed eventualities.

PROPOSITION 5.8. *Let $M=(W, R, \Pi)$ be a tree structure, let $x\in W$, let $\alpha=(\Sigma, S, \rho, s, F)$ be a program, and let g be a formula. Then $M, x\models\langle\alpha\rangle_d g$ if and only if either $M, x\models cycle(\alpha_s^t)$ for some $t\in F$ and $M, x\models g$, or there are states $p, q\in S$, an atomic program b, and a successor y of x such that*

- $M, x\models cycle(\alpha_s^p)$,
- $q\in\rho(p, b)$,

- $(x, y) \in R(b)$, *and*
- $M, y \models \langle \alpha_q \rangle_d g$.

Proof. The "if" direction is immediate. For the "only if" direction, assume that $M, x \models \langle \alpha \rangle_d g$. Then there is an execution sequence $w = w_1 \cdots w_m$, $m \geqslant 0$, accepted by α and nodes $x_0, x_1, \ldots, x_m$ of W such that $x = x_0$, $M, x_m \models g$, $(x_i, x_{i+1}) \in R(w_{i+1})$ for all $0 \leqslant i \leqslant m-1$, and there is $0 \leqslant k \leqslant m$ such that $x_k = x$ and x_i properly succeeds x for $k+1 \leqslant i \leqslant m$ (the last clause is vacuously true if $k = m$). If $k = m$ then $M, x \models cycle(\alpha_s^t)$ for some $t \in F$ and $M, x \models g$. So suppose $k < m$. Let $s_0, \ldots, s_m$ be states of S such that $s_0 = s$, $s_m \in F$, and $s_{i+1} \in \rho(s_i, w_{i+1})$ for $0 \leqslant i \leqslant m-1$. Let $p = s_k$ and $q = s_{k+1}$. Since $x_k = x$, we have $M, x \models cycle(\alpha_s^p)$. Since x_{k+1} properly succeeds x_k, w_{k+1} must be some atomic program b such that $(x, x_{k+1}) \in R(b)$ and $q \in \rho(p, b)$. Finally, since x_i properly succeeds x for $k+1 \leqslant i \leqslant m$, we have that x_i succeeds x_{k+1} for $k+1 \leqslant i \leqslant m$. Consequently, $M, x_{k+1} \models \langle \alpha_q \rangle_d g$. ∎

COROLLARY 5.9. *Let $M = (W, R, \Pi)$ be a tree structure, let $x \in W$, let $\alpha = (\Sigma, S, \rho, s, F)$ be a program, and let g be a formula. Then $M, x \models \langle \alpha \rangle_d g$ if and only if there are nodes $x_0, \ldots, x_k$ of W states $s_0, t_0, \ldots, s_k, t_k$ of S, and atomic programs $b_1, \ldots, b_k$ such that*

- $x_0 = x$, $s_0 = s$, $t_k \in F$, *and* $s_{i+1} \in \rho(t_i, b_{i+1})$ *for* $0 \leqslant i \leqslant k-1$,
- x_{i+1} *is a successor of* x_i *and* $(x_i, x_{i+1}) \in R(b_{i+1})$ *for* $0 \leqslant i \leqslant k-1$,
- $M, x_i \models cycle(\alpha_{s_i}^{t_i})$ *for* $0 \leqslant i \leqslant k$, *and*
- $M, x_k \models g$.

Proof. *If.* We claim that $M, x_i \models \langle \alpha_{s_i} \rangle_d g$ for $0 \leqslant i \leqslant k$. The proof is by induction on $i = k, k-1, \ldots, 0$ using proposition 5.8 and is left to the reader.

Only if. We define the sequences inductively in such a way that $M, x_i \models \langle \alpha_{s_i} \rangle_d g$ for $0 \leqslant i \leqslant k$. Let $x_0 = x$, $s_0 = s$, and by assumption $M, x_0 \models \langle \alpha_{s_0} \rangle_d g$. Suppose that we have defined $x_0, \ldots, x_i$, $s_0, t_0, \ldots, s_i$, and $b_1, \ldots, b_i$, and we have $M, x_i \models \langle \alpha_{s_i} \rangle_d g$. Then there is an execution sequence $w = w_1 \cdots w_m$, $m \geqslant 0$, states $p_0, \ldots, p_m$ of S, and nodes $y_0, y_1, \ldots, y_m$ of W such that

- $p_0 = s_i$, $p_m \in F$, $y_0 = x_i$;
- $p_{j+1} \in \rho(p_j, w_{j+1})$ and $(y_j, y_{j+1}) \in R(w_{j+1})$ for all $0 \leqslant j \leqslant m-1$;
- $M, y_m \models g$;
- there is $0 \leqslant l \leqslant m$ such that $y_l = y_0$ and y_j properly succeeds y_0 for $l+1 \leqslant j \leqslant m$.

We say in this case that $\langle \alpha_{s_i} \rangle_d g$ has *rank* m at x_i.

Let $t_i = p_l$. Then $M, x_i \models cycle(\alpha_{s_i}^{t_i})$. There are now two cases to consider. The first case is that $l = m$. It follows that $M, x_i \models g$ and we are done. The second case is that $l > m$. Let $s_{i+1} = p_{l+1}$. Since y_{l+1} properly succeeds y_l, it must be a successor of $y_0 = x_i$. Let x_{i+1} be y_{l+1}. Thus w_{l+1} must be some atomic program b such that $(y_l, y_{l+1}) \in R(b)$ and $s_{i+1} \in \rho(t_i, b)$. Let b_{i+1} be b. By an argument similar to the one in the proof of Proposition 5.8, we can show that $M, x_{i+1} \models \langle \alpha_{s_{i+1}} \rangle_d g$ and the rank of this eventuality at x_{i+1} is smaller than m. Since each time we extend the

sequence we propagate an eventuality with a smaller rank, this process must terminate. ▮

We now state the propagation property of upward eventualities. The proof is similar to the proof of Proposition 5.8 and left to the reader.

PROPOSITION 5.10. *Let $M=(W, R, \Pi)$ be a tree structure, let $x \in W$ with predecessor y, let $\alpha=(\Sigma, S, \rho, s, F)$ be a program, and let g be a formula. Then $M, x \models \langle\alpha\rangle_u g$ if and only if there are states $p, q \in S$ and an atomic program b, such that*

- $M, x \models cycle(\alpha_s^p)$,
- $q \in \rho(p, b)$,
- $(x, y) \in R(b)$, *and*
- $M, y \models \langle\alpha_q\rangle\, g$.

Note that upward eventualities propagate undirected eventualities and that upward propagation must stop at the root of the tree.

We now define the *extended closure*, $\mathrm{ecl}(f)$, of a *converse-ADPDL* formula f (we identify a formula $\neg\neg g$ with g):

- $f \in \mathrm{ecl}(f)$.
- If $g_1 \wedge g_2 \in \mathrm{ecl}(f)$ then $g_1, g_2 \in \mathrm{ecl}(f)$.
- If $\neg g \in \mathrm{ecl}(f)$ then $g \in \mathrm{ecl}(f)$.
- If $g \in \mathrm{ecl}(f)$ then $\neg g \in \mathrm{ecl}(f)$.
- If $\langle\alpha\rangle\, g \in \mathrm{ecl}(f)$ then $g \in \mathrm{ecl}(f)$.
- If $\langle\alpha\rangle\, g \in \mathrm{ecl}(f)$, where $\alpha=(\Sigma, S, \rho, s_0, F)$, then $g' \in \mathrm{ecl}(f)$ for all $g'? \in \Sigma$.
- If $\langle\alpha\rangle\, g \in \mathrm{ecl}(f)$, where $\alpha=(\Sigma, S, \rho, s_0, F)$, then $\langle\alpha_s\rangle\, g$, $\langle\alpha_s\rangle_d\, g$, $\langle\alpha_s\rangle_u\, g \in \mathrm{ecl}(f)$ for all $s \in S$.
- If $\langle\alpha\rangle\, g \in \mathrm{ecl}(f)$, where $\alpha=(\Sigma, S, \rho, s_0, F)$, then $cycle(\alpha_s^t)$, $cycle_d(\alpha_s^t)$, $cycle_u(\alpha_s^t) \in \mathrm{ecl}(f)$ for all $s, t \in S$.

It is not hard to verify that the size of $\mathrm{ecl}(f)$ is at most quadratic in the length of f.

PROPOSITION 5.11. *Let f be a converse-ADPDL formula. Then* $|\mathrm{ecl}(f)| = O(n^2)$.

We are now in position to define Hintikka trees for *converse-ADPDL*. Unlike in Hintikka trees for *ADPDL*, it is not enough to label a node by the formulas in $\mathrm{ecl}(f)$ it satisfies, Indeed, we also have to know what program connects this node to its predecessor. Thus we also label nodes by atomic programs, and the labeling is to be interpreted as follows: if a node x is labeled by atomic program b and the predecessor of x is y, then $(x, y) \in R(b)$. Note that a node cannot be labeled by more than one atomic program.

A Hintikka tree for a *converse-ADPDL* formula f is an n-ary tree $T: [n]^* \to 2^{\mathrm{ecl}(f) \cup Prog \cup \{\bot\}}$, where n is the number of eventuality formulas in $\mathrm{cl}(f)$, that satisfies the following conditions:

(1) $f \in T(\lambda)$;

and for all $x \in [n]^*$:

(2) (2.1) $|T(x) \cap Prog| \leqslant 1$;
(2.2) if y, z are two distinct successors of x, a is a positive atomic program, and $a^- \in T(y)$, then $a^- \notin T(z)$;
(2.3) if y is a successor of x, a is a positive atomic program, and $a \in T(x)$, then $a^- \notin T(y)$;

(3) (3.1) either $T(x) = \{\perp\}$ or $\perp \notin T(x)$ and $g \in T(x)$ iff $\neg g \notin T(x)$;
(3.2) $g_1 \wedge g_2 \in T(x)$ iff $g_1 \in T(x)$ and $g_2 \in T(x)$;

(4) if $\alpha = (\Sigma, S, \rho, s, \{t\})$ is a program, then
(4.1) $cycle(\alpha) \in T(x)$ if and only if there are states $s_0, \ldots, s_m$ in S, where $0 \leqslant m \leqslant |S|$, such that $s_0 = s$, $s_m = t$, and for $0 \leqslant i \leqslant m-1$, either $cycle_u(\alpha_{s_i}^{s_{i+1}}) \in T(x)$ or $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in T(x)$;
(4.2) if y is the predecesor of x, then $cycle_u(\alpha) \in T(x)$ if and only if there are states $p, q \in S$ and an atomic program b such that
- $b \in T(x)$,
- $cycle(\alpha_p^q) \in T(y)$,
- $p \in \rho(s, b)$ and $t \in \rho(q, b^-)$;

(4.3) $cycle_d(\alpha)$ if either there is a test $g?$ such that $g \in T(x)$ and $t \in \rho(g?, s)$, or there are states $s_1, \ldots, s_m$ in S, $1 \leqslant m \leqslant |S|$, an atomic program b and a successor y of x such that
- $b^- \in T(y)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in T(y)$ for $1 \leqslant i \leqslant m-1$,
- $s_1 \in \rho(s, b)$ and $t \in \rho(s_m, b^-)$;

(4.4) $cycle_d(\alpha) \in T(x)$ only if there is a finite subset $W' \subset [n]^*$ with $x \in W'$ and a mapping ϕ: $W' \to 2^{\mathrm{ecl}(f)}$ such that $cycle_d(\alpha) \in \phi(x)$, and if $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$, then either there is a test $g?$ such that $g \in T(y)$ and $p \in \rho(q, g?)$, or there are states $s_1, \ldots, s_m$ in S, $1 \leqslant m \leqslant |S|$, an atomic program b and a successor $z \in W'$ of y such that
- $b^- \in T(z)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in \phi(z)$ for $1 \leqslant i \leqslant m-1$,
- $s_1 \in \rho(p, b)$ and $q \in \rho(s_m, b^-)$;

(5) if $\alpha = (\Sigma, S, \rho, s, F)$ is a program and g is a formula, then
(5.1) $\langle \alpha \rangle g \in T(x)$ if and only if either $\langle \alpha \rangle_d g \in T(x)$ or $\langle \alpha \rangle_u g \in T(x)$;
(5.2) if y is the predecessor of x, then $\langle \alpha \rangle_u g \in T(x)$ if and only if there are states $p, q \in S$ and an atomic program b, such that
- $cycle(\alpha_s^p) \in T(x)$,
- $q \in \rho(p, b)$,
- $b \in T(x)$,
- $\langle \alpha_q \rangle g \in T(y)$;

(5.3) $\langle \alpha \rangle_d g \in T(x)$ if either $cycle(\alpha) \in T(x)$ and $g \in T(x)$, or there are states $p, q \in S$, an atomic program b, and a successor y of x, such that
- $cycle(\alpha_s^p) \in T(x)$,

- $q \in \rho(p, b)$,
- $b^- \in T(y)$,
- $\langle \alpha_q \rangle_d g \in T(y)$;

(5.4) $\langle \alpha \rangle_d g \in T(x)$ only if there are nodes $x_0, \ldots, x_k$, states $s_0, t_0, \ldots, s_k, t_k$ of S, and atomic programs $b_1, \ldots, b_k$ such that

- $x_0 = x$, $s_0 = s$, $t_k \in F$, and $s_{i+1} \in \rho(t_i, b_{i+1})$ for $0 \leqslant i \leqslant k-1$,
- x_{i+1} is a successor of x_i and $b_{i+1}^- \in T(x_{i+1})$ for $0 \leqslant i \leqslant k-1$,
- $cycle(\alpha_{s_i}^{t_i}) \in T(x_i)$ for $0 \leqslant i \leqslant k$,
- $g \in T(x_k)$.

The clauses of the definition are meant to capture Propositions 5.3–5.10. Note, however, that conditions (4.3) and (4.4) each capture only one direction of Proposition 5.4 and Corollary 5.5. Similarly, conditions (5.3) and (5.4) each captures only one direction of Proposition 5.8 and Corollary 5.9. This will be sufficient, as the following proposition shows.

PROPOSITION 5.12. *A converse-ADPDL formula f has a tree model if and only if it has a Hintikka tree.*

Proof. Only if. Let $M = (W, R, \Pi)$ be an n-ary tree model for f (i.e., $W \subset [n]^*$). We define a Hintikka tree T: $[n]^* \to 2^{\mathrm{ecl}(f) \cup Prog \cup \{\bot\}}$ for f as follows: for a node $x \in [n]^* - W$, $T(x) = \{\bot\}$, and for a node $x \in W$,

$$T(x) = \{g \in \mathrm{ecl}(f): M, x \models g\} \cup \{b: y \text{ is the predecessor of } x \text{ and } (x, y) \in R(b)\}.$$

We leave it to the reader to verify, using Propositions 5.3–5.10 that T is a Hintikka tree for f.

If. Let T be a Hintikka tree for f. We construct a tree model for f as follows. The structure is $M = (W, R, \Pi)$, where $W = \{x \in [n]^*: T(x) \neq \{\bot\}\}$, $R(b) = \{(x, xi): xi \in W \text{ and } b^- \in T(xi)\}$, and for all $x \in W$, $\Pi(x) = \{p \in Prop: p \in T(x)\}$.

By condition (2) for Hintikka trees, M is a structure for *converse-ADPDL*, that is, $R(a)$ is deterministic for every positive atomic program a. It now remains to show that $M, \lambda \models f$. For this, we show by induction on the structure of formulas that for all $g \in \mathrm{ecl}(f)$, $g \in T(x)$ iff $M, x \models g$. For atomic propositions, this is true by construction. The inductive step for formulas of the form $g_1 \wedge g_2$ and $\neg g$ follows from condition (3). For cycle formulas, we will consider first downwards and then upwards and undirected cycle formulas.

For downwards cycle formulas, we need to prove that $cycle_d(\alpha) \in T(x)$ iff $M, x \models cycle_d(\alpha)$. If $cycle_d(\alpha) \in T(x)$, where $\alpha = (\Sigma, S, \rho, s, \{t\})$, then by condition (4.4) there is a finite subset $W' \subset W$ with $x \in W'$ and a mapping ϕ: $W' \to 2^{\mathrm{ecl}(f)}$ such that $cycle_d(\alpha) \in \phi(x)$, and if $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$, then either there is a test $g?$ such that $g \in T(y)$ and $p \in \rho(q, g?)$, in which case by the induction hypothesis $M, y \models g$, or there are states $s_1, \ldots, s_m$ in S, $1 \leqslant m \leqslant |S|$, an atomic program b and a successor $z \in W'$ of y such that

- $b^- \in T(z)$, so $(y, z) \in R(b)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in \phi(z)$ for $1 \leqslant i \leqslant m-1$,
- $s_1 \in \rho(p, b)$ and $q \in \rho(s_m, b^-)$.

Thus by Corollary 5.5, $M, x \models cycle_d(\alpha)$.

Suppose now that $M, x \models cycle_d(\alpha)$. By Corollary 5.5, there is a finite subset $W' \subset W$ with $x \in W'$ and a mapping ϕ: $W' \to 2^{\text{ecl}(f)}$, such that $cycle_d(\alpha) \in \phi(x)$, and if $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$, then either there is a test $g?$ such that $M, y \models g$ and $p \in \rho(q, g?)$, or there are states $s_1, \ldots, s_m$ in S, $1 \leqslant m \leqslant |S|$, an atomic program b and a successor $z \in W'$ of y such that

- $(y, z) \in R(b)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in \phi(z)$ for $1 \leqslant i \leqslant m-1$,
- $s_1 \in \rho(p, b)$ and $q \in \rho(s_m, b^-)$.

We claim that $cycle_d(\alpha_p^q) \in T(y)$ for all $y \in W'$ and $cycle_d(\alpha_p^q) \in \phi(y)$. In particular $cycle_d(\alpha) \in T(x)$.

We now prove the claim. Let $y \in W'$ be such that it has no successors in W' and let $cycle_d(\alpha_p^q) \in \phi(y)$. Then there is a test $g?$ such that $M, y \models g$ and $p \in \rho(q, g?)$, so by the induction hypothesis, $g \in T(y)$, and by condition (3.3), $cycle_d(\alpha_p^q) \in T(y)$. Suppose now that we have already proven the claim for all successors of a node $y \in W'$. It is easy to verify that the claim holds for y, because of condition (3.3). Since W' is finite, the claim holds for all $y \in W'$.

We now turn to upwards and undirected cycle formulas. Note that as upwards cycle formulas propagate undirected cycle formulas, we have to consider both simultaneously. We will prove by induction on the distance of a node from the root of the tree that $cycle_u(\alpha) \in T(x)$ iff $M, x \models cycle_u(\alpha)$ and $cycle(\alpha) \in T(x)$ iff $M, x \models cycle(\alpha)$.

Consider first the root of the tree. By condition (4.2) there are no upward cycle formulas in $T(\lambda)$, and by Proposition 5.6 no upward cycle formula is satisfied in λ. Thus, by condition 4.1, $cycle(\alpha) \in T(x)$ iff there are states $s_0, \ldots, s_m$ in S, $0 \leqslant m \leqslant |S|$, such that $s_0 = s$, $s_m = t$, and $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in T(\lambda)$ for $0 \leqslant i \leqslant m-1$. By the previous argument for downward eventualities, the last condition is equivalent to the following: there are states $s_0, \ldots, s_m$ in S, $0 \leqslant m \leqslant |S|$, such that $s_0 = s$, $s_m = t$, and $M, \lambda \models cycle_d(\alpha_{s_i}^{s_{i+1}})$ for $0 \leqslant i \leqslant m-1$. By Proposition 5.3, the last condition holds iff $M, \lambda \models cycle(\alpha)$.

Consider now a node $xi \in W$. By condition (4.2), $cycle_u(\alpha) \in T(xi)$ iff there are states $p, q \in S$ and an atomic program b such that

- $b \in T(xi)$,
- $cycle(\alpha_p^q) \in T(x)$,
- $p \in \rho(s, b)$ and $t \in \rho(q, b^-)$.

By the induction on nodes, the last condition is equivalent to the following: there are states $p, q \in S$ and an atomic program b such that

- $(xi, x) \in R(b)$,

- $M, x \models cycle(\alpha_p^q)$,
- $p \in \rho(s, b)$ and $t \in \rho(q, b^-)$.

By Proposition 5.6, the last condition holds iff $M, xi \models cycle_u(\alpha)$.

Finally, by condition (4.1), $cycle(\alpha) \in T(x)$ if and only if there are states $s_0, \ldots, s_m$ in S, $0 \leqslant m \leqslant |S|$, such that $s_0 = s$, $s_m = t$, and for $0 \leqslant i \leqslant m-1$ either $cycle_u(\alpha_{s_i}^{s_{i+1}}) \in T(x)$ or $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in T(x)$.

By the induction hypotheses, the last condition is equivalent to the following: there are states $s_0, \ldots, s_m$ in S, $0 \leqslant m \leqslant |S|$, such that $s_0 = s$, $s_m = t$, and for $0 \leqslant i \leqslant m-1$ either $M, x \models cycle_u(\alpha_{s_i}^{s_{i+1}})$ or $M, x \models cycle_d(\alpha_{s_i}^{s_{i+1}})$. By Proposition 5.3, the last condition holds iff $M, x \models cycle(\alpha)$.

The induction step for eventualities is analogous to the induction step for cycle formulas: first the induction is carried out for downward eventualities and, then by induction on the distance of nodes from the root, for upward and undirected eventualities. Details are left to the reader. ∎

The next step is to build a Büchi automaton on n-ary trees over the alphabet $2^{\mathrm{cl}(f) \cup Prog \cup \{\bot\}}$ that accepts precisely the Hintikka trees for f. Rather than do that, we build three automata: the local automaton A_L, the $\langle\ \rangle$-automaton $A_{\langle\rangle}$, and the cycle automaton A_{cycle}, such that $T(A_L) \cap T(A_{\langle\rangle}) \cap T(A_{cycle})$ is the set of Hintikka trees for f. The local automaton checks Hintikka conditions (1)–(3), (4.1–3), and (5.1–3). It is built analogously to the local automaton for *ADPDL*. The $\langle\ \rangle$-automaton checks Hintikka condition (5.4). It is a set-subtree automaton. The cycle automaton checks Hintikka condition (4.4). It is also a set-subtree automaton. Finally, we convert the $\langle\ \rangle$-automaton and the cycle automaton to Büchi automata and combine them with the local automaton.

The $\langle\ \rangle$-Automaton

The $\langle\ \rangle$-automaton is a set-subtree automaton $A_{\langle\rangle} = (\mathrm{ecl}(f) \cup Prog \cup \{\bot\}, \rho_{\langle\rangle})$. For the transition relation $\rho_{\langle\rangle}$, we have that $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \in \rho_{\langle\rangle}(\mathbf{s}, \mathbf{a})$ iff:

- $\mathbf{s} \subset \mathbf{a}$, and
- If $\langle\alpha\rangle_d g \in \mathbf{s}$, where $\alpha = (\Sigma, S, \rho, s, F)$, then there is a state $p \in S$ such that $cycle(\alpha_s^p) \in \mathbf{a}$ and either $p \in F$ and $g \in \mathbf{a}$ or there is an atomic program b and a state $q \in \rho(p, b)$ such that for some $\mathbf{s}_j$ we have $b^- \in \mathbf{s}_j$ and $\langle\alpha_q\rangle_d g \in \mathbf{s}_j$.

The Cycle Automaton

The cycle automaton is a set-subtree automaton $A_{cycle} = (\mathrm{ecl}(f) \cup Prog \cup \{\bot\}, \rho_{cycle})$. For the transition relation ρ_{cycle}, we have that $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \in \rho_{cycle}(\mathbf{s}, \mathbf{a})$ iff:

- $\mathbf{s} \subset \mathbf{a}$, and
- if $cycle_d(\alpha) \in \mathbf{s}$, where $\alpha = (\Sigma, S, \rho, s, \{t\})$, then either there is a test $g?$ such that $g \in \mathbf{a}$ and $t \in \rho(s, g?)$ or these are states $s_1, \ldots, s_m$ in S, $1 \leqslant m \leqslant |S|$, and an atomic program b such that for some $\mathbf{s}_j$ we have
  - $b^- \in \mathbf{s}_j$,
  - $cycle_d(\alpha_{s_i}^{s_{i+1}}) \in \mathbf{s}_j$ for $1 \leqslant i \leqslant m-1$, and
  - $s_1 \in \rho(s, b)$ and $t \in \rho(s_m, b^-)$.

It is immediate to check that conditions (1)–(4) of the definition of set-subtree automata are satisfied for $A_{\langle\rangle}$ and A_{cycle}. Furthermore, $A_{\langle\rangle}$ and A_{cycle} clearly accepts precisely the trees that satisfy Hintikka conditions (5.4) and (4.4), correspondingly. Thus we have

PROPOSITION 5.13. *Let f be a converse-ADPDL formula, and let $T: [n]^* \to 2^{cl(f) \cup Prog \cup \{\perp\}}$ be an n-ary tree. Then T is a Hintikka tree for f iff $T \in T(A_L) \cap T(A_{\langle\rangle}) \cap T(A_{cycle})$.* ∎

As for *ADPDL*, we have proven

THEOREM 5.14. *The satisfiability problem for converse-ADPDL can be solved in exponential time.*

Again, the satisfiability problem for *converse-PDL* is reducible to the satisfiability problem for *converse-ADPDL*. Thus we have also reestablished an exponential upper bound for the satisfiability problem for *converse-PDL*. Note that since *converse-ADPDL* extends *ADPDL*, it has the same exponential lower bound as *ADPDL*.

6. CONCLUDING REMARKS

We have presented a unifying technique for obtaining decision procedures for modal logics of programs. We have demonstrated our technique by proving exponential upper bounds for several variants of deterministic propositional dynamic logic. In [32] we sketched a proof of an exponential upper bound for a propositional μ-calculus $M\mu^-$. A full proof of this result will appear in a future paper.

Our technique is based on the tree model property, which seems to be more fundamental then the small model property. Furthermore, we can actually use our technique to prove the small model property. The algorithm for testing emptiness of Büchi automata works by checking the existence of good embedded subtrees. These subtrees can be combined to form a finite structure that can be unraveled into an accepting run. The size of this structure is polynomial in the size of the given automaton. When this automaton is the automaton that accepts precisely the set of Hintikka trees for a formula f, the above structure is actually a model for f. Thus, if a formula f of *loop-ADPDL* is satisfiable, then it has a model whose size is at most exponential in the length of f. In the case of *converse-DPDL*, this construction does not work. Indeed, because of the presence of the *converse* construct, positive atomic programs in the resulting structure may be nondeterministic. This is to be expected, since *converse-DPDL* does not have the finite model property.

In [14], the maximal model technique was used to prove completeness of an axiom system for *PDL*. Their technique was extended in [1] to *DPDL*. Our automata-theoretic technique can also be used to prove completeness results. The

idea is to allow only *consistent* sets of formula as states in the automata, and then to prove that automata that are built from consistent states necessarily accept some trees. This technique fails, however, for *converse-ADPDL*, since the extended closure for a *converse-ADPDL* formula includes formulas that are not *converse-ADPDL* formulas. Thus the axiomatization of *converse-ADPDL* remains an open problem.

APPENDIX

In [32] we defined *eventuality automata.* In eventuality automata, the acceptance condition is specified by a collection $F \subset 2^S$ of designated sets. A run ϕ of A over T is accepting if and only if, for all infinite paths p starting at λ we have $\inf(\phi, p) \cap X$ for all $X \in F$.

Having defined eventuality automata, we went on to prove that the emptiness problem for eventuality automata can be solved in polynomial time. Unfortunately, while writing that paper we were not aware of [25]. We show now that every eventuality automaton can be converted to an equivalent Büchi automaton, with a polynomial increase in size. This conversion, together with Theorem 1.2, yields a polynomial time algorithm for the emptiness of eventuality automata.

THEOREM A.1. *Let* $A = (\Sigma, S, \rho, S_0, \{F_0, \ldots, F_{k-1}\})$ be an eventuality automaton. There is a Büchi automaton A' with $k|A|$ states such that $T(A) = T(A')$.

Proof. Let $A' = (\Sigma, S', \rho', S_0', F)$, $S' = S \times \{0, \ldots, k-1\}$, $S_0' = S_0 \times \{0\}$, $F' = F_0 \times \{0\}$, and for $0 \leqslant i \leqslant k-1$ $((s_1, j), \ldots, (s_n, j)) \in \rho'((s, i), \sigma)$ iff $(s_1, \ldots, s_n) \in \rho(s, \sigma)$, and either $s \notin F_i$ and $i = j$ or $s \in F_i$ and $j = i + 1 \pmod k$. We leave it to the reader to show that $T(A) = T(A')$. ∎

We note that in [32] we had a direct reduction from the satisfiability problem to the emptiness problem for eventuality automata. In the process of writing the proofs we realized that they can be significantly simplified by the introduction of subtree automata and set-subtree automata.

ACKNOWLEDGMENTS

We'd like to thank the following people for helpful discussions and comments: E. A. Emerson, R. Fagin, J. Y. Halpern, M. Karpinski, J. C. Michell, R. Parikh, and A. P. Sistla.

REFERENCES

1. M. BEN-ARI, J. Y. HALPERN, AND A. PNUELI, Deterministic propositional dynamic logic: finite models, complexity, and completeness, *J. Comput. System Sci.* **25** (1982), pp. 402–417.
2. M. BEN-ARI, Z. MANNA, A. PNUELI, The temporal logic of branching time, *Proc. 8th ACM Symp. on Principles of Programming Languages*, Williamsburg, 1981, pp. 164–176.

3. J. R. BÜCHI, On a decision method in restricted second order arithmetic, *Proc. Internat Congr. Logic, Method and Philos Sci. 1960*, Stanford University Press, 1962, pp. 1–12.
4. E. A. EMERSON, J. Y. HALPERN, Decision procedures and expressiveness in the temporal logic of branching time, *J. Computer and System Sciences* **30** (1985), pp. 1–24.
5. E. A. EMERSON, A. P. SISTLA, Deciding branching time logic, *Proc. 16th ACM Symp. on Theory of Computing*, Washington, 1984, pp. 12–24.
6. M. J. FISHER, R. E. LADNER, Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18**(2), 1979, pp. 194–211.
7. D. M. GABBAY, "Investigation in Modal and Tense Logic," Reidel, 1976.
8. Z. HABASINSKI, "Decidability Problems in Logics of Programs," Ph.D. Thesis, Dpt. of Mathematics, Technical University of Poznan, 1983.
9. J. Y. HALPERN, Private communication, 1983.
10. R. HOSSLEY, C. W. RACKOFF, The emptiness problem for automata on infinite trees, *Proc. 13th IEEE Symp. on Switching and Automata Theory*, 1972, pp. 121–124.
11. D. HAREL, R. SHERMAN, Looping vs. repeating in dynamic logic, *Information and Control* **55** (1982), pp. 175–192.
12. D. HAREL, R. SHERMAN, Propositional dynamic logic of flowcharts, *Proc. Int. Conf. on Foundations of Computational Theory*, Lecture Notes in Computer Science, vol. 158, Springer-Verlag, Berlin, 1983, pp. 195–206.
13. N. D. JONES, W. T. LAASER, Complete problems in deterministic polynomial time, *Theoretical Computer Science* **3** (1977), pp. 105–117.
14. D. KOZEN, R. PARIKH, An elementary proof of the completeness of *PDL*, *Theoretical Computer Science* **14** (1), 1981, pp. 113–118.
15. A. R. MEYER, Weak monadic second order theory of successor is not elementary recursive, *Proc. Logic Colloquium*, 1975, Lecture Notes in Mathematics, vol. 453, Springer-Verlag, pp. 132–154.
16. D. E. MÜLLER, Infinite sequences and finite machines, *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical Design*, New York, 1963, pp. 3–16.
17. R. PARIKH, Propositional logics of programs: systems, models and complexity, *Proc. 7th Symp. on Principles of Programming Languages*, Las Vegas, 1980, pp. 186–192.
18. A. PNUELI, The temporal logic of concurrent programs, *Theoretical Computer Science* **13** (1981), pp. 45–60.
19. V. R. PRATT, Semantical considerations on Floyd-Hoare logic, *Proc. 17th IEEE Symp. on Foundations of Computer Science*, Houston, October 1976, pp. 109–121.
20. V. R. PRATT, Models of program logics, *Proc. 20th IEEE Symp. on Foundations of Computer Science*, San Juan, 1979, pp. 115–122.
21. V. R. PRATT, A near-optimal method for reasoning about action, *J. Comput. System Sci.* **20** (1980), pp. 231–254.
22. V. R. PRATT, Using graphs to understand *PDL*, *Proc. Workshop on Logics of Programs*, (D. Kozen, ed.), Yorktown-Heights, Lecture Notes in Computer Science, vol. 131, Springer-Verlag, Berlin, 1982, pp. 387–396.
23. A. PNUELI, R. SHERMAN, Propositional dynamic logic of looping flowcharts, Technical Report, Weizmann Institute, Rehovot, Israel, 1983.
24. M. O. RABIN, Decidibility of second order theories and automata on infinite trees, *Trans AMS*, **141** (1969), pp. 1–35.
25. M. O. RABIN, Weakly definable relations and special automata, *Proc. Symp. Math. Logic and Foundations of Set Theory* (Y. Bar-Hillel, ed.), North-Holland, 1970, pp. 1–23.
26. M. O. RABIN, Automata on infinite objects and church's problem, *Proc. Regional AMS Conf. Series in Math.* **13** (1972), pp. 1–22.
27. R. SHERMAN, "Variants of Propositional Dynamic Logic," Ph.D. Dissertation, The Weizmann Inst. of Science, 1984.
28. R. S. STREETT, "A Propositional Dynamic Logic for Reasoning about Program Divergence," M. Sc. Thesis, MIT, 1980.

29. R. S. STREETT, Propositional dynamic logic of looping and converse is elementarily decidable, *Information and Control* 54 (1982), pp. 121–141.
30. J. W. THATCHER, J. B. WRIGHT, Generalized finite automata theory with an application to a decision problem of second-order logic, *Mathematical System Theory*, 2 (1968), pp. 57–81.
31. M. Y. VARDI, L. STOCKMEYER, Improved upper and lower bounds for modal logics of programs, *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 240-251.
32. M. Y. VARDI, P. WOLPER, Automata-theoretic techniques for modal logics of programs, *Proc. 16th ACM Symp. on Theory of Computing*, Washington, 1984, pp. 446–456.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium