Improved Upper and Lower Bounds for Modal Logics of Programs: Preliminary Report

Moshe Y. Vardi†

Center for Study of Language and Information at Stanford University

Larry Stockmeyer‡

IBM Research Laboratory at San Jose

Abstract. We describe novel techniques for establishing improved upper and lower bounds for modal logics of programs: 1) We introduce *hybrid* tree automata. These automata seems to be doubly exponential more powerful than Rabin tree automata but their emptiness problem is only exponentially harder (nondeterministic exponential time vs. nondeterministic polynomial time). The satisfiability problem for several logics is reducible to the emptiness problem for hybrid tree automata. Using this reduction we show that the the satisfiability problems for Streett's delta-*PDL*, Kozen's μ-calculus and Parikh's game logic are solvable in nondeterministic exponential time, and the satisfiability problem for Emerson and Halpern's *CTL** and Vardi and Wolper's process logic (*YAPL*) are solvable in nondeterministic doubly exponential time. 2) We encode Turing machine computations by Kripke structures where every state in the structure represents a *single* tape cell. This yields a deterministic doubly exponential time lower bound for *CTL** and *YAPL*. 3) For variants of *CTL** and *YAPL* that deal only with finite computations we prove completeness for deterministic doubly exponential time.

† Address: CSLI, Ventura Hall, Stanford University, Stanford, CA 94305.

‡ Address: IBM Research Laboratory, 5600 Cottle Rd., San Jose, CA 95193.

1. Introduction

While *dynamic logic* [Pr76], and in particular its propositional version *PDL* [FL79], has proven to be a very useful tool to reason about the input/output behavior of programs, it has become clear that it is not adequate for reasoning about the ongoing behavior of programs, and, in particular, about the ongoing behavior of non-terminating programs (such as operating systems). In view of this shortcoming, numerous extensions were studied in the literature (e.g., [BMP81], [EH82], [EH83], [HKP82], [HS83], [Ko83], [Ni80], [Pa78], [Pa83], [Pn81], [Sh84], [St80], [VW83]).

For the extended logics to be a useful tool in program verification, they ought to have a decidable satisfiability problem. A general technique to prove their decidability is by reduction to *SnS*, the second-order theory of n-ary trees [Ga76]. Rabin has shown that *SnS* is decidable [Ra69], but the upper bound established by that reduction is, unfortunately, nonelementary (i.e., the time complexity cannot be bounded by any stack of exponentials of a fixed height) [Me75]. The only known lower bound for the time complexity of the above logics is, however, only exponential, and it is essentially the lower bound proved by Fischer and Ladner [FL79] for *PDL*.

A methodology to prove elementary upper bounds was suggested by Streett [St80]. Almost all program logics have the *tree model property*. That is, models of these logics can be viewed as labeled graphs and these

graphs can be unraveled into bounded-branching infinite tree-structured models. The reduction of the logics to *SnS* depends crucially on this property. The decidability of *SnS* has been established via a reduction to the *emptiness problem* of automata on infinite trees (i.e., the problem whether a given automaton accepts some tree) [Ra69]. This suggest that decision procedures for program logics can be obtained by directly reducing satisfiability to that emptiness problem. The idea is, for given a formula f, to construct a tree automaton A_f such that A_f accepts exactly the tree models of f. Thus f is satisfiable if and only if A_f accepts some tree. This approach was exploited by Streett [St80,St82] to establish elementary upper bounds for *PDL* augmented with the *repeat* and *converse* constructs. This approach was pursued further by other researchers ([ESi84, ESt84, Ha83, VW84]) to establish elementary upper bounds for several logics. Almost all program logics studied in the literature are now known to have an elementary decision procedure.

The complexity of tree-automata based decision procedures depends on two factors: the complexity of the reduction, that is, how big is A_f compared to f, and the complexity of testing emptiness of tree automata, which depends on the type of tree automata in question. (As we shall see later, there are several types of tree automata.)

It turns out that there are two general cases. For some logics, e.g., *loop − PDL* [HS83], it is possible to have an exponential reduction to Buchi tree automata [VW84]. Namely, given a formula f, one can construct a Buchi tree automaton A_f whose size is at most exponential in the length of f, such that f is satisfiable if and only if A_f accepts some tree. Since the emptiness problem for Buchi tree automata can be solved in polynomial time [Ra70], this reduction yields an exponential time upper bound, which matches the known lower bound. (This upper bound was first proven in [PS83].)

For other logics, e.g., *delta − PDL* [St80], things do not work out so nicely. First, Buchi automata are not expressive enough (i.e., there are formulas in these logics whose class of tree models is not definable by a Buchi tree automaton). For these logics we have to use the more expressive Rabin tree automata, for which the best (previously) known algorithm for emptiness runs in exponential time [Ra72]. Furthermore, the complexity of the reduction itself can be quite prohibitive, e.g., the reduction of *delta − PDL* is doubly exponential [St80] and it is even more expensive for other logics [VW83]. Thus, while reducing the satisfiability problem to the emptiness problem of tree automata does give us elementary upper bounds, it yields decision procedures that run in at least triply exponential time, leaving tantalizing gaps between the known lower bounds and upper bounds.

It is this gap that we narrow (and in one case close) in this paper. We define a new class of tree automata, called *hybrid automata*. A hybrid automaton is a pair (A,B), where A is a Rabin tree automaton and B is a Buchi sequential automaton (on infinite words). A tree T is accepted by (A,B) if it is accepted by A and all paths of T are *rejected* by B. Hybrid automata are not more expressive than Rabin tree automata, but they seem to be more powerful − the only translation we know of from Rabin automata to hybrid automata is doubly exponential. The deepest result of the paper is that while hybrid automata are apparently doubly exponentially more powerful than Rabin automata, their emptiness problem is only exponentially harder: we can test emptiness of Rabin automata in nondeterministic polynomial time (this is an improvement upon the exponential algorithm of [Ra72]), and we can test emptiness of hybrid automata in nondeterministic exponential time. (We also prove a deterministic exponential time lower bound for emptiness of hybrid automata.)

Because of the added power of hybrid automata, it is easier to reduce satisfiability to the emptiness of hybrid automata than to reduce it to emptiness of Rabin tree automata. Using hybrid automata we get improved upper bounds for almost all logics for which the previously known upper bounds were superexponential. The logics that we consider divide into three classes. For the logics in the first class, e.g., Street's *delta − converse − PDL* [St82], Kozen's μ*−calculus* [Ko83], or Parikh's game logic [Pa83], our technique yields a *nondeterministic* exponential time upper bound (recall that the lower bound is *deterministic* exponential time). Previously known upper bounds for these logics vary from triply exponential time for the μ*−calculus* [ESt84] to octuply exponential time for *delta − converse − PDL* [St82].

For logics in the second class, e.g., Emerson and Halpern's branching time logic CTL^* [EH83] or Vardi and Wolper's process logic *YAPL* [VW83], our technique yields a nondeterministic *doubly* exponential time. Previously known upper bounds for these logics vary

from triply exponential time for CTL^* [ESi84] to quadruply exponential time for *YAPL* [VW83]. For these logics, however, we prove a stronger lower bound: deterministic *doubly* exponential time. This is the first superexponential time lower bound to be proven for a propositional program logic. Thus for these logics the gap between the lower and the upper bounds is also narrowed to the gap between deterministic and nondeterministic time.

Finally, for some natural variants of CTL^* and *YAPL* (essentially, their restriction to finite computations) we completely close the gap between the lower and the upper bounds by proving that the satisfiability problem for these logics is complete for deterministic doubly exponential time.

2. Automata-Theoretic Background

2.1. Sequential Automata

A *sequential (transition) table* is a tuple $\tau=(\Sigma,S,\rho,S_0)$, where Σ is the alphabet, S is a set of states, $\rho:S\times\Sigma\rightarrow 2^S$ is the transition function, and $S_0\subseteq S$ is the set of starting states. A *run* of τ over an infinite word $w=a_1a_2\cdots$ is an infinite sequence of states $s=s_0,s_1\cdots$ such that $s_0\in S_0$ and $s_{i+1}\in\rho(s_i,a_i)$ for $i\geq 0$. The set $inf(s)$ is the set of states that repeat infinitely often in s, i.e., $inf(s)=\{s : |\{i : s_i=s\}|=\infty\}$.

A *sequential automaton*, abbr. sa, consists of a sequential table τ and an *acceptance condition*. Various acceptance conditions give rise to different kinds of sa. A *Buchi* acceptance condition is specified by a set of repeating states [Bu62]. That is, a Buchi sa A is a pair (τ,F), where $\tau=(\Sigma,S,\rho,S_0)$ is a sequential table and $F\subseteq S$. *A accepts* an infinite word w if there is a run s of τ on w such that some state in F repeats in s infinitely often, that is, $inf(s)\cap F\neq\emptyset$. $L(A)$ is the set of infinite words accepted by A.

The *emptiness problem* for sa is to determine, given an sa, whether it accepts some infinite word.

Theorem 2.1.1. [SVW84] The emptiness problem for Buchi sa is logspace complete for NLOGSPACE. ■

Deterministic Buchi sa are less expressive than *nondeterministic* Buchi sa [Ch74]. Buchi sa can, however, be determinized by using a more general acceptance condition. A *Rabin* acceptance condition is a collection of pairs of sets of states [Ra72]. Intuitively, a pair (L,U) means that some state in U repeats infinitely often and no state in L repeats infinitely often. Formally, a Rabin sa A is a pair (τ,F), where $\tau=(\Sigma,S,\rho,S_0)$ is a sequential table and and $F\subseteq(2^S)^2$. *A accepts* an infinite word w if there is a run s of τ on w such that $inf(s)\cap L=\emptyset$ and $inf(s)\cap U\neq\emptyset$ for some $(L,U)\in F$. Note that a Buchi acceptance condition can be viewed as a special case of a Rabin acceptance condition where $F=\{(\emptyset,F)\}$.

For a description of other types of sa the reader is referred to [Ch74].

In this paper we are interested in determinizing and complementing Buchi sa simultaneously.

Theorem 2.1.2. [Va84] Given a Buchi sa A with n states, we can effectively construct a deterministic Rabin sa $B=(\tau,F)$ such that $L(B)=\Sigma^\omega-L(A)$, where B has $O(c^{c^{n^2\log n}})$ states and F has $O(c^{n^2})$ pairs, for some constant c. ■

We note that the construction to determinize Buchi sa is also doubly exponential [McN66,Va84].

2.2. Tree Automata

Let $[k]$ denote the set $\{1,\ldots,k\}$. A k-ary infinite tree h over an alphabet Σ is a mapping $[k]^*\rightarrow\Sigma$. A *path* starting at a node $x\in[k]^*$ is a finite or infinite sequence $P=x_0,x_1,\ldots$ such that $x_0=x$ and x_{i+1} is a successor of x_i for $0\leq i<|P|$. $h(P)$ denotes the word $h(x_0)h(x_1)\cdots$.

A *tree (transition) table* is a tuple $\theta=(\Sigma,T,\psi,T_0)$, where Σ is the alphabet, T is a set of states, $\psi:T\times\Sigma\rightarrow 2^{T^k}$ is the transition function, and $T_0\subseteq T$ is the set of starting states. Note that the transition function gives for every state and letter a set of k-tuples of states, since it has to specify successor states for all children of a node on the tree. A run of θ on a tree $h:[k]^*\rightarrow\Sigma$ is a tree $r:[k]^*\rightarrow T$ where $r(\lambda)\in T_0$ and for every $x\in[k]^*$, if $r(x)=s$ then $\langle r(x1),\ldots,r(xk)\rangle\in\psi(s,h(x))$.

A *tree automaton*, abbr. ta, consists of a tree table θ and an *acceptance condition*. Various acceptance conditions give rise to different kind of ta. As before, a Buchi acceptance condition is specified by a set of repeating states. That is, a Buchi ta A is a pair (θ,F), where $\theta=(\Sigma,T,\rho,T_0)$ is a tree table and $F\subseteq T$. *A accepts* a tree h if there is a run r of τ on h such that, for all infinite paths P, some state in F repeats on P infinitely often, that is, $inf(r(P))\cap F\neq\emptyset$.

Buchi ta are not expressive enough to characterize tree models of all program logics (though they are expressive enough for certain logics [VW84]). As before, a Rabin acceptance condition is a collection of pairs of sets of states [Ra72]. Formally, a Rabin ta is is a pair $A=(\theta,\mathbf{F})$, where $\theta=(\Sigma,T,\psi,T_0)$ is a tree table and $\mathbf{F}\subseteq(2^T)^2$. A *accepts* a tree h if there is a run $\mathbf{r}$ of θ on h such that, for all infinite paths P, $inf(\mathbf{r}(P))\cap L=\emptyset$ and $inf(\mathbf{r}(P))\cap U\neq\emptyset$ for some $(L,U)\in\mathbf{F}$.

For a description of other types of ta the reader is referred to [HR72, Ra69, St80, St82].

The *emptiness problem* for ta is to determine, given an ta, whether it accepts some tree.

Theorem 2.2.1.

1. [Ra70,VW84] The emptiness problem for Buchi ta is logspace complete for PTIME.
2. [Ra72] The emptiness problem for Rabin ta is solvable in (deterministic) exponential time. ■

3. Dynamic Logic and Tree Automata

We now describe Streett's idea of reducing the satisfiability problem for dynamic logic to the emptiness problem of Rabin ta. We demonstrate this idea for $delta-PDL$, an extension of PDL that can deal with infinite computation [St80]. For simplicity we deal here with $delta-DPDL$ in which atomic programs are required to be deterministic. (We note that there is a polynomial reduction of satisfiability of $delta-PDL$ formulas to satisfiability of $delta-DPDL$ formulas [Pa80].) $Delta-DPDL$ extends $DPDL$ by formulas of the form $\Delta\alpha$, where α is a program. Intuitively, $\Delta\alpha$ means that α can be executed repeatedly infinitely many times. We now give a formal definition.

Formulas of $delta-DPDL$ are built from a set $Prop$ of atomic propositions and a set $Prog$ of atomic programs. The sets of *formulas* and *programs* are defined inductively as follows:

- Every atomic 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 α is a program and f is a formula, then $\langle\alpha\rangle f$ and $\Delta\alpha$ are formulas.
- Every atomic program $a\in Prog$ is a program.
- If f is a formula, then $f?$ is a program.
- If α and β are programs, then $\alpha;\beta$, $\alpha\cup\beta$, and α^* are programs.

$Delta-DPDL$ formulas are interpreted over structures $M=(W,R,\Pi)$ where W is a set of states, $R:Prog\rightarrow 2^{W\times W}$ 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 $\Pi:W\rightarrow 2^{Prop}$ assigns truth values to the propositions 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:

- $R(f?)=\{(u,u):M,u\models f\}$.
- $R(\alpha\cup\beta)=R(\alpha)\cup R(\beta)$.
- $R(\alpha;\beta)=R(\alpha)\circ R(\beta)$ (relational composition).
- $R(\alpha^*)=(R(\alpha))^*$ (reflexive-transitive closure).
- 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 $M,u'\models f$.
- $M,u\models\Delta\alpha$ iff there is an infinite sequence $u_0,u_1,\cdots$ of states of W such that $u_0=u$ and $(u_i,u_{i+1})\in R(\alpha)$ for all $i\geq 0$.

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

To establish a decision procedure for $delta-DPDL$ using tree automata, the first thing we have to prove is that $delta-DPDL$ has the tree model property. Indeed, if a formula f is satisfiable, then it is satisfiable in some state s of some structure M. Now, M can be unraveled into a tree with s 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.

The next step is to convert this tree model to a tree labeled by formulas from the *closure* of f, denoted $cl(f)$, which is, in some general sense, the set of all subformulas of f, of which there are only linearly (in the length of f) many. The idea is to label each node by the formulas that are true in it. Since the tree model is not a full tree, we add dummy nodes and label them by a special symbol $\perp$. Trees that correspond to tree models satisfy some special properties.

The closure of a formula f, denoted $cl(f)$, is defined as follows (we identify a formula g with $\neg\neg g$):

- $f \in cl(f)$
- If $g_1 \wedge g_2 \in cl(f)$ then $g_1, g_2 \in cl(f)$.
- If $\neg g \in cl(f)$ then $g \in cl(f)$.
- If $g \in cl(f)$ then $\neg g \in cl(f)$.
- If $\langle\alpha\rangle g \in cl(f)$ then $g \in cl(f)$.
- If $\langle\alpha;\beta\rangle g \in cl(f)$ then $\langle\alpha\rangle\langle\beta\rangle g \in cl(f)$.
- If $\langle\alpha \cup \beta\rangle g \in cl(f)$ then $\langle\alpha\rangle g \in cl(f)$ and $\langle\beta\rangle g \in cl(f)$.
- If $\langle\alpha^*\rangle g \in f$ then $\langle\alpha\rangle\langle\alpha^*\rangle g \in cl(f)$.
- If $\Delta\alpha \in cl(f)$ then $\langle\alpha\rangle\Delta\alpha \in cl(f)$.

A *Hintikka tree* for a *delta – DPDL* formula f with atomic programs $a_1, \ldots, a_k$ is an k-ary tree $T:[k]^* \to 2^{cl(f) \cup \{\bot\}}$ that satisfies the following conditions:

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

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

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 $\bot \notin T(x)$ then $\langle\alpha\rangle g \in T(x)$ iff there is a descendant y of x such that $g \in T(y)$, and the finite path between x and y is a computation of α,

5) if $\Delta\alpha \in T(x)$, then $\langle\alpha\rangle\Delta\alpha \in T(x)$,

6) if $\neg\Delta\alpha \in T(x)$, then there is no infinite path P starting at x such that P is a computation of α^ω.

(We leave the notion "is a computation of" to the intuition of the reader. A formal definition will be given in the full paper.)

Proposition 3.1. A *delta – DPDL* formula f has a tree model iff it has a Hintikka tree. ■

Now to test whether f is satisfiable, it suffices to construct a ta A_f that accepts precisely all the Hintikka trees of f, and then test for the emptiness of A_f. A_f has to check that conditions 1-6 in the definition of Hintikka trees are satisfied. Conditions 1-5 are relatively easy to deal with. In fact, it is shown in [VW84] that these condition can be checked by a Buchi ta A_1, whose size is at most exponential in f. (Indeed, since only conditions 1-5 are needed for *DPDL*, and since, by Theorem 2.2.1, the emptiness problem for Buchi ta can be solved in polynomial time, this was used in [VW84] to reprove that the satisfiability problem for *DPDL* is solvable in exponential time. This upper bound was originally proven in [BHP82].) The difficulty seems rather to lie with the sixth condition that deals with negation of Δ formulas.

To deal with this condition we do the following. First we construct a Buchi sa B_1 that accepts an infinite word $w = b_1 b_2 \cdots$ over $2^{cl(f) \cup \{\bot\}}$ iff there is some i such that for some $\Delta\alpha \in cl(f)$ we have that $\neg\Delta\alpha \in b_i$ and $b_{i+1} b_{i+2} \cdots$ is a computation of α^ω. In other words B_1 accepts a path if it violates some negated Δ formula. The size of B_1 is linear in the length of f.

Now we use Theorem 2.1.2 to determinize and complement B_1. That is, we construct a deterministic Rabin sa B_2 that is the complement of B_1. The size of B_2 is doubly exponential in the length of f. We now construct a Rabin ta A_2 that accepts a tree iff all paths in the tree are accepted by B_2. This construction can be easily done with no increase in size, since B_2 is deterministic. It can be seen that determinism is essential here.

Finally, we compose A_1 and A_2 to get a Rabin ta A_f that accepts precisely the Hintikka trees of f. The size of A_f is doubly exponential in the length of f. Since, by Theorem 2.2.1, the emptiness problem for Rabin ta is solvable in exponential time, we have a triply exponential time decision procedure for satisfiability of *delta – DPDL*. (The proof in [St80] for *delta – PDL* is actually different from the one we outlined here, since he did not have the results of [Ra72] and [Va84] at his disposal. Our proof here, however, captures the essence of his approach.)

4. Hybrid Automata

Looking carefully at the decision procedure outlined in the previous section, we see that the most expensive (computationally) part is that of determinizing and complementing the sa B_1. Even if we could test emptiness of Rabin ta in polynomial time, the time complexity would still be doubly exponential, just because of the sheer size of B_2. Thus to get a better bound, we have to eliminate that step of the algorithm. This motivates the following definition.

A *hybrid* ta H is a pair (A,B), where A is a Rabin ta and B is a Buchi sa, both over the same alphabet Σ. H *accepts* a tree h if T is accepted by A and, for every infinite path P starting at λ, B rejects the infinite word $h(P)$. Note that hybrid ta are expressively

equivalent to Rabin ta. By determinizing and complementing B and than combining it with A, we can get a Rabin ta equivalent to H. This construction, however, is doubly exponential. On the other hand, every Rabin ta A is trivially equivalent to the hybrid ta (A,B), where B is a Buchi sa that does not accept any infinite word. Thus, hybrid ta seems to be doubly exponential more powerful than Rabin ta.

The following lower bound for emptiness of hybrid ta is proved by a reduction from alternating polynomial-space Turing machines.

Theorem 4.1. The emptiness problem for hybrid ta is logspace hard for deterministic exponential time. ■

For Rabin automata, the only known lower bound is hardness for PTIME [VW84].

Now we are ready to reduce satisfiability of *delta − DPDL* to emptiness of hybrid ta. Consider the automata A_1 and B_1 described in the previous section. $H_f=(A_1,B_1)$, with A_1 viewed as a Rabin ta, is a hybrid ta that accepts precisely the Hintikka trees of f. A_1 checks for the first five conditions, and rejection of all paths by B_1 ensures that the sixth condition is satisfied.

Clearly, the reduction to hybrid ta is much easier than the reduction to Rabin automata. This would be of no use, however, if testing emptiness of hybrid ta were doubly exponentially harder than testing emptiness of Rabin ta. The crux of our approach is that testing emptiness of hybrid ta is only exponentially harder than testing emptiness of Rabin ta. The basic idea is that infinite trees can be generated by finite objects.

Consider k-ary trees over the alphabet Σ. A (Σ,k)-*generator* is a tuple $G=(V,v_0,\alpha,\beta)$, where V is a set of nodes, $v_0\in V$ is a starting node, $\alpha:V\to\Sigma$ labels every node by a letter from Σ, and $\beta:V\to V^k$ labels every node by an k-tuples of nodes. Intuitively, G generates a tree h if G can be unravelled to h, where $\beta(v)$ is viewed as the tuple of the successors of v. Formally, G *generates* a k-ary tree $h:[k]^*\to\Sigma$ if there is a mapping $\xi:[k]^*\to V$ such that:

- $\xi(\lambda)=v_0$,
- $\alpha(\xi(x))=h(x)$, for every $x\in[k]^*$, and
- $\beta(\xi(x))=\langle\xi(x1),\ldots,\xi(xk)\rangle$, for every $x\in[k]^*$.

The *size* of G is the cardinality of V.

Let $H=(A,B)$ be a hybrid ta. We want to show that if some tree is accepted by H, then H accepts a tree h such that both h and the accepting run r of A on h are generated by a "small" generator, and furthermore, there is a "small" generator that generates both h and r simultaneously.

We need some technical notation. Let $X_1,\ldots,X_k$ be a sequence of sets. $\pi_i:X_1\times\cdots\times X_k\to X_i$ is the projection function, i.e., $\pi_i(\langle x_1,\ldots,x_k\rangle)=x_i$. Let $\gamma:X\to X_1\times\cdots\times X_k$ be some mapping. Then $\gamma_i:X\to X_i$ is defined by: $\gamma_i(x)=\pi_i(\gamma(x))$.

Theorem 4.2. Let $\tau=(\Sigma,S,\rho,S_0)$ be a sequential table with $|S|=n$, let $\theta=(\Sigma,T,\psi,T_0)$ be a k-ary tree table with $|T|=m$, and let $H=(A,B)$ be a hybrid ta, where $A=(\theta,F)$ is a Rabin ta, and $B=(\tau,F)$ is a Buchi sa. If H accept some tree, then there is a $(\Sigma\times T,k)$-generator G of size $O(mc^{n(n+1)})$ that generates a tree $h:[k]^*\to\Sigma\times T$ such that h_1 is accepted by H and h_2 is an accepting run of A on h_1.

Proof Sketch. We first "beef up" A so that its runs carry also information about possible runs of B on paths of the tree. This is done by applying two *subset constructions* to B. The first is the classical subset construction as in [RS59], whose complexity is 2^n, and the second is a generalized subset construction from [SVW84], whose complexity is 4^{n^2}.

The result of applying the subset construction to B is the sequential table $\bar{\tau}=(\Sigma,\bar{S},\bar{\rho},\{S_0\})$. That is, $\bar{S}=2^S$ and $\bar{\rho}:\bar{S}\times\Sigma\to\bar{S}$ is defined by

$$\bar{\rho}(X,a)=\{t : t\in\rho(s,a) \text{ for some } s\in X\}.$$

The result of applying the generalized subset construction to B is the sequential table $\tilde{\tau}=(\Sigma,\tilde{S},\tilde{\rho},\{p_0\})$. Let $S=\{s_1,\ldots,s_n\}$. Define $S'=S\times\{0,1\}$ and $\underline{S}=(2^{S'})^n$. Intuitively, a state in $\underline{S}$ is an n-tuple of sets of states of S labeled by 0 or 1. We need an n-tuple of sets rather then a single set, because we are trying to capture information about runs that can start in any state of S. The label on the state (0 or 1) indicates whether the run contains a state in F. The state set of $\tilde{\tau}$ is $\tilde{S}=S\cup\{p_0\}$, i.e., we add to S a special starting state p_0.

The transition function $\tilde{\rho}:\tilde{S}\times\Sigma\to\underline{S}$ is defined as follows:

- $\langle X_1,\ldots,X_n\rangle=\tilde{\rho}(p_0,a)$ iff $X_i=\{\langle u,0\rangle : u\in\rho(s_i,a)\}\cup\{\langle u,1\rangle : u\in\rho(s_i,a)\cap F\}$.
- $\langle X_1,\ldots,X_n\rangle=\tilde{\rho}(\langle Y_1,\ldots,Y_n\rangle,a)$ iff

$$X_i=\{\langle u,0\rangle : u\in\rho(v,a) \text{ for some } \langle v,j\rangle\in Y_j\}\cup$$
$$\{\langle u,1\rangle : u\in\rho(v,a) \text{ for some } \langle v,1\rangle\in Y_j\}\cup$$

$$\{\langle u,1\rangle : u \in \rho(v,a) \cap F \text{ for some } \langle v,j\rangle \in Y_i\}.$$

Let S denote $\tilde{S} \times \tilde{S}$. Clearly $|S| = O(c^{n(n+1)})$.

We now make, without loss of generality, an assumption about A. We assume that there is a state $d \in T$ (d for degenerate) such that $(\emptyset,\{d\}) \in F$ and for all $a \in \Sigma$ we have that $\psi(d,a) = \{\langle d,d,\ldots,d\rangle\}$. The role of d will become clear later.

Let $\bar{A} = (\bar{\theta},F)$ be a Rabin ta, where $\bar{\theta} = (\Sigma,\bar{T},\bar{\psi},\bar{T}_0)$ is a k-ary tree table. We say that $\bar{A}$ is *augmented* by B if the following holds:

(1) $\bar{T} = T \times S$,

(2) $\bar{T}_0 = T_0 \times \{S_0\} \times \{p_0\}$,

(3) $F = \{(L \times S, U \times S) : (L,U) \in F\}$.

(4) Let $\bar{t} = \langle t,X,p\rangle \in \bar{T}$, let $a \in \Sigma$, let $\bar{u} \in \bar{\psi}(\bar{t},a)$, and let $1 \le i \le k$. Then $\pi_2(\pi_i(\bar{u})) = \tilde{\rho}(X,a)$ and $\pi_3(\pi_i(\bar{u})) = \tilde{\rho}(p,a)$.

(5) Let $\bar{t} \in \bar{T}$ be such that $\pi_1(\bar{t}) = d$, let $a \in \Sigma$, let $\bar{u} \in \bar{\psi}(\bar{t},a)$, and let $1 \le i \le k$. Then $\pi_1(\pi_i(\bar{u})) = d$.

Intuitively, $\bar{A}$ is augmented with B if it carries with it the subset and the generalized subset constructions of B, and though this may affects its transitions it does not affect degenerate transitions and it does not affect the acceptance condition.

The *augmentation* of A by B, denoted A_B, is the Rabin ta $\bar{A} = (\Sigma,\bar{T},\bar{\psi},\bar{T}_0)$, such that the above conditions (1)-(5) are satisfied, and in addition we have

(6) Let $\bar{t} \in \bar{T}$ and $a \in \Sigma$. Then $\bar{u} \in \bar{\psi}(\bar{t},a)$ iff

$$\langle \pi_1(\pi_1(\bar{u})),\ldots,\pi_1(\pi_k(\bar{u}))\rangle \in \psi(\pi_1(\bar{t}),a).$$

In another words, A_B is obtained by taking the direct product of A with $\bar{\tau}$ and $\tilde{\tau}$. Clearly, $|\bar{T}| = O(mc^{n(n+1)})$. The following claim is easily verified.

Claim 1. $H = (A,B)$ accepts some tree iff $\bar{H} = (A_B,B)$ accepts some tree.

Let $h : [k]^* \to \Sigma \times \bar{T}$ be a tree such that h_1 is accepted by $\bar{H}$ and h_2 is an accepting run of A_B on h_1. We call h an *accepting tree-run*. We say that a node $x \in [k]^*$ is *degenerate*, if for all infinite paths $P = x_0x_1,\cdots$ starting at x, there is some i such that $\pi_1(h_2(x_i)) = d$. A state $\bar{t} \in \bar{T}$ is *contingently degenerate* if there is a degenerate node x such that $h_2(x) = \bar{t}$. $\bar{t}$ is *degenerate* if it is contingently degenerate and whenever x is a node such that $h_2(x) = \bar{t}$ then x is a degenerate node.

Claim 2. If $\bar{H}$ has an accepting tree-run, then it has an accepting tree-run such that all contingently degenerate states are degenerate.

We now prove that if $\bar{H} = (\bar{A},B)$ is a hybrid ta such that $\bar{A}$ is augmented by B and $\bar{H}$ has an accepting tree-run, then this tree-run is generated by a generator whose size is linear in the number of states of $\bar{A}$. The proof is by induction on the number of nondegenerate states in A_B. The argument is quite involved and uses Ramsey's Theorem to analyze infinite paths on the tree-run. Details will be given in the full paper. ■

We now use Theorem 4.2 to solve the emptiness problem for hybrid automata.

Theorem 4.3. Given a hybrid automata $H = (A,B)$, we can test whether H accepts some tree in nondeterministic time that is polynomial in the size of A and exponential† in the size of B.

Proof. By Theorem 4.2, if H accept some tree then there is a generator G, whose size is linear in the size of A and exponential in the size of B, that generates a tree accepted by H and a run of A on that tree. The algorithm for testing emptiness consists of nondeterministically guessing a generator, and then checking that it generates a tree accepted by H and a run of A on that tree. We now show that this can be checked in time polynomial in the size of the generator.

Let $B = (\tau,F)$, where $\tau = (\Sigma,S,\rho,S_0)$. Let $G = (V,v_0,\alpha,\beta)$ be a $(\Sigma \times T,k)$-generator. We first check that all paths in the tree generated by $G_1 = (V,v_0,\pi_1(\alpha),\beta)$ are rejected by B. We construct a Buchi ta B', whose size is polynomial in the size of B and G, such that all paths in the tree generated by G_1 are rejected by B iff B' accepts no infinite word. By Theorem 2.1.1, this can be checked in time polynomial in the size of B'.

Now we have to check that the run generated by $G_2 = (V,v_0,\pi_2(\alpha),\beta)$ is an accepting run of A on the tree generated by G_1. By a technique similar to the one used in the first part of the proof, we reduce it polynomially to the emptiness problem of a *Streett* sa, which is shown to be solvable in polynomial time in [EL85]. (We define Streett sa in the full paper.) ■

Corollary 4.4. The emptiness problem for Rabin automata is in NP. ■

† By "exponential" we mean $O(2^{p(n)})$ for some polynomial p.

The corollary, independently proven by Emerson [Em85], improves the exponential time upper bound of [Ra72].

We now apply our algorithm for testing emptiness of hybrid ta to satisfiability of *delta − DPDL*.

Theorem 4.5. The satisfiability problem for *delta − DPDL* (and hence for *delta − PDL*) is solvable in nondeterministic exponential time‡.

Proof. Given a formula f we construct a hybrid ta $H_f=(A,B)$, where the size of A is exponential in the length of f, and the size of B is linear in the length of f, such that H_f accepts precisely the Hintikka trees for f. Thus f is satisfiable iff H_f accepts some tree. The claim now follows by Theorem 4.3. ■

Our technique can be combined with the techniques of Emerson and Street ([ESt84]), to establish a nondeterministic exponential time upper bound for Kozen's μ-calculus ([Ko83]) and Parikh's game logic ([Pa83]). (Emerson and Street established a triply exponential time upper bound for the μ-calculus.) Finally, our technique also yields a nondeterministic exponential time upper bound for *delta − converse − PDL*. Here the hard part is reducing satisfiability to emptiness of hybrid automata, because of the presence of the *converse* construct (which intuitively means to run a program backwards). The reduction is described in [Va85]. The previous upper bound for *delta − converse − PDL* was *octuply* exponential time [St82].

5. Temporal and Process Logics

In dynamic logic computations are described explicitly by means of regular expressions over atomic programs and tests. In contrast, in *temporal logic* ([Pn81]) computations are described implicitly by means of their ongoing behavior. This behavior is captured by temporal formulas such as $X\ f$, meaning "f is true in the next state of the computation", and $f\ U\ g$, meaning "there is a state in the computation in which g is true, and f is true in all states that precede that state". *CTL** is a *branching time* logic introduced by Emerson and Halpern [EH83]. *CTL* formulas are built from assertions of the form $E\ f$, meaning "there exists a computation that satisfies the temporal formula f". We now give a formal definition of *CTL** and its variant CTL^*_f.

‡ By "exponential" we mean $O(2^{p(n)})$ for some polynomial p.

Formulas of *CTL** are built from a set *Prop* of atomic propositions. The sets of *path* formulas and *state* formulas are defined inductively as follows:

- Every atomic proposition $p \in Prop$ is a state formula.
- If f_1 and f_2 are state formulas, then $\neg f_1$ and $f_1 \wedge f_2$ are state formulas.
- Every state formula f is a path formula.
- If f_1 and f_2 are path formulas, then $\neg f_1$ and $f_1 \wedge f_2$ are path formulas.
- If f_1 and f_2 are path formulas, then $X\ f_1$ and $f_1\ U\ f_2$ are path formulas.
- If f is a path formula, then $E\ f$ is a state formula.

*CTL** formulas are interpreted over structures $M=(W,R,\Pi)$ where W is a set of states, $R \subseteq W \times W$ is a binary relation on W, and $\Pi: W \to 2^{Prop}$ assigns truth values to the propositions in *Prop* for each state in W. A *path* x in M starting in u is an infinite sequence $x_0, x_1, \cdots$ such that $x_0 = u$ and $(x_i, x_{i+1}) \in R$ for all $i \geq 0$. x^i is the suffix path $x_i, x_{i+1}, \cdots$. We now define satisfaction for state and path formulas inductively:

- For a proposition $p \in Prop$, $M,u \models p$ iff $p \in \Pi(u)$.
- For state formulas f_1, f_2, $M,u \models f_1 \wedge f_2$ iff $M,u \models f_1$ and $M,u \models f_2$, and $M,u \models \neg f_1$ iff not $M,u \models f_1$.
- For a state formula f, $M,\mathrm{x} \models f$ iff $M,x_0 \models f$.
- For path formulas f_1, f_2, $M,\mathrm{x} \models f_1 \wedge f_2$ iff $M,\mathrm{x} \models f_1$ and $M,u \models f_2$, and $M,\mathrm{x} \models \neg f_1$ iff not $M,\mathrm{x} \models f_1$.
- For path formulas f_1, f_2, $M,\mathrm{x} \models X\ f_1$ iff $M,x^1 \models f_1$, and $M,\mathrm{x} \models f_1\ U\ f_2$ iff for some $i \geq 0$ we have that $M,\mathrm{x}^i \models f_2$ and $M,\mathrm{x}^j \models f_1$ for $0 \leq j \leq i-1$.
- For a path formula f, $M,u \models E\ f$ iff there exists a path x starting at u such that $M,\mathrm{x} \models f$.

In *CTL** all computation paths are infinite. If we restrict attention to finite computation paths, we get a different logic, which we denote CTL^*_f. CTL^*_f has the same syntax and is interpreted over the same structures as *CTL**. The only difference in the semantics is that we consider finite paths rather than infinite paths. A *finite* path x in M starting in u is a finite sequence $x_0, \ldots, x_k$ such that $x_0 = u$ and $(x_{i-1}, x_i) \in R$ for all $1 \leq i \leq k$. x^i is the suffix finite path $x_i, \ldots, x_k$. If $i > k$,

then x^i is the empty path. We need to modify some of the clauses in the definition of satisfaction, to take empty paths into account:

- For a state formula f, $M,x \models f$ iff x is nonempty and $M,x_0 \models f$.
- For a path formula f, $M,x \models X f$ iff x^1 is nonempty and $M,x^1 \models f$.

Process logics combine the features of both dynamic logic and temporal logic, that is, computations are described both explicitly, by regular expressions, and implicitly, by temporal formulas [Ni80,HKP82]. For example, *YAPL* is obtained by combining CTL^* with *delta* $-$ *PDL* [VW83]. We will not describe these logics in detail; the reader is referred to [HKP84, Ni80, VW83].

It turns out that the ability to describe computations implicitly does not add any expressive power to the logic [SPH84]. In particular, *YAPL* was shown to be expressively equivalent to *delta* $-$ *PDL* [VW83]. Thus we can test satisfiability of *YAPL* formula, by translation them to equivalent *delta* $-$ *PDL* formulas, and then applying the decision procedure of the preceding section.

Theorem 5.1. The satisfiability problem for *YAPL* (and hence for CTL^*) is solvable in nondeterministic doubly exponential time†.

Proof Sketch. Vardi and Wolper [VW83] considered a version of *delta* $-$ *PDL* in which programs are described by automata on finite words rather than by regular expressions, and they have shown that there is an exponential translation of *YAPL* to this version of *delta* $-$ *PDL*. It can be shown that the technique of the previous section yields a nondeterministic exponential time upper bound for this version of *delta* $-$ *PDL*. The claim follows. ■

The previously known time upper bounds were triply exponential for CTL^* [ESi84] and quadruply exponential for *YAPL* [VW83]. A nondeterministic doubly exponential upper bound for CTL^* was independently shown by Emerson [Em85]. His proof is different from ours, and it does not extend to *YAPL*.

It may seem that we could do better by directly reducing satisfiability of *YAPL* to emptiness of hybrid automata. Indeed, we can define Hintikka trees for *YAPL* formulas with conditions analogous to the conditions in Section 3. Given a *YAPL* formula f, we can now construct a hybrid ta $H_f=(A,B)$ that accepts precisely the Hintikka trees for f. While the size of A is, as before, exponential in the length of f, the size of B is *exponential* in the length of f, rather than *linear* in the length of f, as is the case with *delta* $-$ *PDL* formulas. Thus testing satisfiability of *YAPL* formulas by directly reducing it to emptiness of hybrid ta still takes nondeterministic doubly exponential time, since the nondeterministic time complexity for testing emptiness of $H_f=(A,B)$ is polynomial in the size of A but exponential in the size of B.

Nevertheless, the next theorem shows that our upper bound is not too far removed from the optimal.

Theorem 5.2. The satisfiability problem for CTL^* (and hence for *YAPL*) and for CTL_f^* is logspace hard for *deterministic* doubly exponential time.

Proof. We first sketch the proof for CTL_f^* and then sketch how to modify the proof for CTL^*. Recall that deterministic doubly exponential time is exactly the class of languages accepted by alternating Turing machines [CKS81] with space bound 2^{cn} for constant $c>0$. Let Z be such an alternating TM and let w be an input to Z of length n. The proof is done by constructing a CTL_f^* formula f_w of length polynomial in n such that Z accepts w iff f_w is satisfiable. We describe informally what the parts of f_w should express about the model. Translation into CTL_f formulas is straightforward. For example, in several places we need a path formula which is true of the finite path P iff the formula g is true in the last state of P. This is expressed by the path formula

$$\textbf{true}\ U\ (g \wedge (\neg X\ \textbf{true})).$$

Similarly, we can talk about the next-to-the-last state of P, etc.

As in the lower bound proved for *PDL* by Fischer and Ladner [FL79], satisfying models of f_w correspond to accepting computation trees of Z on input w. However, we cannot represent a configuration of Z by a state in a Kripke model as in [FL79], since in our case configurations are of length 2^{cn}. Instead, we represent the contents of a single tape cell by a state in a model, a configuration is represented by a path of 2^{cn} states, and computations are obtained by concatenating configurations. The main difficulty is in checking that each configuration in a computation follows legally from

† By "doubly exponential" we mean $O(2^{2^{cn}})$ for some constant $c>0$.

the preceding one. To illustrate how this is done in this sketch, suppose that Z is a nondeterministic TM with space bound 2^{cn}.

The formula contains propositional variables $S_1,...,S_d$ where the constant d is chosen so that these variables can encode symbols of $\Gamma \cup (Q \times \Gamma)$ where Q (Γ) is the set of states (tape symbols) of Z. The string $\alpha(q,\gamma)\beta$ where $\alpha,\beta \in \Gamma^*$, $\gamma \in \Gamma$, and $q \in Q$ represents the configuration where $\alpha\gamma\beta$ is written on the tape and Z is in state q scanning γ. The variables $C_1,...,C_{cn}$ represent a counter where the truth values of these variables at a state give the binary digits of the value of the counter at that state. Given a model $M=(W,R,\Pi)$ and a state $u \in W$, we associate $S(u)$, the symbol encoded by $S_1,...,S_d$ at state u, and $C(u)$, the integer between 0 and $2^{cn}-1$ represented by the counter at u. The formula f_w is a conjunction of subformulas which express the following about the model.

(1) $C(u)=0$ in the root u of the model.

(2) The counter counts correctly, that is for all $u,u' \in W$, if $(u,u') \in R$, then $C(u')=C(u)+1 (\mathrm{mod}\ 2^{cn})$.

(3) The computation starts with the initial configuration. There exists a path $u_0,u_1,...,u_p$ starting at the root u_0 such that $C(u_0)=0, C(u_p)=0, C(u_i)\neq 0$ for $0<i<p$, and $S(u_0)S(u_1)...S(u_{p-1})=(q_0,w_1)w_2 \cdots w_n \# \cdots \#$. where q_0 is the initial state of Z and $\#$ is the blank symbol.

(4) There is a state $u \in W$ with $S(u)=(q_a,\gamma)$ where q_a is the accepting state of Z.

(5) Each configuration follows legally from the preceding one. We write a path formula g which must hold for every path $u_0,...,u_p,u_{p+1},u_{p+2}$ starting at every state $u_0 \in W$. g is a disjunction of four subformulas:

  (5.1) there is no i, $0 \leq i < p$, with $C(u_i)=0$,

  (5.2) there are $0 \leq i < j < p$ with $C(u_i)=C(u_j)=0$,

  (5.3) there is a k, $1 \leq k \leq cn$, with $C_k(u_1) \neq C_k(u_p)$,

  (5.4) the symbols $S(u_p),S(u_{p+1}),S(u_{p+2})$ follow legally from $S(u_0),S(u_1),S(u_2)$ according to the transition rules of Z.

To see that g works, note that the path falsifies (5.1), (5.2) and (5.3) iff u_0 and u_p represent the same tape cell in two adjacent configurations of the computation, since $C(u_0)=C(u_p)$ and the counter equals zero exactly once on the subpath $u_0,...,u_{p-1}$. For each such path, (5.4) makes a local check in two adjacent configurations of the computation. Since g must hold for all paths from every state, (5) checks that the entire computation is legal.

If Z is an alternating TM, we add more subformulas as in [FL79] to force the computation (i.e., the model) to branch after every universal configuration into the two possible next configurations. This completes the proof sketch for CTL_f^*.

A modification is needed for CTL^*. Since all paths are infinite, we cannot talk about the last state on a path, which is needed for (5). The trick (which is apparently known in the folklore) is to introduce a new propositional variable I. We add formulas which require that I is true at the root of the model, if I is true at state u then there are states u_1 and u_2 with $(u,u_1),(u,u_2) \in R$ such that I is true at u_1 and false at u_2, and if I is false at state u then I is false at all R-successors of u. The portion of the model on which I is true is used to represent the accepting computation tree. A finite path P from state u to state v in the computation tree is represented by the infinite path P' which starts at u, coincides with P until state v, and then leaves along a path on which I is identically false. We can talk about the last state of P since it is the last state of P' where I is true.

The final version of the paper will contain a more detailed description of the proof. ■

We note that the above lower bound is the first superexponential time lower bound to be proven for a propositional program logic.

Just as CTL_f^* is the finite version of CTL^*, $YAPL_f$ is the finite version of $YAPL$ [VW83]. A deterministic doubly exponential decision procedure for $YAPL_f$ (and hence for CTL_f^*) was described in [VW83]. Combining that result with Theorem 5.2, we get:

Theorem 5.3. The satisfiability problem for CTL_f^* and $YAPL_f$ is logspace complete for deterministic doubly exponential time. ■

6. Concluding Remarks

Using new automata-theoretic techniques and new reduction techniques, we have obtained better upper and lower time bounds for a whole variety of program logics. For some logics, the bounds are tight, while for other

logics the gap between the lower and the upper bounds is narrowed to the gap between deterministic and nondeterministic time.

The observant readers may have asked themselves whether that gap is not due to our use of overly powerful automata. Let us define *weak hybrid ta* as pairs (A,B), where A is a Buchi ta (rather than a Rabin ta) and B is a Buchi sa, with acceptance defined analogously to acceptance by hybrid ta. A careful study of our reduction of satisfiability to emptiness of hybrid ta shows that we could have reduced satisfiability to emptiness of weak hybrid ta. If we could test emptiness of weak hybrid ta in a more efficient way (exponential time is a lower bound), then we could get a better upper bound for satisfiability. Unfortunately, we do not know of such a better algorithm for emptiness of weak hybrid ta.

Acknowledgements. Part of the research reported here was done while the first author was visiting the IBM Research Lab at San Jose. Research at Stanford University was supported by a gift from System Development Foundation. We are grateful to Allen Emerson for inspiring discussions. We'd also like to thank Ron Fagin and Joe Halpern for helpful comments on a previous draft of this paper.

References

[BHP82] M. Ben-Ari, J.Y. Halpern, A. Pnueli, "Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness", *J. Computer and System Science*, 25(1982), pp. 402-417.

[BMP81] 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.

[Bu62] J.R. Buchi, "On a Decision Method in Restricted Second Order Arithmetic", *Proc. Int'l Congr. Logic, Method and Phil. Sci. 1960*, Stanford University Press, 1962, pp. 1-12.

[Ch74] Y. Choueka, "Theories of Automata on ω-Tapes: A Simplified Approach", *J. Computer and System Sciences*, 8 (1974), pp. 117-141.

[CKS81] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, "Alternation", *J. ACM* 28 (1981), pp. 114-133.

[EH82] E.A. Emerson, J.Y. Halpern,"Decision Procedures and Expressiveness in the Temporal Logic of Branching Time", *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, 1982, pp. 169-180.

[EH83] E.A. Emerson, J.Y. Halpern,""Sometimes" and "Not Never" Revisited: On Branching vs. Linear Time", *Proc. 10th ACM Symp. on Principles of Programming Languages*, 1983.

[EL85] E.A. Emerson, C.L. Lei, "Modalities for Modal Checking: Branching Time Strikes Back", to appear in *Proc. 10th ACM Symp. on Principles of Programming Languages*, 1985.

[Em85] E.A. Emerson, "Automata, tableaux, and temporal logics", to appear in *Proc. Workshop on Logics of Programs*, Brooklyn, June 1985.

[ESi84] E.A. Emerson, A.P. Sistla,"Deciding Branching Time Logic", *Proc. 16th ACM Symp. on Theory of Computing*, Washington, 1984, pp. 14-24.

[ESt84] E. A. Emerson, A. P. Streett, "An Elementary Decision Procedure for the μ-calculus", *Proc. 11th Int. Colloq. on Automata, Languages and Programming*, 1984, Lecture Notes in Computer Science, Springer-Verlag.

[FL79] M.J. Fisher, R.E. Ladner, "Propositional Dynamic Logic of Regular Programs", *J. Computer and System Sciences*, 18(2), 1979, pp. 194-211.

[Ga76] D.M., Gabbay, *"Investigation in Modal and Tense Logic"*, Reidel, 1976.

[Ha83] Z. Habasinski, *"Decidability Problems in Logics of Programs"*, Ph.D. Dissertation, Dept. of Mathematics, Technical University of Poznan, 1983.

[HKP82] D. Harel, D. Kozen, R. Parikh, "Process Logic: Expressiveness, Decidability, Completeness", *Journal of Computer and System Science* 25, 2 (1982), pp. 144-170.

[HR72] 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.

[HS83] D. Harel, R. Sherman, "Looping vs. Repeating in Dynamic Logic", *Information and Control* 55(1982), pp. 175-192.

[Ko83] D. Kozen, "Results on the Propositional μ-Calculus", *Theoretical Computer Science*, 27(1983), pp. 333-354.

[McN66] R. McNaughton, "Testing and Generating Infinite Sequences by a Finite Automaton", *Information and Control* 9 (1966), pp. 521-530

[Me75] 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.

[Ni80] H. Nishimura, "Descriptively Complete Process Logic", *Acta Informatica*, 14 (1980), pp. 359-369.

[Pa78] R. Parikh, "A Decidability Result for a Second Order Process Logic", *Proceedings 19th IEEE Symposium on Foundations of Computer Science*, Ann Arbor, October 1978.

[Pa80] R. Parikh, "Propositional Logics of Programs: Systems, Models and Complexity", *Proc. 7th ACM Symp. on Principles of Programming Languages*, Las Vegas, 1980, pp. 186-192.

[Pa83] R. Parikh, "Propositional Game Logic", *Proc. 25 IEEE Symp. on Foundations of Computer Science*, Tuscon, 1983, pp. 195-200.

[Pn81] A. Pnueli, "The Temporal Logic of Concurrent Programs", *Theoretical Computer Science* 13(1981), pp. 45-60.

[PS83] A. Pnueli, R. Sherman, *"Propositional Dynamic Logic of Looping Flowcharts"*, Technical Report, Weizmann Institute, Rehovot, Israel, 1983.

[Pr76] V.R. Pratt, "Semantical Considerations on Floyd-Hoare Logic", *Proc. 17th IEEE Symp. on Foundations of Computer Science*, Houston, 1976, pp. 109-121.

[Ra69] M.O. Rabin, "Decidability of Second Order Theories and Automata on Infinite Trees", *Trans. AMS*, 141(1969), pp. 1-35.

[Ra70] 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.

[Ra72] M.O. Rabin, "Automata on Infinite Objects and Church's Problem", *Proc. Regional AMS Conf. Series in Math.* 13(1972), pp. 1-22.

[RS59] M. O. Rabin, D. Scott, "Finite Automata and their Decision Problems", *IBM J. Res. & Dev.*, 3(2), 1959, pp 114-125.

[Sh84] R. Sherman, *"Variants of Propositional Dynamic Logic,"* Ph.D. Dissertation, The Weizmann Inst. of Science, 1984.

[SPH84] R. Sherman, A. Pnueli, D. Harel, "Is the Interesting Part of Process Logic Uninteresting?", *SIAM J. Computing* 13(1984), pp. 825-839.

[St80] R.S. Streett, *"A propositional Dynamic Logic for Reasoning about Program Divergence"*, M.Sc. Thesis, MIT, 1980.

[St82] R.S. Streett, "Propositional Dynamic Logic of Looping and Converse is elementarily decidable", *Information and Control* 54(1982), pp. 121-141.

[SVW85] A.P. Sistla, M.Y. Vardi, P. Wolper, "The Complementation Problem for Buchi Automata with Applications to Temporal Logic", to appear in *Proc. 12th Int. Colloq. on Automata, Languages and Programming*, 1985.

[Va84] M.Y. Vardi, *"On deterministic ω-automata"*, to appear.

[Va85] M.Y. Vardi, "The Taming of Converse: Reasoning about Two-Way Computations", to appear in *Proc. Workshop on Logics of Programs*, Brooklyn, June 1985.

[VW83] M.Y. Vardi, P. Wolper, "Yet another process logic", *Proc. Workshop on Logics of Programs 1983*, Springer-Verlag, Lecture Notes in Computer Science - vol. 164 (Clarke and Kozen, eds.), pp. 501-512.

[VW84] 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.