The Complexity of Tree Automata and Logics of Programs[†] (Extended Abstract)

E. Allen EMERSON[1,2] and *Charanjit S. JUTLA*[1]

1. Department of Computer Sciences, The University of Texas at Austin, USA

2. Mathematics and Computing Science Department, Technical University of Eindhoven, The Netherlands

Abstract: We investigate the (computational) complexity of testing nonemptiness of finite state automata on infinite trees. We show that for tree automata with m states and n pairs nonemptiness can be tested in time $O((mn)^{3n})$, even though we show that the problem is in general NP-complete. The new nonemptiness algorithm enables us to obtain exponentially improved, essentially tight upper bounds for numerous important modal logics of programs, interpreted with the usual semantics over structures generated by binary relations. For example, we show that satisfiability for the full branching time logic CTL* can be tested in deterministic double exponential time. It also follows that satisfiability for Propositional Dynamic Logic with a repetition construct (PDL-delta) and for the Propositional Mu-calculus ($L\mu$) can be tested in deterministic single exponential time.

1. Introduction

There has been a resurgence of interest in automata on infinite objects, due to their intimate relation with temporal and modal logics of programs. They provide an important and uniform approach to the development of decision procedures for testing satisfiability of the propositional versions of these logics ([VW84], [St81]). Such logics and their corresponding decision procedures are not only of inherent mathematical interest, but are also potentially useful in the specification, verification, and synthesis of concurrent programs (cf. [Pn77], [EC82], [MW84], [LPZ85]).

In the case of branching time temporal logic, the standard paradigm nowadays for testing satisfiability is the reduction to the non-emptiness problem for finite state automata on infinite trees; i.e., one builds a tree automaton which accepts essentially all models of the candidate formula and then tests nonemptiness of the tree automaton. Thus in order to improve the complexity there are two issues: (1) the size of the tree automaton, and (2) the complexity of testing non-emptiness of the tree automaton.

[†]This work was supported in part by NSF grant DCR-8511354, ONR URI contract N00014-86-K-0763, and a grant from the Netherlands NWO.

In this paper we obtain new, improved and essentially tight bounds on testing nonemptiness of tree automata that allow us to close an exponential gap which has existed between the Upper and Lower bounds of the satisfiability problem of numerous important Modal Logics of Programs. These logics include CTL^*- the full Branching Time Logic [EH83], PDL-delta - the Propositional Dynamic Logic with infinite repetition construct [St81], and the propositional Mu-calculus ($L\mu$) - a language for characterizing temporal correctness properties in terms of extremal fixpoints of predicate transformers [Ko83].

To obtain these improvements, we focus on the complexity of testing nonemptiness of tree automata. We first note, however, that the size of an automaton has two parameters: the number of states in the automaton's transition diagram and the number of pairs in its acceptance condition. We next make the following important observation: for most logics of programs, the number of pairs is logarithmic in the number of states.

We go on to analyze the complexity of testing nonemptiness of (Rabin) Tree Automata [Ra69] and show that it is NP-complete. However, a multi-parameter analysis shows that there is an algorithm that runs in time $O((mn)^{3n})$ which is polynomial in the size m of the automaton's transition diagram and exponential in the number n of pairs in its acceptance condition. The algorithm is based on a type of "pseudo model checking" for some restricted kinds of Mu-calculus formulae, essentially those constructed in terms of the simple correctness properties such as inevitability of P (along all paths eventually P) and invariance of P (along all paths always P). Moreover since the problem is NP-complete, it is unlikely to have a better algorithm which is polynomial in the size of both parameters. The previous best known algorithm was in NP ([Em85], [VS85]).

0272-5428/88/0000/0328$01.00 © 1988 IEEE

The above nonemptiness algorithm now permits us to obtain a deterministic double exponential time decision procedure for CTL^*, by using the reduction from CTL^* to Tree Automata obtained in [ESi84], in which the size of the automaton is double exponential in the length of the formula and the number of pairs are only exponential in the length of the formula. The bound follows by simple arithmetic, since a double exponential raised to a single exponential power is still a double exponential.

This amounts to an exponential improvement over the best previously known algorithm which was in nondeterministic double exponential time ([Em85], [VS85]) i.e., three exponentials when determinized. It is also essentially tight since CTL^* was shown to be double-exponential time hard [VS85]; thus CTL^* is deterministic double exponential time complete.

Using the recent single exponential general McNaughton [Mc66] construction of Safra [Sa88] (i.e., construction for determinizing a Buchi finite automata on infinite strings), our new non-emptiness algorithm also gives us a deterministic single exponential time decision procedure for both PDL-delta and the Mu-Calculus, since the Safra construction allows us to reduce satisfiability of these logics to testing nonemptiness of a tree automaton with exponentially many states and polynomially many pairs. This represents an exponential improvement over the best known deterministic algorithms for these logics, which took deterministic double exponential time, corresponding to the nondeterministic exponential time upper bounds of [VS85]. The bounds are essentially tight also, since the exponential time lower bound follows from that established for ordinary PDL (Propositional Dynamic Logic) by Fischer-Ladner [FL79].

The above results have been obtained using only the classical Rabin tree Automata. We also, however, consider the tree automata of Streett [St81] which were specifically introduced to facilitate formulation of temporal decision procedures. We show that the non-emptiness problem of Streett Automata is co-NP-complete, by reducing the complement of the problem to non-emptiness of the Pairs Automata and vice versa. The reduction employs the fact that Infinite Borel Games are determinate (Martin's Theorem [Ma75]). This reduction also gives an algorithm which is exponential in the number of states and polynomial in the number of pairs. We can thus reestablish the above upper bounds for logics of programs using Streett automata as well.

It is interesting to note that, for most logics (including PDL-delta and the Mu-Calculus but excluding CTL^*), our non-emptiness algorithm(s) and Safra's construction both play a crucial role. Each is independent of the other. Moreover, each is needed and neither alone suffices. Our algorithm improves the complexity of testing non-emptiness by an exponential factor, while Safra's construction independently applies to reduce the size of the automaton by an exponential factor. For example, the "traditional" result of Streett [St81] gave a deterministic triple exponential algorithm for PDL-delta. Our algorithm alone improves it to deterministic double exponential time. Alternatively, Safra's construction alone improves it to deterministic double exponential time. As shown in this paper, the constructions can be applied together to get a cumulative double exponential speedup for PDL-delta. In the case of CTL^*, we already had the effect of Safra's construction, because [ESi84] gave a way to determinize with only a single exponential blowup the Buchi string automata corresponding to linear Temporal Logic formula, using the special structure of such automata (unique accepting run). So Safra's construction provides no help for the complexity of the CTL^* logic. (Actually, in the workshop version of [ESi84] it was stated without proof that Buchi Automata could be determinized, without regard to their special structure, with a single exponential blowup. But it was harder than it appeared, and the insights of Safra's construction are needed.)

The remainder of the paper is organized as follows: in section 2 we give preliminary definitions and terminology. In section 3 we give the main technical results on testing nonemptiness. Theorem 3.1 in particular describes the pseudo-model checking algorithm for Rabin tree automata, while Theorem 3.2 deals with the lower bound. Streett type automata are considered in Theorem 3.3. Applications of the algorithm to testing satisfiability of modal logics of programs including, CTL^*, PDL-delta, and the Mu-calculus are described in section 4.

2. Preliminaries

2.1 Logics of Programs

The *Full Branching Time Logic CTL** [EH83] derives its expressive power from the freedom of combining modalities which quantify over paths and the modalities which quantify states along a particular path. We inductively define a class of state formulae and a class of path formulae (true or false of state and path resp.). Any atomic proposition is a state formula. If p is a path formula then Ep and Ap (along some path and along all paths resp.) are state formulae. Any state formula is also a path formula. If p, q are path formulae then Xp (nexttime), Fp (sometime), Gp (everytime), (p U q) (until) are path formulae. Finally, both path and state formulae are closed under usual boolean connectives. Some of the above connectives can be defined as abbreviations: Ap abbreviates $\neg E \neg p$, Fp abbreviates *true* $U\, p$, and Gp abbreviates $\neg F \neg p$.

The semantics of a formula are defined with respect to an R-generable structure $M = (S, R, L)$, where S is a non-empty set of states, R is a non-empty total binary relation on S, and L is a labeling which assigns to each state a set of atomic propositions true in the state. *CTL** and the other logics we study, have the property that their models (like M above) can be unwound into an infinite tree. (Note: in [ESi84] it was shown that a *CTL** formula of length k is satisfiable iff it has an infinte tree model with finite branching bounded by k, i.e. iff it is satisfiable over a k-ary tree. Our results on tree automata apply to such k-ary trees. We consider only binary trees merely to simplify the exposition).

A ***fullpath*** $(s_1, s_2, ...)$ is an infinite sequence of states such that $(s_i, s_{i+1}) \in R$ for all i. We write $M, s \models p$ ($M, x \models p$) to mean that state formula p (path formula p) is true in structure M at state s (of path x, resp.). When M is understood, we write simply $s \models p$ ($x \models p$). We define $\models$ inductively using the convention that $x = (s_1, s_2, ...)$ denotes a path and x^i denotes the suffix path $(s_i, s_{i+1}, ...)$.
For a state s:

(S1) $s \models P$ iff $P \in L(s)$ for atomic proposition P

(S2) $s \models p \wedge q$ iff $s \models p$ and $s \models q$
$s \models \neg p$ iff not $(s \models p)$

(S3) $s \models Ep$ iff for some fullpath x starting at s, $x \models p$

For a path $x = (s_1, s_2, ...)$:

(P1) $x \models p$ iff $s_1 \models p$, for any state formula p

(P2) $x \models p \wedge q$ iff $x \models p$ and $x \models q$
$x \models \neg p$ iff not $(x \models p)$

(P3) $x \models Xp$ iff $x^2 \models p$
$x \models (p\, U\, q)$ iff for some $i \geq 1$, $x^i \models q$ and for all $j \geq 1 : j \geq i$ implies $x^j \models p$

Thus, AFGQ, a state formula, holds at state s iff along all paths starting at s, eventually always Q holds. AFGQ is also a path formula, and it holds for a path $x = (s_1, s_2, \ldots)$ iff it holds for s_1 as a state formula.

We say that state formula p is ***valid***, and write $\models p$, if for every structure M and every state s in M, $M, s \models p$. We say that state formula p is satisfiable if for some structure M and some state s in M, $M, s \models p$. In this case we also say that M defines a ***model*** of p. We define validity and satisfiability for path formulae similarly.

As opposed to *CTL**, in which the models represent behaviors of the programs, in *PDL-delta* the programs are explicit in the models. So the modalities in PDL-delta (and other process logics) quantify the states reachable by programs explicitly stated in the modality. Thus, for a program B (which is obtained from atomic programs and tests using regular expressions) $< B >$p ($[B]$p) states that there is an execution of B leading to p (after all executions of B p holds). Also included is the infinite repetition construct (which makes it much more expressive than PDL) *delta*(Δ). ΔB states that it is possible to execute B repetitively infinitely many times. PDL-delta formulae are interpreted over structures $M = (S, R, L)$, where S is a set of states, R : $Prog \rightarrow 2^{W \times W}$ is a transition relation, *Prog* stands for the atomic programs, and L is a labeling of S with propositions in *Prop*. For more details see [St81].

A ***least fixpoint*** construct can be used to increase the power of simple Modal Logics. Thus, by adding this construct to PDL, we get the ***Mu-calculus*** [Ko83], a logic which subsumes PDL-delta. The least fixpoint can also be used to increase the power of the simple

subset CTL of CTL^*. This construct has the syntax $\mu X.f(X)$, where $f(X)$ is any formula syntactically monotone in the propositional variable X, i.e. all occurrences of X in $f(X)$ fall under an even no. of negations. It is interpreted as the smallest set S of states such that $S = f(S)$. By the well known Tarski-Knaster Theorem $\mu X.\ f(X) = \bigcup_i\ f^i(\text{false})$ where i ranges over all ordinals and f^i (intuitively) denotes the i-fold composition of f with itself; when the domain is finite we may take i as ranging over just the natural numbers. Its dual, the greatest fixpoint is denoted $\nu X.f(X)$ ($\equiv\ \neg\mu X.\neg f(\neg X)$). Thus, e.g, $\mu X.[A]X$ is equivalent to $\neg\ \Delta\ A$ of PDL-delta. Similarly, using Temporal Logic, $\mu x(P \vee AXx)$ is equivalent to AFP (i.e. along all paths eventually P holds). For more details, see [Ko83],[StE84],[EL86].

2.2 Automata on Infinite Trees

For notational simplicity, we only consider finite automata on infinite binary trees. A *finite automaton* $\mathcal{A}$ on infinite binary Σ-trees consists of a tuple (Σ, S, δ, s_0) plus an acceptance condition (described subsequently) where

Σ is the input alphabet labeling the nodes of the input tree,

S is the set of states of the automaton,

$\delta : S \times \Sigma \rightarrow$ Powerset(S^2) is the non-deterministic transition function, and

$s_0 \in S$ is the start state of the automaton.

A run of $\mathcal{A}$ on the input Σ-tree T is a function $\rho : \{0,1\}^* \rightarrow S$ such that for all $v \in \{0,1\}^*$, $(\rho(v0), \rho(v1)) \in \delta(\rho(v), T(v))$ and $\rho(\lambda) = s_0$. We say that $\mathcal{A}$ accepts input tree T iff there exists a run ρ of $\mathcal{A}$ on T such that for all paths x starting at the root of T if $r = \rho|x$, the sequence of states $\mathcal{A}$ goes through along path x, then the acceptance condition (as below) holds along r.

For a *Pairs automaton* (Rabin [Ra69]) acceptance is defined in terms of a finite list $((RED_1, GREEN_1), \ldots, (RED_k, GREEN_k))$ of pairs of sets of states (think of them as pairs of colored lights where $\mathcal{A}$ flashes the red light of the first pair upon entering any state of the set RED_1, etc.): r satisfies the pairs condition iff there exists a pair $i \in [1..k]$ such that RED_i flashes finitely often and $GREEN_i$ flashes infinitely often. Or in Temporal Logic notation, $A[\bigvee_{i\in[1..k]} (GF\ GREEN_i \wedge \neg GF\ RED_i)]$. Finally, a *Complemented Pairs (*or Streett [St81]*) automaton* is defined by the above condition being false, i.e. for all pairs $i \in [1..k]$, infinitely often $GREEN_i$ flashes implies that RED_i flashes infinitely often too.

The transition diagram of $\mathcal{A}$ can be viewed as an AND/OR-graph where the set S of states of $\mathcal{A}$ comprises the set of OR-nodes, while the AND-nodes define the allowable moves of the automaton. Intuitively, OR-nodes indicate that a non-deterministic choice has to be made (depending on the input label), while the AND-nodes force the automaton along all directions. We merge OR-nodes representing the same set of states of $\mathcal{A}$. Suppose that for $\mathcal{A}$, $\delta(s,a) = \{(t_1, u_1), \ldots, (t_m, u_m)\}$ and $\delta(s,b) = \{(v_1, w_1), \ldots, (v_n, w_n)\}$ then the transition diagram contains the portion shown in Figure 1.

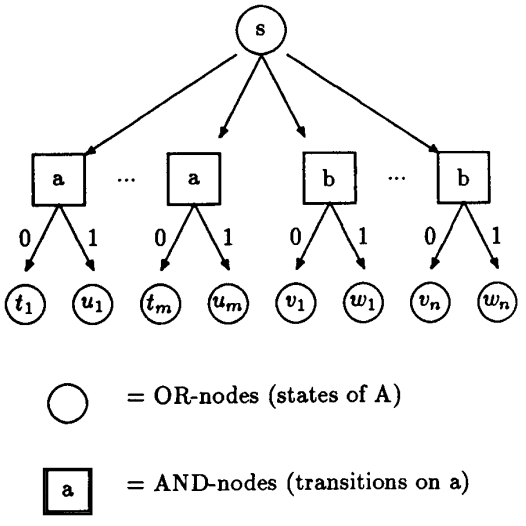


Figure 1

2.3 Small Model Theorem

A Σ-*Labeled binary Directed Graph* $G = (V, A_0, A_1, L)$ consists of V a set of underlying nodes, A_0 a total function $V \rightarrow V$ which assigns to each node v a unique successor node called the 0-successor of v, A_1 similarly defines the unique 1-successor of each node, and L a labeling function $V \rightarrow \Sigma$ which assigns to each node a symbol from Σ. Note that this Labeled Graph can be viewed as a tree generator by unwind-

ing it in the obvious fashion.

A *run* of automaton $\mathcal{A}$ on G is a mapping $\rho : V \rightarrow S$ such that $\forall v \in V$, $(\rho(A_0(v)), \rho(A_1(v))) \in \delta(\rho(v), L(v))$ and $\rho(v_0) = s_0$. Intuitively, a run is a labeling of G with states of $\mathcal{A}$ consistent with the local structure of $\mathcal{A}$'s transition diagram.

As already noted, the transition diagram of $\mathcal{A}$ can be viewed as an AND-OR graph. A *Hintikka Structure M contained in $\mathcal{A}$* is a subgraph of $\mathcal{A}$, such that for each OR-node s in M, s has at most one successor, and for each AND-node t common to M and $\mathcal{A}$, if u is a successor of t in $\mathcal{A}$, then u is also a successor of t in M. Note that in M the OR-nodes become redundant (the choice has been forced), and hence by removing the OR-nodes we can collapse M into a Labeled Graph.

The following *Small Model Theorem* was proved in [Em85].

Theorem: If pairs automaton $\mathcal{A}$ accepts some tree T, then $\mathcal{A}$ accepts some Labeled Graph G with number of nodes linear in the size of $\mathcal{A}$'s transition diagram. In fact, $\mathcal{A}$ accepts a Hintikka Structure *contained* in $\mathcal{A}$.

3. Complexity of Tree Automata

Definitions: Let T be the transition diagram of a tree automaton $\mathcal{A}$. We say, $T, s \| - p$, i.e. in diagram T state s is a *pseudomodel* of p, iff there exists a Hintikka structure M contained in T such that $M, s \models p$.

By the Small Model Theorem, the automaton is non empty iff $A\,\Phi$ is "satisfiable" at the start state of the automaton; in other words, automaton $\mathcal{A}$ is non-empty iff there is a Hintikka Structure M contained in the Transition Diagram T of the automaton such that for the start state s_0 of $\mathcal{A}$: $M, s_0 \models A\,\Phi$, i.e. $T, s_0 \| - A\,\Phi$.

For a formula $p(y)$, $p^0(y) = \mathit{false}$, $p^1(y) = p(y)$ and $p^i(y) = p(p^{i-1}(y))$.

Theorem 3.1: Non-emptiness of a pairs automaton $\mathcal{A}$ having m states and n pairs can be tested in deterministic time $O((mn)^{3n})$.

Proof: Let $\mathcal{A}$ be a tree automaton with transition diagram T of size m states and with acceptance condition $A\,\Phi_\Gamma$ where $\Phi_\Gamma = \bigvee_{\gamma \in \Gamma} (GFQ_\gamma \wedge FGP_\gamma)$ is the pairs condition and Γ is the index set $[0..n-1]$ of pairs. Here, Q_γ and P_γ stand for $GREEN_\gamma$ and $\neg RED_\gamma$ resp. (Note: When Γ is understood from context we can drop it and write just $A\,\Phi$.)

The idea of the algorithm is to inductively compute the set of states in T at which $A\,\Phi$ is satisfiable, using the fixpoint characterization of $A\,\Phi$ in terms of the pairs condition for fewer pairs and the modalities AF and AG given in Lemma 1 below:

Lemma 1: $A\,\Phi_\Gamma = \mu Y.\tau(Y)$ where $\tau(Y) = \bigvee_{\gamma \in \Gamma} AFAG\,((P_\gamma \vee Y) \wedge A(FQ_\gamma \vee \Phi_{\Gamma/\{\gamma\}}))$ □

We thus successively calculate (the set of states where each of the following is satisfiable) $Y^1 = \tau(\mathit{false})$, $Y^2 = \tau(Y^1)$, ..., and inductively $Y^{i+1} = \tau(Y^i)$. By the Tarski-Knaster Theorem $(T, s \| - A\,\Phi_\Gamma) \equiv (T, s \| - \exists i\, Y^i)$, where $i \leq |T|$.

To compute the subformulae of $\tau(Y)$ we also use their fixpoint characterizations. For example, subformulae of the form AFq are computed using the sequence of approximations corresponding to the fixpoint characterization $AFq = \mu Z.(q \vee AXZ)$. $Z^0 =$ false, $Z^1 = q \vee AXZ^0$, $Z^2 = q \vee AXZ^1$, etc. until stabilization. Similarly subformulae of the form AGr are computed using a descending chain of approximations corresponding to $AGr = \nu W.(r \wedge AXW)$: W^0 = true, $W^1 = r \wedge AXW^0$, $W^2 = r \wedge AXW^1$, etc.

If each OR-node in T had a unique successor then the computation of $A\,\Phi$ would simply amount to model checking in the Mu-calculus [EL86]. But in general, each OR-node has more than one successor. Therefore, our algorithm must simultaneously (i) exhaust the search space of all Hintikka Structures contained in T and (ii) check if one of these Hintikka structures is a model of the pairs-condition. We explain below the modifications needed to permit both steps to be done together, therby performing the desired pseudo-model checking.

For each subformula p of the fixpoint characterization of $A\,\Phi$ the algorithm computes a set of states $\mathrm{val}(T, p)$, intended to correspond to those states in T at which p is satisfiable. The set $\mathrm{val}(T, p)$ is calculated recursively, i.e. by induction on structure of subformulae of p. For simple formulae we have the following straightforward rules:

$\mathrm{val}(T, \mathit{false}) = \phi$
$\mathrm{val}(T, q \vee r) = \mathrm{val}(T, q) \cup \mathrm{val}(T, r)$
$\mathrm{val}(T, q \wedge r) = \mathrm{val}(T, q) \cap \mathrm{val}(T, r)$.

For extremal fixpoints the basic principle is calculation by the Tarski-Knaster Theorem; e.g., $\text{val}(T, A\,\Phi_\Gamma) = \bigcup_i \text{val}(T, Y^i)$. However, there are certain subtleties involved in defining the predicate transformers on the right hand side. The lemmas below show that each $\text{val}(T,p)$ is computed correctly in these cases as well.

Lemma 2 below shows how to compute $\{s : T, s \,||\!-\, AFq\}$, where q is a proposition. Intuitively, since T is an AND-OR graph, there is a Hintikka Structure M in T (i.e. we pick unique alternatives from OR-nodes) such that in M along all paths eventually q holds, iff in T along paths chosen by picking one alternative at each OR-node and picking all successors at each AND-node, eventually q holds. Thus we literally do model-checking in T, considered simply as a structure, of a modified formula. For the 'if' part we have to make sure that if the model-checking algorithm visits an OR-node more than once, then the same alternative from that OR-node suffices. This is possible because q is a proposition.

Lemma 2: For q a proposition,
$T, s \,||\!-\, AFq$ iff $T, s \models \mu x.(q \vee EXAXx)$ □

Thus we compute, $\text{val}(T,\ AFQ_\gamma) = \{s : T, s \models \mu x.(Q_\gamma \vee EXAXx)\}$

We now show how to compute $\text{val}(T, A(FQ_\gamma \vee \Phi_{\Gamma/\{\gamma\}}))$. It would be nice if we could compute it as $\text{val}(T, AFQ_\gamma) \cup \text{val}(T, A\Phi_{\Gamma/\{\gamma\}})$, but this doesn't work. However, this naive idea can be made to work by computing $A\Phi_{\Gamma/\{\gamma\}}$ in the subgraph induced by deleting AFQ_γ.

Definition: If U is a subgraph of T and Y is a subset of OR-nodes of U, then U/Y is the subgraph obtained by deleting Y, and all edges incident on Y. Note that it is possible that, in U/Y, there are AND-nodes with no outgoing edges. However, if Y is of the form $Y = \{s : U, s \,||\!-\, AFq\}$ for some q, then each AND-node (which is not isolated) in U/Y will have at least one out-going edge.

Lemma 3: $T, s \,||\!-\, A(FQ_\gamma \vee \Phi_{\Gamma/\{\gamma\}})$ iff
$T, s \,||\!-\, AFQ_\gamma \quad \vee \quad T/\text{AFQ}, s \,||\!-\, A\Phi_{\Gamma/\{\gamma\}}$
where $\text{AFQ} = \{s : T, s \,||\!-\, AFQ_\gamma\}$ □

Thus we compute,$\text{val}(T,\ A(FQ_\gamma \vee \Phi_{\Gamma/\{\gamma\}})) = \text{val}(T, AFQ_\gamma) \cup \text{val}(T/\text{val}(T, AFQ_\gamma),\ A\Phi_{\Gamma/\{\gamma\}})$

Lemma 4 below can be used to show that Y^i implies every subexpression of the fixpoint characterization of $A\,\Phi$ which has Y^i as a subexpression itself. Intuitively, this means that if a tree t satisfies a subexpression q of $A\,\Phi$ which contains Y^i, then if any tree t' satisfying Y^i is grafted anywhere in t, then the new tree thus obtained still satisfies q. This fact is used in Lemma 5.

Lemma 4:
$Y^i \Rightarrow \bigwedge_{\gamma \in \Gamma}\ AG[(P_\gamma \vee Y^i) \wedge A(FQ_\gamma \vee \Phi_{\Gamma/\{\gamma\}})]$ □

Let $g_\gamma(Y)$ be $((P_\gamma \vee Y) \wedge A(FQ_\gamma \vee \Phi_{\Gamma/\{\gamma\}})$. When γ is clear from context, we drop the subscript from g_γ. We now discuss how to compute $AGg(Y)$. The fixpoint characterization of AGr, for any r, is $\nu x.(r \wedge AXx)$. Now, we have to check that at each OR-node $s : r$ holds and there is a choice of an AND-node successor at s such that AXx holds(i.e. at fixpoint each successor of the OR-node satisfies AGr). Moreover, we have to make sure that the choice of successor picked by r at s is the same as the one picked by AXx. The trick is to recompute r in the AND-OR graph induced by the current approximation Z^j, below:

Let $Z^0 = \text{val}(T, g(Y^{i-1}))$ and
$Z^j = \{s : s \in \text{val}(Z^{j-1}, g(\text{val}(T, Y^{i-1}))) \wedge T, s \models EXAX\ Z^{j-1}\}$
Let Z^k be the fixpoint.
Thus we compute, $\text{val}(T,\ AG\ g(Y^{i-1})) = Z^k$

Note: Here, we identify the set of OR-nodes Z^j with the transition diagram obtained from U by deleting all OR-nodes not in Z^j, and by deleting all AND-nodes which do not have all its successors in Z^j (cf. the definition of U/Y).

Note also that in the definition of $\text{val}(T, AGg(Y^i))$, while computing each successive approximation Z^j, we just compute $\text{val}(Z^{j-1},\ g(\text{val}(T,\ Y^{i-1})))$, instead of $\text{val}(Z^{j-1},\ g(\tau^{i-1}(false))$. Computing the latter would cause an exponential blowup.

Lemma 5: $T, s \,||\!-\, AGg(Y^{i-1})$
$\Rightarrow s \in Z^k$
$\Rightarrow T, s \,||\!-\, \exists\, \alpha\ AGg(Y^\alpha)$ □

Finally we compute $\text{val}(T,\ AFAGg(Y^{i-1})) = \{s : T, s \models \mu\, x.(\text{val}(T, AGg(Y^{i-1})) \vee EXAXx)\}$.

This is justified by Lemma 6 which is similar to Lemma

2, except that the ($\Leftarrow$) part of Lemma 2 doesn't hold. Instead of proving that there is a model contained, we prove that there is a model "generated", i.e a model obtained by unwinding T.

Lemma 6: $T, s \Vdash AFAGg(Y^{i-1})$
$\Rightarrow s \in \text{val}(T, AFAGg(Y^{i-1}))$
$\Rightarrow T, s \Vdash \exists\alpha\, AFAGg(Y^{\alpha})$ □

Correctness:
Induction Hypothesis (1) below, claims that for $p = A\,\Phi_\Gamma$, val(T,$A\,\Phi_\Gamma$) computes the set $\{s : T, s \Vdash A\,\Phi_\Gamma\}$. For the successive approximations Y^i of $A\,\Phi_\Gamma$, induction hypothesis (2), intuitively means that $s \in \text{val}(T, Y^i)$ implies there is model M obtained by unwinding T in the obvious fashion s.t. $M, s \models Y^i$. Note that since Y^i is an approximation of $A\,\Phi$, $M, s \models Y^i$ implies $M, s \models A\,\Phi$. By the Small Model Theorem it follows that $T, s \Vdash A\,\Phi$, and hence $T, s \Vdash \exists\alpha\, Y^\alpha$.

Induction Hypothesis
(1) $s \in \text{val}(T, A\,\Phi_\Gamma) \equiv T, s \Vdash \exists i\, Y^i$.
(2) $s \in \text{val}(T,\ Y^i) \Rightarrow T, s \Vdash \exists\alpha\, Y^\alpha$
(Soundness) and,
$T, s \Vdash Y^i \Rightarrow s \in \text{val}(T, Y^i)$
(Completeness).

Induction Hypothesis 1 follows from Induction Hypothesis 2, by noting that all the ordinals concerned are $\leq |T|$, i.e. although $s \in \text{val}(T, Y^i)$ only implies that there is a model contained in T of Y^α, for some α which could be greater than i, but since the ranks are bound by $|T|$, the union $\bigcup_i \text{val}(T, Y^i)$ converges to $\{s : T, s \Vdash A\,\Phi_\Gamma\}$. Induction Hypothesis 2 follows by assuming that Induction Hypothesis 1 holds for all $\Gamma' \subset \Gamma$, using Lemma 6 above, plus the definition of val(T, $q \vee r$).

Complexity: Note that, if we were just model checking, i.e. computing $\{s : T, s \models A\,\Phi_\Gamma\}$, then we could compute the different Y^i's bottom up, i.e. by computing Y^0,Y^1.. etc as is done in the model checking algorithm for $L\mu$. Y^i has $A\Phi_{\Gamma/\{\gamma\}}$ as a subexpression, and we could have model checked the different $A\Phi_{\Gamma'}$ in T, where $\Gamma' \subseteq \Gamma$, bottom up as well. But, while computing $\{s : T, s \Vdash A\,\Phi_\Gamma\}$ by modifying the model checking algorithm it did not suffice, as we saw earlier, to just compute $\{s : T, s \Vdash A\Phi_{\Gamma'}\}$. The modification we gave requires computing $\{s : U, s \Vdash A\Phi_{\Gamma'}\}$, where U is a bipartite AND-OR graph subset of T. Moreover, these U's in which we pseudo model check $A\Phi_{\Gamma'}$, depend on Γ and T. In fact, pseudo model checking $A\,\Phi_\Gamma$ in T, requires $O(|\Gamma| \cdot |T| \cdot |T|)$ computations of pseudo model checking pairs conditions with one less pair than Γ. The local cost of each computation is polynomial in size of T, and thus we get an algorithm which is polynomial in size of T and exponential in the number of pairs.

We analyze the complexity in more detail as follows: Let $Com(T, f)$ stand for the complexity of computing val(T, f). Let $|T| = m$ and $|\Gamma| = n$. Clearly, $Com(T, A\Phi_\Gamma) \leq Com(T, Y^m)$.

$$
\begin{aligned}
&Com(T, Y^i) \\
&\quad \leq n \cdot Com(T, AFAGg(Y^{i-1})) \\
&\qquad \{\text{ By Lemma 6 cost of computing the AF part is } O(m^2)\} \\
&\quad \leq n \cdot (m^2 + Com(T, AGg(Y^{i-1}))) \qquad (1) \\
&Com(T, AGg(Y^{i-1})) \\
&\quad \leq m \cdot Com(Z^j, g(Y^{i-1})) \\
&Com(Z^j, g(Y^{i-1})) \\
&\quad \leq Com(Z^j/AFQ_\gamma, A\Phi_{\Gamma/\{\gamma\}}) + \\
&\qquad Com(Z^j, AFQ_\gamma) + Com(T, Y^{i-1}) \qquad (2)
\end{aligned}
$$

Note that, val(T, Y^{i-1}) needs to be computed only once, and hence can be removed from the last inequality (2), and instead we can add $Com(T, Y^{i-1})$ to right hand side of inequality (1). That is,

$$
\begin{aligned}
&Com(T, Y^i) \leq \\
&\qquad n \cdot (m^2 + m \cdot (m^2 + Com(U, A\Phi_{\Gamma/\{\gamma\}}))) + \\
&\qquad Com(T, Y^{i-1})
\end{aligned}
$$

where $|U| \leq |T|$. Let $C(m, n)$ denote $Com(T, A\Phi_\Gamma)$. Then,

$$C(m, n) \leq 2nm^4 + nm^2 \cdot C(m, n-1) \qquad (3)$$

It is easily seen that $C(m, n)$ is at most $O(n \cdot n! \cdot m^{2n+2})$, which is at most $O((mn)^{3n})$. This completes the proof of Theorem 3.1. □

Theorem 3.2: Pairs automaton non-emptiness is NP-complete.
Proof Sketch: Pairs automaton non-emptiness was shown to be in NP in [Em85] (cf. [VS85]). To get NP-hardness we reduce 3-SAT to Pairs-automaton non-emptiness. Let C be a 3-CNF formula with m clauses

$C_1, C_2, \ldots, C_m$. We reduce satisfiability of C to the non-emptiness problem of a pairs automaton $\mathcal{A}$ with $O(|C|)$ states and $O(|C|)$ pairs; or equivalently to testing whether there exists a Hintikka structure contained in the transistion diagram meeting the pairs condition.

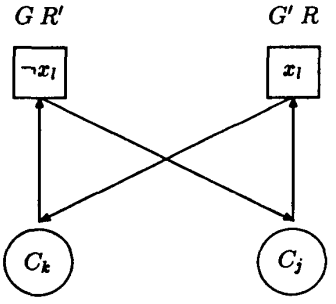
The transition diagram of $\mathcal{A}$ is viewed as an AND-OR graph as usual. Since we are only concerned about the emptiness problem, $\mathcal{A}$ is assumed to have a one symbol alphabet. Corresponding to each clause C_i, $\mathcal{A}$ will have an OR-node C_i (of the same name). The idea is to get all the clauses satisfied by a single assignment θ to the variables $x_1, \ldots x_k$, so let start state S_0 be an OR-node with a unique AND-node successor S which has edges to each of the OR-nodes C_i. Each Clause C_i, can be satisfied by any one of the literals occurring in C_i. Therefore, we have AND-nodes corresponding to all possible literals (i.e. all variables and their negations), with OR-node C_i having edges to the AND-nodes corresponding to the literals occurring in it. If C_i uses literal x (i.e. x is set true in θ), then their may be other clauses in which the negation of x occurs, and hence these other clauses (which we will call pending clauses) must use some other literals to satisfy them. To force this, we have edges out of each AND-node x to OR-nodes corresponding to these pending clauses, i.e. clauses in which the negation of x occurs.

But, this way it may be possible that these pending clauses get satisfied by using edges to AND-node $\neg x$, since $\neg x$ is a literal in these pending clauses. To ensure that this does not happen we use the pairs in the acceptance condition. We have a pair for each literal. The green light of a pair corresponding to literal x will be on AND-node x, and the red light on the AND-node $\neg x$. Similarly, for the pair corresponding to the literal $\neg x$. Thus,

$\mathcal{A}$ is non-empty

iff {By Small Model Theorem}
There is a Hintikka structure H contained in $\mathcal{A}$, accepted by $\mathcal{A}$

iff In H, each C_j has an edge to x_l, and no other C_k has an edge to the negation of x_l. { The if part is easy to see, because the right hand side of the above "iff" implies that for every green light labelling a node in H its corresponding red light doesn't occur in H. For the 'only if' part consider the negation of the right hand side, i.e. suppose there is a C_j with an edge to x_l, and a C_k with an edge to $\neg x_l$. Then we have the following subgraph in H



Here (G, R) and (G', R') are the pairs corresponding to the literals x_l and $\neg x_l$ resp. The arcs $x_l \rightarrow C_k$ and $\neg x_l \rightarrow C_j$ exist by definition of $\mathcal{A}$, considering that $\neg x_l$ is in C_k and x_l is in C_j. The above subgraph generates a path which has for all green lights flashing infintely often, its corresponding red light flashing infinitely often too. This contradicts the left hand side of the above "iff"}

iff C is satisfiable. □

Theorem 3.3: Complemented-pairs non- emptiness is coNP-complete. □

The proof reduces complemented-pairs automaton emptiness to pairs automaton non-emptiness, and vice versa, by making AND-nodes OR-nodes, and OR-nodes AND-nodes, and complementing the acceptance condition. Note that, the reduction is not trivial, because the full acceptance condition for a tree automata is of the form: THERE EXISTS a run such that FOR ALL paths an infinite string condition holds, while the complementation of the string condition in Streett's automata takes place within the scope of the two second order quantifiers. The proof goes by defining infinite games ([GH82], [MS84]), which are known to be Borel, and using the fact that they are determinate (Martin's Theorem [Ma75]). The same reduction can be used to check non-emptiness of Complemented pairs automaton in time exponential in the

no. of pairs, but only polynomial in the size of the automaton.

4. Applications to Logics of Programs

Using existing reductions and the new algorithm for pairs automaton, we give essentially optimal algorithms for CTL^*, PDL-delta, and Mu-calculus. The existing reductions we use, reduce satisfiability of these logics to non-emptiness of Rabin Automata. These reductions employ determinization of a Non- deterministic Buchi automata on ω-strings. With existing algorithms for testing non-emptiness of Rabin-Automata and the McNaughton determinization Construction [Mc66], we get a Non-det. double exponential algorithm for testing satisfiability of PDL-delta and Mu-calculus. Using the new determinization construct of Safra [Sa88], we would get a Non-det. single exponential algorithm for these two logics. Then, using our new non-emptiness testing algorithm we get a deterministic single exponential algorithm. The existing Upper Bound for these logics, i.e. non-deterministic single exponential [VS85], employed Hybrid-automata. The Safra construction alone does not help reduce the complexity using the Hybrid Automata. For CTL^*, the existing reductions [ESi84], which use a special structure of the automata associated with the logic to determinize it, and the previous non-emptiness algorithm give a Nondet. double-exponential algorithm. The new non-emptiness algorithm reduces the Upper Bound to deterministic double exponential.

Theorem 4.1[ESi84]: There is a exp(exp($|p|$)) time reduction of a CTL^* formula p into an automaton A_p on infinite trees with exp(exp($|p|$)) states and exp($|p|$) pairs such that p is satisfiable iff A_p is non-empty. □

Theorem 4.2: The satisfiability problem for CTL^* is complete for deterministic double exponential time.
Proof: Applying the non-emptiness algorithm (Theorem 3.1) to the pairs automaton obtained by Theorem 4.1 it follows that CTL^* can be decided in deterministic double exponential time. CTL^* was shown hard for double exponential time in [VS85]. □

Theorem 4.3: Satisfiability of Mu-calculus is in det. exponential time.
Proof: For a Mu-calculus formula p a Rabin's Pair automaton A_p of size exp($|p|$) and number of pairs polynomial in $|p|$ can be constructed in time exp($|p|$), such that p is satisfiable iff A_p is non-empty. This reduction follows by the technique described in [StE84], but using Safra's construction instead of McNauhton's construction. The upper bound follows by using the non-emptiness algorithm for Rabin Automata (Theorem 3.1). □

PDL-delta has a linear blow-up translation to Mu-calculus ([EL86], [Ko83]) and hence PDL-delta is also in deterministic exponential time. A direct algorithm, similar to the one mentioned for Mu-calculus, can also be given (See [St81], [VS85]). Because of the known deterministic exponential time Lower Bound for PDL (FL79]), it follows that the above algorithms for Mu-calculus and PDL-delta are essentially optimal (i.e. upto polynomial blowup).

In the full paper we show how, using our algorithm plus Safra's construction, to test nonemptiness of the succinct Hybrid Automata of [VS85] in deterministic exponential time. We also show that the process logic YAPL of [VW83] is in deterministic double exponential time, and that the branching time logic analogous to CTL* but with the linear Mu-calculus in place of linear temporal logic is also in deterministic double exponential time.

References

[BB86] B. Banieqbal and H. Barringer, "A Study of an Extended Temporal Language and a Temporal Fixed Point Calculus", Univ. of Manchester Tech. Report UMCS-86-10-2.

[CE81] E.M. Clarke, E.A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic", Proc. IBM Workshop on Logics of Programs, *Lecture Notes in Computer Science* No. 131, Springer-Verlag.

[CVW86] C. Courcoubetis, M.Y. Vardi, P. Wolper, "Reasoning about Fair Concurrent Programs", *Proc. 16th ACM STOC* , 1986.

[Em85] E.A. Emerson, "Automata,tableaux, and temporal logics", *Proc. Workshop on Logics of Programs*, Brooklyn, June 85.

[EC82] E.A. Emerson, E.M. Clarke, "Using Branching Time Logic to Synthesize Synchronization Skeletons", *Science of Computer Programming*, 2, pp. 241-266, 1982.

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

[EL86] E.A. Emerson, C.L. Lei, "Efficient Model Check-

ing in Fragments of the Propositional Mu-Calculus ", *IEEE Symposium on Logics in Computer Science*, 1986, pp.267-278.

[ESi84] E.A. Emerson, A.P. Sistla, "Deciding Branching Time Logic", *Information and Control*, vol. 61, no. 3., pp. 175-201, June 1984.

[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.

[GH82] Y. Gurevich, L. Harrington, "Trees, Automata, and Games", *14th ACM STOC*, 1982.

[Ha82] J.Y. Halpern, "Deterministic Process Logic is Elementary", *23rd IEEE FOCS*, 1982.

[HKP82] D. Harel, D. Kozen, R. Parikh, " process Logic: Expressiveness, Decidability, Complexity", *JCSS*, 25, pp. 144-170, 1982.

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

[LPZ85] Lichtenstein, O., Pneuli, A., Zuck, L., "The Glory of the Past", *International Colloq. on Automata, Languages, and Programming*, Lecture Notes in Computer Science, 1985, Springer- Verlag.

[Ma75] D.A. Martin, "Borel Determinacy", *Annals of Mathematics*, 1975, 102, 363-371.

[Mc66] R. McNaughton, "Testing and Generating Infinite Sequences by a finite Automaton", *Information and Control*, 9, 521-530, 1966.

[MS84] D.E. Muller, P.E. Schupp, "Alternating Automata on Infinite Objects, Determinacy and Rabin's Theorem", *Automata on Infinite Words*, 1984, Lecture Notes in Computer Science, 192.

[MW84] Z. Manna, P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications", *ACM TOPLAS*, Vol. 6, No.1, pp 68-93, 1984.

[Pa83] R. Parikh, "Propositional Game Logic", *25th IEEE FOCS*, 1983, pp. 195-200

[Pn77] A. Pneuli, "The Temporal Logic of Programs", *19th IEEE FOCS*, 1977.

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

[Sa88] S. Safra, "On Complexity of ω-automata", *to appear in 29th IEEE FOCS, 1988.*

[St81] R.S. Streett, "A Propositional dynamic Logic of Looping and Converse", *MIT LCS* Technical Reeport TR-263.

[SC85] A.P. Sistla, E.M. Clarke, "The Complexity of Propositional Linear Temporal Logic", *JACM*, vol. 32, no. 3, pp 733-749, July 1985

[StE84] R.S.Streett, E.A. Emerson, "An Elementary Decision Procedure for the Mu-calculus", *Proc. 11th Int. Colloq. on Automata, languages and Programming*, 1984, Lecture Notes in Computer Science, Springer-Verlag.

[V87] M.Y. Vardi, "Verification of Concurrent Programs: The Automata-Theoretic Framework", *Logics in Computer Science*, 1987.

[V88] M. Y. Vardi, A Temporal Fixpoint Calculus, POPL88

[VS85] M. Vardi, L. Stockmeyer, " Improved Upper and Lower Bounds for Modal logics of Programs", *17th ACM STOC*, 1985, pp.240-251.

[VW83] M. Y. Vardi , P. L. Wolper, Yet Another Process Logic, CMU Workshop on Logics of Programs, Springer LNCS no. 164, June 1983.

[VW84] M.Y. Vardi, P. L. Wolper, "Automata-Theoretic Techniques for Modal Logics of Programs", *Proc. 14th ACM STOC* , 1984.