Automata Theoretic Techniques for Modal Logics of Programs

(extended abstract)

Moshe Y. Vardi[†]
IBM Research, San Jose

Pierre Wolper[‡]
AT&T Bell Laboratories

Abstract

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

1. Introduction

Propositional modal logics of programs are formal systems for reasoning about the behavior of program schemes. They are of two different types: dynamic logics, a la Pratt [Pr76], are used for reasoning about the input/output behavior of program schemes, while temporal logics, a la Pnueli [Pn77], are used for reasoning about their ongoing behavior. Most of the propositional program logics studied in the literature are known to have decidable satisfiability problems. A general technique to show their decidability is by reduction to SnS, the second-order theory of n successor functions [Ra69]. The upper bound established by that reduction is, unfortunately, nonelementary [Me75].

For several of these logics exponential time upper bounds have been established using the so-called *small model property*. This property, established first for propositional dynamic logic [FL79], says that if a formula of length n is satisfiable, i.e., if it has a model, then it also has a "small model", i.e., a model whose cardinality is at most exponential in n. While this property by itself gives only a nondeterministic exponential time upper bound, it has been sharpened by Pratt [Pr79,Pr80] to give a deterministic exponential time upper bound. Pratt has shown two techniques, the *tableau* technique [Pr80] and the *maximal model* technique [Pr79], to deterministically construct models of size exponential in the length of the formula.

Unfortunately, for certain logics the structures constructed by Pratt's techniques are actually not models. However, they are still useful for a decision

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

procedure, because they can be shown to be *pseudo-models* [BHP82,EH82], i.e., they can generate a (possibly infinite) model by a process of *unwinding*. Showing that pseudo-models can be unwound to models is the most difficult part in the decidability proofs. It depends on the intricacies of the logic at hand, and despite obvious similarity does not carry over from logic to logic. Thus, while it is known that propositional dynamic logic with deterministic programs (*DPDL*) is decidable in exponential time [BHP82], it is not known whether this upper bound still holds if we add to the logic the *loop* construct, as is the case for propositional dynamic logic with nondeterministic programs (*PDL*) [PS83]. (Note that *DPDL* is not a special case of *PDL*.)

Another useful property of these logics is the *tree model property*. 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 the this property. The decidability of *SnS* has been established via a reduction to the emptiness problem of automata on infinite trees [Ra69]. This suggest that decision procedures for the propositional program logics can be obtained by directly reducing satisfiability to that emptiness problem. The idea is, for a given a formula f, to construct a tree automaton A such that A accepts exactly the tree models of f. Thus f is satisfiable if and only if A accepts some tree. This approach was used by Streett [St81] to establish elementary upper bounds for *PDL* augmented with the *repeat* and *converse* constructs, and one is tempted to try to apply Streett's technique to other logics as well.

Unfortunately, this technique can not even establish the known exponential upper bounds. For example, for *PDL*, whose exponential upper bound proof does not even require the pseudo-model argument [Pr79,Pr80], Streett's technique establishes a triply exponential upper bound. Indeed, the size of the constructed automaton is exponential in the size of the formula, and testing for emptiness takes time doubly exponential in the size of the automaton. The problem is that, for most logics, one does not need the full power of Streett's automata, and it is that power that makes it so hard to test for emptiness. Here, we introduce a new type of automaton on infinite trees, which we call *eventuality automaton*, and show that for these automata emptiness can be tested in polynomial time. Nevertheless, eventuality automata are powerful enough for the reduction from satisfiability to emptiness to work.

The resulting technique turns out to be powerful and unifying. It enables us to supply simpler proofs for known results and to obtain many new exponential upper bounds. The power of the technique lies in the fact that the proof of our polynomial time emptiness algorithm relies on an unwinding argument. This unwinding corresponds to the unwinding of pseudo-models to models. It is done here, however, in an automata-theoretic framework, with no need to take the intricacies of the logic into account, and it is done once and for all, with no need to repeat it for every logic. Another advantage is that this technique does not depend on the small model property, which is much harder to establish than the tree model property.

We demonstrate our technique here with two proofs. We first reprove an exponential upper bound for *DPDL* (propositional dynamic logic with deterministic programs). Our proof is significantly simpler than the original proof in [BHP82]. The same technique also yields simpler exponential upper bound proofs for *PDL* [FL79], *PDL+loop* [PS83], *flowchart-PDL* [Pr81,HS83b], *unified branching time logic* [BMP81], and *computation tree logic* [EH82].

We then prove an exponential upper bound for a propositional μ-calculus, $M\mu^-$, which is a propositional modal logic with mutual fixpoints. An exponential upper bound for another μ-calculus was proven by Kozen [Ko82]. Though his logic is more expressive than *PDL*, it does not seem to be able to express the *loop* construct. Also the known translation from *PDL* causes an exponential blow-up, unless common subformulas are consolidated. In contrast, $M\mu^-$, which extends Kozen's logic, is strictly more expressive than *LPDL*, which is *PDL* augmented with *loop*, and there is a linear translation from *LPDL* to $M\mu^-$. The hard part in establishing the exponential upper bound for $M\mu^-$ is proving that it has the tree property, i.e., that it has tree models of bounded branching. It is not hard to show that $M\mu^-$ has tree models, but bounding the branching requires an elaborate model-theoretic analysis. $M\mu^-$ is the most expressive extension of PDL for which an exponential decision procedure is known.

2. Eventuality Automata

We define a new class of finite automata on n-ary infinite trees, which we call *eventuality automata*. What distinguishes them from other automata on infinite trees [Ra69, HR72, St81] is the type of acceptance condition that is imposed. Formally, an n-ary infinite tree over an alphabet Σ is a mapping $\{1,...,n\}^* \to \Sigma$, and an eventuality automaton is a tuple $A=(\Sigma,S,\rho,s_0,F)$ where

- Σ is the alphabet.
- S is a set of states.
- $\rho: S \times \Sigma \to 2^{S^n}$ is the transition function. For each state and letter it gives the possible sets of n successors.
- $s_0 \in S$ is the initial state.
- $F \subseteq 2^S$ is a collection of sets of states. As we will see later, F defines the acceptance condition.

A run of an automaton A over a tree $T:\{1,...,n\}^* \to \Sigma$ is a mapping $\phi:\{1, \ldots ,n\}^* \to S$ where $\phi(\Lambda)=s_0$ and for every $x \in \{1,...,n\}^*$, if $\phi(x)=s$ then $<\phi(x1), \ldots ,\phi(xn)> \in \rho(s,T(x))$. An infinite path p through a tree is an infinite sequence $p=p_0,p_1,p_2,\ldots \in \{1, \ldots ,n\}^\omega$. Given a run ϕ, an infinite path defines an infinite sequence of states $\sigma_p=\phi(\Lambda),\phi(p_0),\phi(p_0p_1),\phi(p_0p_1p_2), \cdots$. Let us denote by $inf(\sigma_p)$ the set of states from S appearing infinitely often in σ_p. A run of A over T is accepting iff, for all infinite paths p and for all $X \in F$, $inf(\sigma_p) \cap X \neq \emptyset$ (this condition is similar to the one used by Buchi [Bu62] for automata on infinite words).

In Streett's *complemented pair automata*, the acceptance condition is given by a collection of pairs of sets $\{(X_i,Y_i)\}$, with the requirement that if $inf(\sigma_p) \cap X_i \neq \emptyset$ then $inf(\sigma_p) \cap Y_i \neq \emptyset$. Thus eventuality automata can be viewed as complemented pair automata, where every set $X \in F$ correspond to a pair $(S-X,X)$. Our definition was motivated by the way *eventualities* behave in modal program logics (hence the name eventuality automaton). An eventuality is a formula that requires that some other formula will eventually hold (e.g. Fp in temporal logic). In automata terms, it can be viewed as stating that if one goes through a given state (where the eventuality is required) then one will go through a state where it is satisfied. The important property of eventualities, is that an eventuality will continuously be required until it is satisfied. This explains our definition of acceptance if one views a set $S-X$, $X \in F$, as the set of states where an eventuality is required and not satisfied, and the set X as the set of states where it is satisfied or not required.

This small difference in definition makes a huge difference in the complexity of the emptiness problem for the automata, i.e., the problem whether there is some tree accepted by a given automaton. For complemented pair automata, the best known upper time bound is doubly exponential in the size of the automata [St81], whereas for eventuality automata, we have obtained a polynomial time algorithm. Intuitively, the reason the problem is easier for eventuality automata is that when the automata reaches a state in some $S-X$, $X \in F$, it remains in $S-X$ until it reaches a state in X. This implies that there is not need to remember past states, which is what makes the general problem hard.

Theorem 1: The emptiness problem for eventuality automata is logspace complete for PTIME.

Proof: Since we deal with nondeterministic automata, we can assume that the alphabet Σ consists of a single letter a.

Let us define a tree embedded in an automaton A and rooted at a state s as a mapping $t:\{1, \ldots, n\}^* \rightarrow S$, where $t(\Lambda)=s$ and for every $x \in \{1, \ldots, n\}^*$, we have $<t(x1), \ldots, t(xn)> \in \rho(t(x),a)$. A finite tree embedded in an automaton is then simply a finite subtree of an embedded tree. The algorithm for testing emptiness works by repeatedly eliminating states of the automaton according to the following rules:

1) Eliminate states s such that $\rho(s,a)=\emptyset$.

2) Eliminate all states s such that $s \in S-X$, $X \in F$ and for which there is no finite tree embedded in the automaton rooted at s whose frontier is entirely in X.

After a state is eliminated, the transition function ρ is updated accordingly. The algorithm stops when no more states can be eliminated. The automaton accepts some tree iff the initial state is not eliminated.

To implement rule (2) we use the algorithm in [TW68] for testing emptiness of automata on *finite* trees, which runs in time polynomial in the size of the automata, i.e., the length of their encoding. Clearly, our algorithm runs in time polynomial in the size of the automata. It remains to prove that the initial state is not eliminated iff the automaton accepts some tree.

- If the initial state is eliminated, then the automaton does not accept any tree.

We prove this by showing that no eliminated state can be part of an accepting run.

- If the initial state is not eliminated, then the automaton accepts some tree.

We prove this by constructing an accepting run. Let $F=\{X_0, \ldots, X_{k-1}\}$. Since the algorithm has eliminated all eliminable states, we know that for all states s there is a finite tree of positive depth embedded in the automaton, which is labeled by noneliminable states. Furthermore, if $s \in S-X_i$, then there is such a finite tree whose frontier is labeled by states in X_i. We construct the run in countably many stages, where at each stage we have a finite subtree of an accepting run. At stage 0 we have $\phi(\Lambda)=s_0$. at stage i, $0<i<\omega$, we append to each leaf of the finite tree constructed in stage $i-1$ a finite tree of positive depth. Furthermore if the label on the leave is s and $s \in S-X_{i(\mathrm{mod}\ n)}$, then the frontier of the appended tree is labeled by states in $X_{i(\mathrm{mod}\ n)}$. We can prove that the constructed run is an accepting one.

We have shown that the emptiness problem for eventuality automata is in PTIME. We can also show that the problem is hard for PTIME by reduction from the path system problem in [JL77]. □

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

3. Deterministic Propositional Dynamic Logic

Deterministic propositional dynamic logic (*DPDL*) is a propositional dynamic logic [FL79] where the atomic programs are deterministic. It was studied in [BHP82] where a one exponential decision procedure was given. The proof of the decision procedure given there is quite complicated. Here, we show how it can be substantially simplified using the automata theoretic result established in the previous section. We will consider a variant of *DPDL*, in which programs are described by finite automata rather than by regular expressions (c.f. [HS83]). This variant is called *ADPDL*. *ADPDL* is more succinct than *DPDL* and also has the advantage of fitting nicely with our automata theoretic techniques. As the translation from regular expressions to automata is linear, our results for *ADPDL* apply easily to *DPDL*.

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

- every proposition $p \in Prop$ is a formula.
- if f_1 and f_2 are formulas, then $\neg f_1$ and $f_1 \wedge f_2$ are formulas.

- if α is a program and f is a formula, then $<\alpha>f$ is a formula
- If α is a nondeterministic finite automaton over an alphabet Σ, where Σ is a finite subset of $Prog \cup \{f? \mid f$ is a formula$\}$, then α is a program.

We will assume that the automaton α is of the form $\alpha=(\Sigma,S,\rho,s_0,F)$ where S is a set of states, $\rho:S\times\Sigma\rightarrow 2^S$ is the transition relation, s_0 is the initial state, and F is the set of accepting states. If α accepts exactly the word a, where $a\in Prog$, then we denote the formula $<\alpha>f$ by $<a>f$. Similarly, if α accepts exactly the word $g?$, where g is a formula, then we denote the formula $<\alpha>f$ by $<g?>f$.

ADPDL 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 only 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:

- $R(f?)=\{(u,u):M,u\models f\}$.
- $R(\alpha)=\{(u,u')$: there exists a word $w=w_0w_1\cdots w_n$ accepted by α and states $u_0,u_1,\ldots,u_{n+1}$ such that $u=u_0$, $u'=u_{n+1}$ and for all $0\leq i\leq n$ we have $(u_i,u_{i+1})\in R(w_i)\}$.
- For a proposition $p\in Prop$, $M,u\models p$ iff $p\in\Pi(u)$.
- $M,u\models f_1\wedge f_2$ iff $M,u\models f_1$ and $M,u\models f_2$.
- $M,u\models\neg f_1$ iff not $M,u\models f_1$.
- $M,u\models <\alpha>f$ iff there exists a state u' such that $(u,u')\in R(\alpha)$ and $M,u'\models f$.

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

A formula f is *satisfiable* if there is a structure M and a state u in the structure such that $M,u\models f$. The *satisfiability problem* is to determine, given a formula f, whether f is satisfiable. Before giving the decision procedure for satisfiability of *ADPDL* formulas, we need to define a notion of closure of *ADPDL* formulas similar to the closure defined for *PDL* in [FL79]. Given a program automaton $\alpha=(\Sigma,S,\rho,s_0,F)$, for each $s\in S$ we define α_s to be the automaton (Σ,S,ρ,s,F), i.e., the state s replaces s_0 as the initial state. The closure of a formula f, denoted $cl(f)$, is defined as follows:

- $f\in cl(f)$
- If $f_1\wedge f_2\in cl(f)$ then $f_1,f_2\in cl(f)$.
- If $\neg f_1\in cl(f)$ then $f_1\in cl(f)$.
- If $f_1\in cl(f)$ and f_1 is not of the form $\neg f_2$ then $\neg f_1\in cl(f)$.
- If $<\alpha>f_1\in cl(f)$ then $f_1\in cl(f)$.
- If $<f_1?>f_2\in cl(f)$ then $f_1\in cl(f)$.
- If $<\alpha>f_1\in cl(f)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$ and $s\in\rho(s_0,w)$ for some $w\in\Sigma$, then $<w><\alpha_s>f_1\in cl(f)$.

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

For our techniques to be usable, we first have to prove that *ADPDL* has the tree model property. If a formula f is satisfiable, then it is satisfiable in some state u of some structure M. Clearly, M can be unraveled into a tree with u as its root. Furthermore, as all atomic programs are deterministic, the branching factor of the tree is at most the number of atomic programs that occur in f. Note that the tree model can have infinitely many states, even if the original model was finite.

Let the atomic programs that occur in f be $a_1,\ldots,a_k$. A tree model M gives rise to a full k-ary tree T over the alphabet $2^{cl(f)}$ in a natural way. Each node in M is labeled by the formulas in $cl(f)$ that are satisfied at that node. If a state u_2 is the a_i-successor of a state u_1, i.e., $(u_1,u_2)\in R(a_i)$, then we take u_2 to be the i-th successor of u_1 in T. To make T into a full tree, we add dummy nodes. These nodes are labeled by the empty set. We call T the *annotation* of M.

The next step is to build an eventuality automaton on k-ary infinite trees over the alphabet $2^{cl(f)}$ that accepts an infinite tree iff it is the annotation of a tree model of f. The construction proceeds in two steps. First we build an automaton, called the *local* automaton, that checks the model locally. For example, it verifies that if the formula $f \wedge g$ belongs to a label of a node u, then the formulas f and g also belong to that label. Next, we build an automaton that ensures that all *eventualities* are indeed satisfied (the <>-automaton). Eventualities are formulas of the form $<\alpha>f$, for some program α. For example, if the formula $<\alpha>f$ belongs to a label of a node u, where α accepts the language a^* for some atomic program a, then the formula f must belong to the label of some a^*-descendant of s. Finally, we combine the local and <>-automaton.

The Local Automaton

The local automaton is $L=(2^{cl(f)},N_L,\rho_L,N_f,\varnothing)$. The state set N_L will be the collection of all sets Y of formulas in $cl(f)$ in which there is no propositional inconsistency or contradiction with the transitions of the program automata. Namely, they must satisfy the following conditions (we identify a formula g with $\neg\neg g$):

- $g \in Y$ iff $\neg g \notin Y$.
- $g_1 \wedge g_2 \in Y$ iff $g_1 \in Y$ and $g_2 \in Y$.
- $<g_1?>g_2 \in Y$ iff $g_1 \in Y$ and $g_2 \in Y$.
- $<\alpha>g \in Y$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, iff either $s_0 \in F$ and $g \in Y$ or there exists some $w \in \Sigma$ and $s \in S$ such that $s \in \rho(s_0,w)$ and $<w><\alpha_s>g \in Y$.

For the transition relation ρ_L, we have that $(Y_1, \ldots, Y_n) \in \rho_L(Y,X)$ iff $X=Y$ and:

- If $<a_i>g \in Y$, where a_i is an atomic program, then $g \in Y_i$.
- If $\neg<a_i>g \in Y$, where a_i is an atomic program, then $g \notin Y_i$.

Finally, the set of starting states N_f consists of all sets Y such that $f \in Y$. The local automaton does not impose any acceptance conditions, thus the set of acceptance sets is empty.

The <>-Automaton

Given an *ADPDL* formula f, we defined the set $e(f)$ of its eventualities as the subset of $cl(f)$ that contains all formulas of the form $<\alpha>g$. The <>-automaton is $E=(2^{cl(f)},2^{e(f)},\rho_E,\varnothing,\{\varnothing\})$. The state set will be the collection of all sets Y of formulas in $e(f)$. For the transition relation ρ_E, we have that $(Y_1, \ldots, Y_n) \in \rho_E(Y,X)$ iff either:

- $Y=\varnothing$: If $<a_i><\alpha>g \in X$, where a_i is an atomic program, then $<\alpha>g \in Y_i$.
- $Y \neq \varnothing$:
  - If $<\alpha>g \in Y$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, then either $s_0 \in F$ and $g \in X$ or there exists some $w \in \Sigma$ and $s \in S$ such that $s \in \rho(s_0,w)$ and $<w><\alpha_s>g \in Y$.
  - If $<g_1?><\alpha>g_2 \in Y$ then $g_1 \in X$ and $<\alpha>g_2 \in Y$.
  - If $<a_i><\alpha>g \in Y$, where a_i is an atomic program, then $<\alpha>g \in Y_i$.

Note that the definition of ρ_E also takes care of immediate violations of eventualities in the states of E.

Intuitively,the <>-automaton tries to satisfies the eventualities in the model. When the current state is $\varnothing$, it looks at the model to see which eventualities have to be satisfied. Then, the current state says which eventualities have yet to be satisfied. Observe that E does not try to satisfy the eventualities of each state of the model. However, it tries to satisfy the eventualities infinitely often on all paths of the tree model, which is sufficient.

Combining the Automata

The *model automaton*, which accepts exactly the annotations of tree models of f, is the cross product of the local automaton and the <>-automaton. The model automaton is an eventuality automaton. Its size is exponential in the size of $cl(f)$ and hence in the length of f. The procedure described in the previous section can be used to test if it accepts some infinite tree. As it accepts an infinite tree iff the formula f is satisfiable, this establishes the existence of a one exponential decision procedure for the satisfiability problem for *ADPDL*.

4. Propositional μ-calculus

The μ-calculus was suggested as a formalism for reasoning about programs in [SdB69], and was further developed in [Pa70], [Sc71], [HP73], and elsewhere. More recently, its usefulness for reasoning about parallel programs was demonstrated in [Cl77, Re79, FS81]. This approach is centered around a *least fixpoint* operator, μ, which serves to capture the notions of iteration and recursion. The μ-calculus was originally defined as a predicate calculus. In [Ko82] a propositional version of the μ-calculus, called $L\mu$, was introduced. $L\mu$ was inspired by [Pr81], where a propositional *least-root*-calculus was studied. The calculus $L\mu$ is decidable [KP83], but the best known upper time bound is multiply exponential [ES84]. However, in [Ko82], Kozen obtained an exponential time upper bound for a weaker version of $L\mu$, which we call $L\mu^-$. $L\mu^-$ is obtained from $L\mu$ by imposing a certain *aconjunctivity* condition.

Here, we define another μ-calculus, called $M\mu$, which extends $L\mu$ with the ability to define *mutual* fixpoints (mutual fixpoints enable us to reason directly about mutual recursion). We also introduce a restricted version of $M\mu$, $M\mu^-$, for which we give a one exponential decision procedure.

Syntax.

Formulas of $M\mu$ are defined from a set of atomic propositions *Prop*, a set of atomic programs *Prog* and a set of propositional variables *Var*. Formulas of $M\mu$ are defined inductively as follows:

- Every element $p \in Prop$ and $x \in Var$ is a formula.
- If f_1 and f_2 are formulas, then so are $f_1 \wedge f_2$, and $\neg f_1$.
- If f is a formula and a is an atomic programs, then $<a>f$ is a formula.
- Let $f_1, \ldots, f_k$ be formulas positive in the free variables $x_1, \ldots, x_k$, i.e., every free occurrence of the x_i's is inside an even number of negations. Then $\mu x_1, \ldots, x_k{:}x_i.(f_1, \ldots, f_k)$ is a formula.

(In $L\mu$, one must have for the last clause $k=1$, i.e., one can not define mutual fixpoints.)

Semantics.

$M\mu$ formulas are interpreted over structures $M=(W,R,\Pi)$ where W is a set of states, $R{:}Prog \rightarrow 2^{(W\times W)}$ assigns binary relations over W to atomic programs and $\Pi{:}Prop \rightarrow 2^W$ assigns subsets of W to propositions in *Prop*. A formula f with free variables among $x_1, \ldots, x_k$ is interpreted as a mapping f^M from $(2^W)^k$ to 2^W, i.e., it is interpreted as a predicate transformer. We write $f(\mathbf{x})$ to denote that all free variables of f are among $\mathbf{x}=x_1, \ldots, x_k$, and $f^M(\mathbf{A})$ to denote the value of f^M on the arguments $A_1, \ldots, A_k$. f^M is defined inductively:

- $p^M(\mathbf{A})=\Pi(p)$ for an atomic proposition p.
- $x_i^M(\mathbf{A})=A_i$ for a variable x_i
- $(f_1 \wedge f_2)^M(\mathbf{A})=f_1^M(\mathbf{A}) \cap f_2^M(\mathbf{A})$.
- $(\neg f)^M(\mathbf{A})=S-f^M(\mathbf{A})$.
- $(<a>f)^M(\mathbf{A})=\{u \mid \exists v \in f^M(\mathbf{A}) \text{ s.t. } (u,v) \in R(a)\}$.
- Let $\mathbf{f}=(f_1, \ldots, f_l)$ be formulas positive in the variables $\mathbf{y}=y_1, \ldots, y_l$. Then

$$(\mu \mathbf{y}{:}y_i.\mathbf{f})^M(\mathbf{A})=\bigcap\{B_i \mid \mathbf{f}^M(\mathbf{B},\mathbf{A}) \subseteq \mathbf{B}\}.$$

In other words, the mapping $\mathbf{f}^M=(f_1^M, \ldots, f_l^M)$ from $(2^W)^l$ to $(2^W)^l$ is monotone in the variables $\mathbf{y}$ with respect to the component-wise subset relation and the value of $(\mu \mathbf{y}{:}y_i.\mathbf{f})^M(\mathbf{A})$ is the i-th component of the least fixpoint of $\mathbf{f}^M$, whose existence is guaranteed by the Tarski-Knaster Fixpoint Theorem [Ta55].

If a formula f is closed, i.e., has no free variables, then f^M is a constant. In this case, a state u of a structure M *satisfies* f, denoted $M,u \models f$ if $u \in f^M$. The formula f is *satisfiable* if it has a *model*, i.e., if there is a structure M and a state u such that

$M,u \models f$.

For convenience, we convert all closed formulas of $M\mu$ to *positive normal form*, by pushing negation down until they are applied only to propositional constants. To do this, we introduce the logical connectives $\vee$, $[\,]$, which is the dual of $< >$, and ν, which is the dual of μ (ν is the *greatest fixpoint* operator). We also assume without loss of generality that no variable is quantified twice.

$M\mu^-$ is obtained from $M\mu$ by restricting the interaction of least fixpoints and greatest fixpoint. Let $\sigma y_1, \ldots, y_k{:}y_i.(f_1, \ldots, f_k)$ be a formula, where σ is either μ or ν. We say that a variable x *affects* y_j, if x is a subformula of f_j. We say that x affects y_j *conjunctively*, if in the parse tree of f_j there is a path from the root to x with no $\vee$-node. A formula f of $M\mu$ is in $M\mu^-$ if the following is not the case: there are formulas $f_1, \ldots, f_m$ and variables $y_1, \ldots, y_m$, such that

(1) f_1 is a μ-subformula of f and f_i is a ν-subformula of f_{i-1} for $2 \le i \le m$,

(2) y_i is a quantified variable of f_i, y_i affects y_{i+1} for $1 \le i \le m-2$, and y_{m-1} affects y_m conjunctively.

The above restriction is rather technical, and is quite hard to understand out of the context of the the proofs. The restriction is related to, albeit weaker than, *syntactic continuity* [Pa76,Pr81]. In particular it is weaker than Kozen's *aconjunctivity* restriction. It follows that $L\mu^-$ is a subset of $M\mu^-$. In the full paper we will give the intuition behind our restriction and relate it to a certain *compactness* property, which is related to *semantic continuity*. We will show that $M\mu^-$ is strictly more expressive than *LPDL*, and there is a linear translation from *LPDL* to $M\mu^-$. (Recall that *LPDL* is propositional dynamic logic augmented with the *loop* construct.) We also will show how $M\mu^-$ can describe "computation trees" in a straightforward way.

In order to reduce the satisfiability problem for $M\mu^-$ to the emptiness problem for eventuality automata, we have to show that $M\mu^-$ has the tree model property.

Theorem 2. Let f be a formula of $M\mu^-$. If f has a model, then it has a tree model with branching factor at most n, where n is the length of f.

Idea of Proof. Let M be a model of f. It is easy to show that M can be unraveled to a tree, so we can assume that M is already a tree. We have, however, no bound on the branching of this tree, which can even be infinite. (Note that for *DPDL* the bound on the branching follows from the determinism of the atomic programs). Nevertheless, intuitively, most of the branches are unessential to the satisfaction of f at the root. A branch is essential if is required to satisfy a subformula of f of the form $<a>g$, but f can have at most n such subformulas. While this intuitively seems to hold also for $M\mu$, our proof holds only for $M\mu^-$. The essential feature of $M\mu^-$ is that, in a tree model, a μ-formula is actually satisfied by a *finite* subtree (this is the compactness property mentioned above). In order to prove the property for $M\mu^-$, we use an infinitary extension of $M\mu$ where fixpoints are computed by explicit iteration. □

Now that we have established the tree property, we can apply our technique. Again we need to define a notion of closure of $M\mu^-$ formulas. The closure of a formula f, denoted $cl(f)$, is defined as follows:

- $f \in cl(f)$.
- If $f_1 \wedge f_2 \in cl(f)$ or $f_1 \vee f_2 \in cl(f)$ then $f_1, f_2 \in f$.
- If $\neg f_1 \in cl(f)$, $<a>f_1 \in cl(f)$, or $[a]f_1 \in cl(f)$ then $f_1 \in cl(f)$.
- If $f_1 \in cl(f)$ and f_1 is not of the form $\neg f_2$ then $\neg f_1 \in cl(f)$.
- If $\sigma y_1, \ldots, y_k{:}y_i.(f_1, \ldots, f_k) \in cl(f)$ then $f'_i \in cl(f)$, where f'_i is obtained from f_i by simultaneously substituting $\sigma y_1, \ldots, y_k{:}y_j.(f_1, \ldots, f_k)$ for y_j, $1 \le j \le k$.

Theorem 3. The satisfiability problem for $M\mu^-$ is logspace complete in EXPTIME.

Idea of Proof. Hardness for EXPTIME follows from that result for PDL [FL79]. To show that the problem is in EXPTIME, we consider tree models of bounded-branching. As in the previous section, we annotate these tree models by labeling the nodes by the formulas in $cl(f)$ that are satisfied at that node. We also label a node by the atomic programs that lead to it. That is, if u_2 is an a_i-successor of u_1 in the model, then u_2 is a successor of u_1 in the annotation and it has a_i in its label. Thus the annotation are n-ary labeled trees over $2^{cl(f)\cup Prog}$, where n is the length of f. We build now an eventuality automaton on n-ary trees that accepts exactly annotations of tree models of f. The construction is similar to that in the previous section. We take a cross product of two automata. The first one, the *local* automaton, checks the model locally, i.e., it checks that states are consistent and that $< >$ and $[\ ]$-formulas are obeyed. The second one, the μ-automaton, check that μ-formulas are satisfied.

The Local Automaton

The local automaton is $L=(2^{cl(f)\cup Prog},N_L,\rho_L,N_f,\emptyset)$. The state set N_L will be the collection of all sets Y in $2^{cl(f)\cup Prog}$ in which there is no immediate inconsistency. Namely, they must satisfy the following conditions (we identify a formula g with $\neg\neg g$):

- $g\in Y$ iff $\neg g\notin Y$.
- $g_1\wedge g_2\in Y$ iff $g_1\in Y$ and $g_2\in Y$.
- $g_1\vee g_2\in Y$ iff $g_1\in Y$ or $g_2\in Y$.
- If $\sigma y_1,\ldots,y_k{:}y_i.(f_1,\ldots,f_k)\in Y$ then $f'_i\in Y$, where f'_i is obtained from f_i by simultaneously subtituting $\sigma y_1,\ldots,y_k{:}y_j.(f_1,\ldots,f_k)$ for y_j, $1\le j\le k$.

For the transition relation ρ_L, we have that $(Y_1,\ldots,Y_n)\in\rho_L(Y,X)$ iff $Y=X$ and:

- If $<a_i>g\in Y$, then for some Y_j we have $a_i\in Y_j$ and $g\in Y_j$.
- If $[a_i]g\in Y$, then for all Y_j such that $a_i\in Y_j$ we have $g\in Y_j$.

Finally, the set of starting states N_f consists of all sets Y such that $f\in Y$. The local automaton does not impose any acceptance conditions, thus the set of acceptance sets is empty.

The μ-Automaton

For each μ-formula g in $cl(f)$, we build an eventuality automaton E_g that checks for the satisfaction of this formula. The μ-automaton is taken to be the cross product of all the automata E_g. Let g be the formula $\mu x_1,\ldots,x_k{:}x_i.(g_1,\ldots,g_k)$. E_g is the automaton $(2^{cl(f)\cup Prog},N_L,\rho_E,\emptyset,\{\emptyset\})$. The state set N_L is the same set as for the local automaton.

For the transition relation ρ_E, we have that $(Y_1,\ldots,Y_n)\in\rho_E(Y,X)$ iff either:

- $Y=\emptyset$: If $<a_i>f_1\in X$ and g is a Boolean subformula of f_1,[†] then for some Y_j we have $a_i\in Y_j$ and $f_1\in Y_j$.
- $Y\neq\emptyset$: $Y\subset X$ and if $<a_i>f_1\in Y$ and x_i is a subformula of f_1, then for some Y_j we have $a_i\in Y_j$ and $f_1\in Y_j$.

As noted before, μ-formulas are satisfied by finite subtrees of the model. The μ-automaton essentially searches for these trees. □

5. Concluding Remarks

We have presented a unifying technique for solving the decision problem for modal logics of programs. We have demonstrated our technique by proving exponential upper bounds for *DPDL* and for the propositional μ-calculus $M\mu^-$.

The technique is also easy to apply to other logics. In the full version of the paper we will prove exponential upper bounds for two extensions of *DPDL*, *DPDL+loop* and *DPDL+converse*, and for an extension of Parikh's *dual-free game logic* [Par83].

In [KP81], the maximal model technique was used to prove completeness of an axiom system for *PDL*. Their technique was extended in [BHP82] to *DPDL*. Can our technique be used to prove completeness results? This problem is now under investigation.

[†] g is a Boolean subformula of f_1 if g is a a conjunct in a disjunct in the disjunctive normal form of f_1.

References

[BHP82] M. Ben-Ari, J. Halpern, A. Pnueli, "Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness", *Journal of 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 Symposium on Principles of Programming Languages,* Williamsburg, January 1981, pp. 164-176.

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

[Cl77] E. M. Clarke, "Program Invariants as Fixpoints", *Proc. 18th IEEE Symposium on Foundations of Computer Science,* Providence, 1977, pp. 18-28.

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

[ES84] E. A. Emerson, A. P. Sistla, "An Elementary Decision Procedure for the μ-calculus", *Proc. 11th Int. Colloq. on Automata, Languages and Programming,* Lecture Notes in Computer Science, Springer-Verlag, Berlin, to appear.

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

[FS81] L. Flon, M. Suzuki, "The Total Correctness of Parallel Programs", *SIAM Journal of Computing,* 10(1981), pp. 227-245.

[HP73] P. Hitchcock, D. Park, "Induction Rules and Termination Proofs", *Proc. 1st Int. Colloq. on Automata, Languages and Programming,* Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1973, pp. 225-251.

[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.*

[HS83a] D. Harel, R. Sherman, "Looping vs. Repeating in Dynamic Logic", Technical Report CS83-02, Weizmann Institute, Rehovot, Israel, 1983.

[HS83b] D. Harel, R. Sherman, "Propositional Dynamic Logic of Flowcharts", *Proc. Int. Conf. on Foundations of Computation Theory,* Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1983.

[Ko82] D. Kozen, "Results on the Propositional μ-Calculus", *Theoretical Computer Science,* 27(1983), pp. 333-354.

[KP81] D. Kozen, R. Parikh, "An Elementary Proof of the Completeness of PDL", *Theoretical Computer Science,* 14(1), 1981, pp. 113-118.

[KP83] D. Kozen, R. Parikh, "A Decision Procedure for the Propositional μ-Calculus", *Proc. of the Workshop on Logics of Programs,* Springer-Verlag Lecture Notes in Computer Science - Vol. 164, 1984 (to appear), also IBM Research Report RC9987.

[JL77] N.D. Jones, W.T. Laaser, "Complete problems in deterministic polynomial time", *Theoretical Computer Science,* 3(1977), pp. 105-117.

[Me75] A. R. Meyer, "Weak Monadic Second Order Theory of Successor is not Elementary Recursive", *Proc. Logic Colloquium,* Lecture Notes in Mathematics - Vol. 453, Springer-Verlag, 1975, pp. 132-154.

[Pa70] D. Park, "Fixpoint Induction and Proof of Program Semantics", *Machine Intelligence 5*, (Metlzer, Mitchie, eds.), Edinburgh University Press, 1970, pp. 59-78.

[Pa76] D. Park, "Finiteness is mu-ineffable", *Theoretical Computer Science*, 3(1976), pp. 173-181.

[Par83] R. Parikh, "Propositional Game Logic", *Proc. 24th IEEE Symp. on Foundations of Computer Science*, Tucson, 1983, pp. 195-200.

[Pn77] A. Pnueli, "The Temporal Logic of Programs", *Proc. 8th IEEE Symp. on Foundations of Computer Science*, Providence, 1977, pp. 46-57.

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

[Pr79] V.R. Pratt, "Models of program logics", *Proc. 20th IEEE Symp. on Foundation of Computer Science*, San Juan, 1979, pp. 115-122.

[Pr80] V.R. Pratt, "A Near-Optimal Method for Reasoning about Action", *J. Computer and Systems Sciences* 20(1980), pp. 231-254.

[Pr81] V.R. Pratt, "A Decidable μ-Calculus", *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, Nashville, TN, October 1981, pp. 421-427.

[PS83] A. Pnueli, R, Sherman, "Propositional dynamic logic of looping flowcharts", Technical Report, Weizmann Institute, Rehovot, Israel, 1983.

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

[Re79] J. Reif, "Data Flow Analysis of Communicating Processes", *Proc. 6th ACM Symp. on Principles of Programming Languages*, San Antonio, January 1979.

[Sc71] D. Scott, "Lattice of Flow Diagrams", *Proc. Symp. on Semantics of Algorithmic Languages* (E. Engeler, Ed.), Lecture Notes in Mathematics - Vol. 188, Springer-Verlag, Berlin, 1971, pp. 311-366.

[SdB69] D. Scott, J. deBakker. "A Theory of Programs", unpublished lecture notes, Vienna, 1969.

[St81] R. Streett, "Propositional Dynamic Logic of Looping and Converse", *Proc. 13th ACM Symp. on Theory of Computing*, Milwaukee, 1981, pp. 375-383.

[Ta55] A. Tarski, "A Lattice-Theoretical Fixpoint Theorem and its Applications", *Pacific Journal of Mathematics*, 5(1955), pp. 285-309.

[TW68] J. W. Thatcher, J. B. Wright, "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic", *Mathematical System Theory, 2(1968), pp. 57-81.*