The Taming of Converse: Reasoning about Two-Way Computations

Moshe Y. Vardi[†]
Center for Study of Languages and Information, Stanford University

ABSTRACT

We consider variants of propositional dynamic logic (*PDL*) augmented with the *converse* construct. Intuitively, the converse α^- of a program α is a programs whose semantics is to run α backwards. While *PDL* consists of assertions about *weakest preconditions*, the *converse* construct enable us to make assertions about *strongest postconditions*. We investigate the interaction of *converse* with two constructs that deal with infinite computations: *loop* and *repeat*. We show that *converse* $-$ *loop* $-$ *PDL* is decidable in exponential time, and *converse* $-$ *repeat* $-$ *PDL* is decidable in nondeterministic exponential time.

1. Introduction

Propositional dynamic logic (*PDL*) [FL79], and its extensions (e.g., [HS83a], [Sh84], [St80]) are formal systems for reasoning about input/output and ongoing behavior of programs schemes. The basic constructs in these logics are assertions of the form $\langle\alpha\rangle q$, asserting that the program α can terminate in a state satisfying q, and assertions about the possibility of infinite computations, e.g., *loop*(α) [HS83a] or *repeat*(α) [St80].

The *converse* construct [Pr76] enables us to reverse our mode of reasoning. Intuitively, the converse of a program α, denoted α^-, is a program whose semantics is to run α backwards. Thus, if the assertion $\langle\alpha\rangle q$ is the weakest precondition for the program α to terminate in a state satisfying q, the assertion $\langle\alpha^-\rangle p$ is the strongest postcondition that holds after termination of α when started at a state satisfying p [dB80].

Intuitively, one would not expect the addition of *converse* to a logic L to change the properties of L in any significant way. Indeed, *converse* $-$ *PDL*, the extension of *PDL* with *converse*, satisfies the same *small model property* as *PDL* [FL79], and the known decision procedures and completeness results for *PDL* extend without difficulty to *converse* $-$ *PDL* [Pa80, Pr79, Pr80].

This turns out, however, to be the exception rather then the rule. In variants of *PDL*, *converse* interacts with other constructs in a quite unpredictable way. For example, *converse* $-$ *DPDL*, where atomic programs are required to be deterministic, does not have the *finite model property*. Indeed, the *converse* $-$ *DPDL* formula $P \wedge [a^-]\langle a^-\rangle \neg P$ is not satisfiable in any finite model though it is satisfiable in an infinite model [Harel83]. The finite model property similarly fails for *converse* $-$ *loop* $-$ *PDL* [St82]. The complications arising from the addition of *converse* seem also to make the decision problem much harder; the addition of *converse* to *repeat* $-$ *PDL* increases the running time of the decision procedure in [St82] quintuply exponentially (from $O(exp^3(n))$ to $O(exp^8(n))$).

In [VW84] it is shown that even though *converse* does make life harder, it does not make them as exorbitantly harder as it would seem from the results in [St82]. Specifically, it is shown there that validity for *converse* $-$ *DPDL* can be decided in time $O(exp(n^2))$ (the bound for *DPDL* is $O(exp(n))$[*BHP*82]). The increase in the bound is due to a quadratic increase in the size of *closures* of formulas.

† Research supported by a gift from System Development Foundation. Address: CSLI, Ventura Hall, Stanford University, Stanford, CA 94305, USA.

In this paper we show that in general *converse* behaves in the manner suggested in [VW84] rather than in the manner suggested in [St82]. That is, it does makes life harder, but not too hard. We show this by proving two upper bounds. We prove that validity for $converse-loop-PDL$ can be decided in time $O(exp(n^2))$ (the bound for $loop-PDL$ is $O(exp(n))$ [PS83]), and we prove that the validity for $converse-repeat-PDL$ can be decided in nondeterministic time $O(exp(n^4))$ (the bound for $repeat-PDL$ is $O(exp(n^2))$ [VS85]. (Actually, we prove our results for extensions of $DPDL$, from which follow the results for extensions of PDL. No previous results about $converse-loop-DPDL$ and $converse-repeat-DPDL$ were known.) As in [VW84], the increase in the bounds is due to a quadratic increase in the size of closures of formulas.

We prove our bounds by the automata-theoretic approach suggested in [St80] and further developed in [VW84] and [VS85]. This approach is based on the fact that our 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. This suggest that decision procedures for program logics can be obtained by reducing satisfiability to the *emptiness problem* for certain classes of *tree automata*. The idea is to construct a tree automaton A_f for a given a formula 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.

The reduction of satisfiability to emptiness for logic such as $DPDL$ [VW84] or $repeat-PDL$ [St80,VS84] is relatively straightforward. This is not the case once *converse* is introduced. By their nature, tree automata are one-way devices, i.e., they scan the input in one direction. Computation of programs with *converse* are, on the other hand, two-way computations, and it is not clear how to check assertions about two-way computations by one-way automata. Indeed, to solve the decision problem for $converse-repeat-PDL$, Streett introduced two-way automata [St82]. The crux of his result is that two-way automata can be transfomed to equivalent one-way automata, but the transformation causes a quadruply exponential blow-up.

What we show in this paper is how to reduce satisfiability in the presence of *converse* to emptiness of one-way automata. The key to our reduction is the extension of the closure by assertions about cycling computations. Intuitively, $cycle(\alpha)$ holds in a state u if there is a computation of α that starts and terminates at u. We show that two-way computations can be viewed as one-way computations with cycles. Thus assertions about two-way computations can be checked by one-way automata.

2. *Converse – DPDL*

We assume familiarity with *dynamic logic* [Pr76], with *propositional dynamic logic* (PDL) [FL79], and with *deterministic propositional dynamic logic* ($DPDL$) [BHP82]. We also assume familiarity with the necessary automata-theoretic background (see [VW84]). We will consider a variant of $DPDL$, in which programs are described by sequential automata rather than by regular expressions (c.f. [HS83b,Pr81]). 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 $DPDL$. In this section we describe Vardi and Wolper's approach to $converse-ADPDL$ [VW84]. In the sequel we extend this approach to $converse-loop-DPDL$ and $converse-repeat-DPDL$.

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

- every proposition $p \in Prop$ is a formula.
- if f_1 and f_2 are formulas, then $\neg f_1$ and $f_1 \wedge f_2$ are formulas.
- If f is a formula, then $f?$ is a test.
- If a is an atomic program, then a^- is a backward program.
- if α is a program and f is a formula, then $\langle\alpha\rangle f$ is a formula

- If α is a sequential automaton over an alphabet Σ, where Σ is a finite set of atomic programs, backward programs, and tests, then α is a program. (A sequential automaton is a tuple (Σ,S,ρ,s,F), where Σ is a finite alphabet, S is a finite set of states, $\rho: S\times\Sigma\to 2^S$ is a nondeterministic transition function, $s\in S$ is the starting state, and $F\subseteq S$ is a set of accepting states.) A word w accepted by α is called an *execution sequence of* α.

Let $Prog'$ be the set of atomic and backward programs.

Converse – ADPDL formulas are interpreted over structures $M=(W,R,\Pi)$ where W is a set of states, $R: Prog\to 2^{W\times W}$ is a deterministic transition relation (for each state u and atomic program a there is at most one pair $(u,u')\in R(a)$), and $\Pi: W\to 2^{Prop}$ 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:

- $R(f?)=\{(u,u): M,u\models f\}$.
- $R(a^-)=\{(v,u):(u,v)\in R(a)\}$.
- $R(\alpha)=\{(u,u')$: there exists an execution sequence $w=w_1\cdots w_n$ of α and states $u_0,u_1,\ldots,u_n$ of W such that $u=u_0$, $u'=u_n$ and for all $1\le i\le n$ we have $(u_{i-1},u_i)\in R(w_i)\}$.
- For a proposition $p\in Prop$, $M,u\models p$ iff $p\in\Pi(u)$.
- $M,u\models f_1\wedge f_2$ iff $M,u\models f_1$ and $M,u\models f_2$.
- $M,u\models\neg f_1$ iff not $M,u\models f_1$.
- $M,u\models\langle\alpha\rangle f$ iff there exists a state u' such that $(u,u')\in R(\alpha)$ and $M,u'\models f$.

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

A formula f is *satisfiable* if there is a structure M and a state u in the structure such that $M,u\models f$. The *satisfiability problem* is to determine, given a formula f, whether f is satisfiable.

To use the automata-theoretic technique, we first have to prove that *converse – ADPDL* has the *tree model property*. A *tree structure* for a formula f is a structure $M=(W,R,\Pi)$ such that:

1) $W\subseteq[n]^*$, where n is bounded by the length of f and $W\neq\emptyset$.
2) $xi\in W$ only if $x\in W$.
3) $(x,y)\in R(a)$ for an atomic or a backward program a only if x is the predecessor or the successor of y and $(x,y)\notin R(b)$ for any other atomic or backward program b.

A tree structure $M=(W,R,\Pi)$ is a *tree model* for f if $M,\lambda\models f$ (note that since $W\neq\emptyset$, $\lambda\in W$).

Proposition 2.1: [VW84] *Converse – ADPDL* has the tree model property. ■

In tree models for *ADPDL* eventualities are accomplished by "downward" paths. That is, if $\langle\alpha\rangle g$ is satisfied in a state x, then the sequence of states that leads to a state that satisfies g is of the form $x, xi_1, xi_1i_2, xi_1i_2i_3,\cdots$. Thus an automaton that checks for satisfaction of eventualities only needed to go down the tree (we view the trees as growing downwards). In the presence of the *converse* construct, however, eventualities may require "two-way paths". Indeed, in [St82] *two-way automata* are defined in order to deal with *converse*. Unfortunately, the way the emptiness problem is solved for these automata is to convert them to one-way automata with a fourfold exponential increase in the number of states.

To avoid this difficulty we extend the logic by adding formulas that deal with "cycling" computations. If α is a program, then *cycle*(α) is a formula. Let $M=(W,R,\Pi)$ and $u\in W$, then $M,u\models$ *cycle*(α) if $(u,u)\in R(\alpha)$. That is, *cycle*(α) holds in the state u if there is a computation of α that starts and terminates at u. Note that we do not consider cycle formulas as formulas of *converse – ADPDL*, but they will be helpful in the decision procedure, because they enable us to check eventualities using "one-way" automata. (It is interesting to note that the extended logic, i.e., the logic that contains also the cycle formulas, is undecidable [Da84]).

It is not clear yet how cycle formulas help us solve our problem. Furthermore, how are we going to check satisfaction of cycle formulas by one-way automata? The solution is to add *directed* cycle formulas, in which the direction of the computation is specified. The semantics of such formulas is defined only on tree structures. If α is a program, then both $cycle_d(\alpha)$ and $cycle_u(\alpha)$ are formulas. We call these formulas *directed cycle formulas*. Formulas of the first type are called *downward cycle formulas*, and formulas of the second type are called *upward cycle formulas*. We now define the semantics of these formulas.

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

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

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

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

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

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

The relationship between the various *cycle* formulas is expressed in the following proposition. Let $\alpha=(\Sigma,S,\rho,s_0,F)$ be a program, and let $p,q\in S$. We denote by the program (Σ,S,ρ,p,F) by α_p, $(\Sigma,S,\rho,s,\{q\})$ by α^q, and $(\Sigma,S,\rho,p,\{q\})$ by α_p^q. As we shall see later, when dealing with cycle formulas it suffices to consider programs of the form α_p^q.

Proposition 2.2: [VW84] Let $M=(W,R,\Pi)$ be a tree structure, let $x\in W$, and let $\alpha=(\Sigma,S,\rho,s,\{t\})$ be a program. Then $M,x\models cycle(\alpha)$ if and only if there are states $s_{j_1},\ldots,s_{j_k}$ in S, where $1\leq k\leq |S|$, such that $s_{j_1}=s$, $s_{j_k}=t$, and for all $1\leq i\leq k-1$, if $p=s_{j_i}$ and $q=s_{j_{i+1}}$, then $M,x\models cycle_u(\alpha_p^q)$ or $M,x\models cycle_d(\alpha_p^q)$. ■

As with cycle formulas, we distinguish between downward and upward accomplishment of eventualities. We therefore introduce two new types of formulas, whose semantics is defined only on tree structures. If α is a program and g is a formula, then both $\langle\alpha\rangle_d g$ and $\langle\alpha\rangle_u g$ are formulas. We call these formulas *directed eventualities*. Formulas of the former type are called *downward eventualities*, and formulas of the latter type are called *upward eventualities*. We now define the semantics of directed eventualities.

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

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

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

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

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

Proposition 2.3: [VW84] Let $M=(W,R,\Pi)$ be a tree structure, let $x\in W$, let α be a program, and let g be a formula. Then $M,x\models\langle\alpha\rangle g$ if and only if either $M,x\models\langle\alpha\rangle_d g$ or $M,x\models\langle\alpha\rangle_u g$. ■

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

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

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

To establish a decision procedure for *ADPDL*, we reduce the satisfiability problem to the emptiness problem for Büchi automata. To this end we associate an infinite n-ary tree over $2^{ecl(f)\cup Prog'\cup\{\perp\}}$ with the tree model $M'=(W',R',\Pi')$ constructed above in a natural way: every node in W' is labeled by the formulas in $ecl(f)$ that are satisfied at that node, and the other nodes are labeled by the special symbol $\perp$. We also label nodes by atomic programs, and the labeling is to be interpreted as follows: if a node x is labeled by atomic program b and the predecessor of x is y, then $(x,y)\in R(b)$. Note that a node cannot be labeled by more than one atomic program. Trees that correspond to tree models satisfy some special properties.

A Hintikka tree for a *converse – ADPDL* formula f is an n-ary tree $T:[n]^*\rightarrow 2^{ecl(f)\cup Prog'\cup\{\perp\}}$ that satisfies the following *Hintikka conditions*:

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

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

2)

2.1) $|T(x)\cap Prog'|\leq 1$,

2.2) if y,z are two distinct successors of x, $a\in Prog$, and $a^-\in T(y)$, then $a^-\notin T(z)$,

2.3) if y is a successor of x, $a\in Prog$, and $a\in T(x)$, then $a^-\notin T(y)$,

3)

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

3.2) $g_1/\backslash g_2\in T(x)$ iff $g_1\in T(x)$ and $g_2\in T(x)$,

4) if $\alpha=(\Sigma,S,\rho,s,\{t\})$ is a program, then

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

4.2) if y is the predecessor of x, then $cycle_u(\alpha)\in T(x)$ if and only if there are states $p,q\in S$ and a $b\in Prog'$ such that

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

4.3) $cycle_d(\alpha)\in T(x)$ if either there is a test $g?$ such that $g\in T(x)$ and $t\in\rho(g?,s)$, or there are states $s_1,\ldots,s_m$ in S, $1\leq m\leq|S|$, a program $b\in Prog'$ and a successor y of x such that

- $b^-\in T(y)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}})\in T(y)$ for all $1\leq i\leq m-1$,
- $s_1\in\rho(s,b)$ and $t\in\rho(s_m,b^-)$,

4.4) $cycle_d(\alpha)\in T(x)$ only if there is a finite subset $W'\subseteq[n]^*$ with $x\in W'$ and a mapping $\varphi:W'\to 2^{ecl(f)}$ such that $cycle_d(\alpha)\in\varphi(x)$, and if $y\in W'$ and $cycle_d(\alpha_p^q)\in\varphi(y)$, then either there is a test $g?$ such that $g\in T(y)$ and $p\in\rho(q,g?)$, or there are states $s_1,\ldots,s_m$ in S, $1\leq m\leq|S|$, a program $b\in Prog'$ and a successor $z\in W'$ of y such that

- $b^-\in T(z)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}})\in\varphi(z)$ for all $1\leq i\leq m-1$,
- $s_1\in\rho(p,b)$ and $q\in\rho(s_m,b^-)$.

5) if $\alpha=(\Sigma,S,\rho,s,F)$ is a program and g is a formula, then

5.1) $\langle\alpha\rangle g\in T(x)$ if and only if either $\langle\alpha\rangle_d g\in T(x)$ or $\langle\alpha\rangle_u g\in T(x)$.

5.2) if y is the predecessor of x, then $\langle\alpha\rangle_u g\in T(x)$ if and only if there are states $p,q\in S$ and a program $b\in Prog'$, such that

- $cycle(\alpha_s^p)\in T(x)$,
- $q\in\rho(p,b)$,
- $b\in T(x)$,
- $\langle\alpha_q\rangle g\in T(y)$,

5.3) $\langle\alpha\rangle_d g\in T(x)$ if either $cycle(\alpha)\in T(x)$ and $g\in T(x)$, or there are states $p,q\in S$, a program $b\in Prog'$, and a successor y of x, such that

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

5.4) $\langle\alpha\rangle_d g\in T(x)$ only if there are nodes $x_0,\ldots,x_k$, states $s_0,t_0,\ldots,s_k,t_k$ of S, and programs $b_1,\ldots,b_k\in Prog'$ such that

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

Proposition 2.4: [VW84] A *converse – ADPDL* formula f has a tree model if and only if it has a Hintikka tree. ■

It remains now to construct a Büchi tree automaton A_f that accepts precisely the Hintikka trees for f. This is described in [VW84]. A_f has $O(\exp(n^2))$ states, where n is the length of f. This yields a decision procedure whose running time is $O(\exp(n^2))$.

3. *Converse – loop – ADPDL*

Converse – ADPDL is a logic to reason about input/output behavior of programs. This is not adequate for reasoning about the behavior of nonterminating programs such as operating systems. To this end we extend the logic by constructs that deal with infinite computations. One such construct is the *loop* construct [HS83a]. Intuitively, the formula $loop(\alpha)$ holds in a state if there is an infinite computation of α from that state. Sherman and Pnueli have shown that *loop – PDL* is decidable in exponential time [PS83], and Vardi and Wolper have shown by automata-theoretic techniques that *loop – DPDL* is also decidable in exponential time [VW84]. We now show how the automata-theoretic framework can be extended to *converse – loop – ADPDL*, for which no previous results were known.

Formally, we get *converse – loop – ADPDL* by extending the definition of of *converse – ADPDL* by the following syntactic and semantic clauses.

- If α is a program, then $loop(\alpha)$ is a formula.
- $M,u\models loop(\alpha)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, iff there are an infinite word $w=w_1w_2\cdots$ over Σ, an infinite sequence $s_0,s_1,\cdots$ of states of S, and an infinite sequence $u_0,u_1,\cdots$ of nodes of W such that:
  - $u_0=u$, $s_0=s$ and
  - for all $i\geq 1$, $s_i\in\rho(s_{i-1},w_i)$ and $(u_{i-1},u_i)\in R(w_i)$.

Proposition 3.1: *Converse – loop – ADPDL* has the tree model property. ■

As with eventualities, we distinguish between downward and upward fulfillment of loop formulas. We introduce two new types of formulas, whose semantics is defined only on tree structures. If α is a program, then both $loop_d(\alpha)$ and $loop_u(\alpha)$ are formulas. We call these formulas *directed loop formulas*. Formulas of the former type are called *downward loop formulas*, and formulas of the latter type are called *upward loop formulas*. We now define the semantics of directed loop formulas.

Let $M=(W,R,\Pi)$ be a tree structure, and let $x\in W$. We have that $M,x\models loop_d(\alpha)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, iff there are an infinite word $w=w_1w_2\cdots$ over Σ, an infinite sequence $s_0,s_1,\cdots$ of states of S, and an infinite sequence $x_0,x_1,\cdots$ of nodes of W such that:

- $x_0=x$, $s_0=s$ and
- for all $i\geq 1$, $s_i\in\rho(s_{i-1},w_i)$ and $(x_{i-1},x_i)\in R(w_i)$.
- if there some $i\geq 0$ such that $x_i=x$ and $x_j\neq x$ for all $j>i$, then x_j properly succeeds x for all $j>i$.

Let $M=(W,R,\Pi)$ be a tree structure, and let $x\in W$. We have that $M,x\models loop_u(\alpha)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, iff there are an infinite word $w=w_1w_2\cdots$ over Σ, an infinite sequence $s_0,s_1,\cdots$ of states of S, and an infinite sequence $x_0,x_1,\cdots$ of nodes of W such that:

- $x_0=x$, $s_0=s$ and

- for all $i \geq 1$, $s_i \in \rho(s_{i-1}, w_i)$ and $(x_{i-1}, x_i) \in R(w_i)$.
- there is some $i \geq 1$ such that x_i is the predecessor of x.

Note that a downward loop formula actually requires that the computation eventually goes downward (unless it loops forever on the same state), while an upward eventuality does not require that the computation eventually goes upward. Also note that a loop formula can be satisfied both upwards and downwards. The relationship between the various types of loop formulas is expressed in the next proposition.

Proposition 3.2: [VW84] Let $M = (W, R, \Pi)$ be a tree structure, let $x \in W$, and let α be a program. Then $M, x \models loop(\alpha)$ if and only if either $M, x \models loop_d(\alpha)$ or $M, x \models loop_u(\alpha)$. ■

To deal with *loop* formulas, we extend the definition of the extended closure by the following clause:

- If $loop(\alpha) \in ecl(f)$, where $\alpha = (\Sigma, S, \rho, s_0, F)$, then $loop(\alpha_s), loop_d(\alpha_s), loop_u(\alpha_s) \in ecl(f)$ for all $s \in S$.

The size of $ecl(f)$ is of course still at most quadratic in the length of f.

We can now define Hintikka trees for $converse - loop - ADPDL$ formulas.

A Hintikka tree for a $converse - loop - ADPDL$ formula f is an n-ary tree $T: [n]^* \to 2^{ecl(f) \cup Prog' \cup \{\sqcup\}}$ that satisfies Hintikka conditions 1-5 and also

6) if $\alpha = (\Sigma, S, \rho, s, F)$ is a program then

6.1) $loop(\alpha) \in T(x)$ if and only if either $loop_d(\alpha) \in T(x)$ or $loop_u(\alpha) \in T(x)$.

6.2) if y is the predecessor of x, then $loop_u(\alpha) \in T(x)$ if and only if there are states $p, q \in S$ and an atomic program b, such that

- $cycle(\alpha_s^p) \in T(x)$,
- $q \in \rho(p, b)$,
- $b \in T(x)$,
- $loop(\alpha_q) \in T(y)$,

6.3) if $loop_d(\alpha) \in T(x)$, where $\alpha = (\Sigma, S, \rho, s, F)$, then there exists a state $p \in S$ such that either

a) $cycle(\alpha_s^p) \in T(x)$ and $cycle(\alpha_p^p) \in T(x)$, or

b) there are a state $q \in S$, a program $b \in Prog'$, and successor y of x such that

- $cycle(\alpha_s^p) \in T(x)$,
- $q \in \rho(p, b)$,
- $b^- \in T(y)$,
- $loop_d(\alpha_q) \in T(y)$.

6.4) if $\neg loop_d(\alpha) \in T(x)$ then, there is a finite subset $W' \subseteq [n]^*$ with $x \in W'$ and a mapping $\varphi: W' \to 2^{ecl(f)}$ such that $\neg loop_d(\alpha) \in \varphi(x)$, and if $y \in W'$ and $\neg loop_d(\alpha_p) \in \varphi(y)$, then

- there is no state $p \in S$ such that $cycle(\alpha_s^p) \in T(y)$ and $cycle(\alpha_p^p) \in T(y)$,
- if for some states $p, q \in S$, $b \in Prog'$, and z successor of y we have that
  - $cycle(\alpha_s^p)$,
  - $q \in \rho(p, b)$, and
  - $b^- \in T(z)$,

  then $z \in W'$ and $\neg loop_d(\alpha_q) \in T(z)$.

Proposition 3.3: A $converse - loop - ADPDL$ formula f has a tree model if and only if it has a Hintikka tree. ■

It remains now to construct a Büchi tree automaton A_f that accepts precisely the Hintikka trees for f. The method is that of [VW84]. A_f has $O(\exp(n^2))$ states, where n is the length of f. This yields a decision procedure whose running time is $O(\exp(n^2))$.

4. *Converse – repeat – ADPDL*

Another construct that deal with infinite computations is the *repeat* construct [St80] (*repeat* is denoted by Δ in [St80]). Intuitively, the formula $repeat(\alpha)$ is true in a state if there is a way to repeatedly execute α without stopping. It is known that the construct *repeat* is strictly more powerful than the construct *loop* [HS83a]. The addition of *repeat* to *PDL* seems to make the decision problem quite harder. The best known upper bound for *repeat* – *(D)PDL* is nondeterministic time $O(exp(n^2))$ [VS85]. We now show how to extend our technique to *converse – repeat – ADPDL* for which no previous results were known.

Formally, we get *converse – repeat – ADPDL* by extending the definition of *converse – ADPDL* by the following syntactic and semantic clauses:

- If α is a program, then $repeat(\alpha)$ is a formula.
- $M,u \models repeat(\alpha)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, iff there are an infinite sequence $w_1,w_2\cdots$ of execution sequences of α, and an infinite sequence $u_0,u_1,\cdots$ of nodes of W such that: $u_0=u$ and $(u_{i-1},u_i)\in R(w_i)$ for all $i\geq 1$.

Rather than deal with *converse – repeat – ADPDL*, we deal with an equivalent logic, *converse – Büchi – ADPDL*. The latter logic has infinite programs described by Büchi automata.

Formally, we get *converse – Büchi – ADPDL* by extending the definition of of *converse – ADPDL* by the following syntactic and semantic clauses.

- If α is a program, then $\ll\alpha\gg$ is a formula.
- $M,u \models \ll\alpha\gg$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, iff there an infinite word $w=w_1w_2\cdots$ over Σ, an infinite sequence $s_0,s_1,\cdots$ of states of S, and an infinite sequence $u_0,u_1,\cdots$ of nodes of W such that:
  - $u_0=u$ and $s_0=s$,
  - for some $s\in F$ we have $|\{i : s_i=s\}|=\omega$ (i.e., some state in F occurs infinitely often in the sequence $s_0,s_1,\cdots$), and
  - for all $i\geq 1$, $s_i\in\rho(s_{i-1},w_i)$ and $(u_{i-1},u_i)\in R(w_i)$.

Note that the semantics of $\ll\alpha\gg$ is very closed to the semantics of $loop(\alpha)$; the only difference is the additional requirement that some state in F repeats infinitely often. This condition is essentially Büchi acceptance condition for automata on infinite words [Bu62].

Proposition 4.1. There is linear translation from *converse – repeat – ADPDL* to *converse – Büchi – ADPDL*. Namely, there is a logspace mapping γ such that if φ is a *converse – repeat – ADPDL* formula, then $\gamma(\varphi)$ is a *converse – Büchi – ADPDL* formula, $|\gamma(\varphi)|=O(|\varphi|)$, and φ is logically equivalent to $\gamma(\varphi)$. Similarly, there is quadratic translation from *converse – Büchi – ADPDL* to *converse – repeat – ADPDL*. ■

Thus it suffices to consider *converse – Büchi – ADPDL*.

Proposition 4.2: *Converse – Büchi – ADPDL* has the tree model property. ■

It turns out that cycle formulas, and even directed cycle formulas, are not sufficient to enable us to deal with tree models for *converse – Büchi – ADPDL* by one-way automata. What we need is to strengthen our cycle formulas in the following way. If $\alpha=(\Sigma,S,\rho,s,F)$ is a program, then $scycle(\alpha)$ is a formula. Let $M=(W,R,\Pi)$ and $u\in W$, then $M,u \models scycle(\alpha)$ if there are a sequence $s_0,\ldots,s_k$, of states in S, an execution sequence $w_1\cdots w_k$ of α, and a sequence $u_0,\ldots,u_k$ of nodes in W, $k\geq 1$, such that $s_0=s$, $u_0=u_k=u$, $s_i\in\rho(s_{i-1},w_i)$ and $(u_{i-1},u_i)\in R(w_i)$ for all $1\leq i\leq k$, and $s_i\in F$ for *some* $1\leq i\leq k$. That is, $scycle(\alpha)$ holds in the state u if there is a

computation of α that starts and terminates at u and goes through a state in F.

We now can strengthen also directed cycle formulas in an analogous way. In fact the whole treatment of cycle formulas [VW84] can be strengthened in a straightforward way to deal with the requirement that the cycling computations go through designated states. To deal with $\langle\langle\rangle\rangle$ formulas, we extend the definition of extended closure by the following clauses:

- If $\ll\alpha\gg g \in ecl(f)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, then $scycle(\alpha_s^t)$, $scycle_d(\alpha_s^t)$, $scycle_u(\alpha_s^t)\in ecl(f)$ for all $s,t\in S$.
- If $\ll\alpha\gg\in ecl(f)$, where $\alpha=(\Sigma,S,\rho,s_0,F)$, then $\langle\alpha^t\rangle\ll\alpha_t\gg\in ecl(f)$ for all $t\in S$.

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

We can now define Hintikka trees for *converse* $-$ *Büchi* $-$ *ADPDL* formulas.

A Hintikka tree for a *converse* $-$ *Büchi* $-$ *ADPDL* formula f is an n-ary tree $T:[n]^*\to 2^{ecl(f)\cup Prog'\cup\{\Box\}}$ that satisfies Hintikka conditions 1-5 and also

7) if $\alpha=(\Sigma,S,\rho,s,\{t\})$ is a program, then

7.1) $scycle(\alpha)\in T(x)$ if and only if there are states $s_0,\ldots,s_m$ in S, where $1\le m\le|S|$, such that

- $s_0=s$, $s_m=t$,
- for all $0\le i\le m-1$ either $cycle_u(\alpha_{s_i}^{s_{i+1}})\in T(x)$ or $cycle_d(\alpha_{s_i}^{s_{i+1}})\in T(x)$,
- for some $0\le i\le m-1$ either $scycle_u(\alpha_{s_i}^{s_{i+1}})\in T(x)$ or $scycle_d(\alpha_{s_i}^{s_{i+1}})\in T(x)$,

7.2) if y is the predecessor of x, then $scycle_u(\alpha)\in T(x)$ if and only if there are states $p,q\in S$ and a program $b\in Prog'$ such that

- $b\in T(x)$,
- $cycle(\alpha_p^q)\in T(y)$, and if $p,q\notin F$ then $scycle(\alpha_p^q)\in T(y)$,
- $p\in\rho(s,b)$ and $t\in\rho(q,b^-)$,

7.3) $scycle_d(\alpha)\in T(x)$ if either there is a test $g?$ such that $g\in T(x)$ and $t\in\rho(g?,s)\cap F$, or there are states $s_1,\ldots,s_m$ in S, $1\le m\le|S|$, a program $b\in Prog'$, and a successor y of x such that

- $b^-\in T(y)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}})\in T(y)$ for all $1\le i\le m-1$, and $scycle_d(\alpha_{s_i}^{s_{i+1}})\in T(y)$ for some $1\le i\le m-1$,
- $s_1\in\rho(s,b)$ and $t\in\rho(s_m,b^-)$,

7.5) $scycle_d(\alpha)\in T(x)$ only if there is a finite subset $W'\subseteq[n]^*$ with $x\in W'$ and a mapping $\varphi:W'\to 2^{ecl(f)}$ such that $cycle_d(\alpha)\in\varphi(x)$, and if $y\in W'$ and $cycle_d(\alpha_p^q)\in\varphi(y)$, then either there is a test $g?$ such that $g\in T(y)$ and $p\in\rho(q,g?)\cap F$, or there are states $s_1,\ldots,s_m$ in S, $1\le m\le|S|$, a program $b\in Prog'$ and a successor $z\in W'$ of y such that

- $b^-\in T(z)$,
- $cycle_d(\alpha_{s_i}^{s_{i+1}})\in\varphi(z)$ for all $1\le i\le m-1$, and $cycle_d(\alpha_{s_i}^{s_{i+1}})\in\varphi(z)$ for some $1\le i\le m-1$,
- $s_1\in\rho(p,b)$ and $q\in\rho(s_m,b^-)$.

8) if $\alpha=(\Sigma,S,\rho,s,F)$ is a program then

8.1) $\ll\alpha\gg\in T(x)$ iff $\langle\alpha^t\rangle\ll\alpha_t\gg\in T(x)$ for some $t\in S$,

8.2) $\ll\alpha\gg\in T(x)$ if there there is a state $t\in S$ such that $cycle(\alpha_s^t)\in T(x)$ and $scycle(\alpha_t^t)\in T(x)$,

8.3) $\ll\alpha\gg\in T(x)$ if there are an infinite sequences $x_0,x_1,\cdots$ of nodes in $[n]^*$, infinite sequences $s_0,s_1,\cdots$ and $t_0,t_1,\ldots$ of states in S, and an infinite sequence $b_0,b_1,\cdots$ of programs in $Prog'$ such that

- $x_0 = x$ and x_{i+1} is a successor of x_i for all $i \geq 0$,
- $s_0 = s$, $cycle(\alpha_{s_i}^{t_i}) \in T(x_i)$, $s_{i+1} \in \rho(t_i, b_i)$, and $b_i^- \in T(x_{i+1})$ for all $i \geq 0$,
- there is a sequence $0 \leq i_0 < i_1, \cdots$ such that if $p = s_{i_j}$ and $q = t_{i_j}$ then $scycle(\alpha_q^p) \in T(x_{i_j})$ for all $j \geq 0$.

Theorem 4.3: A *converse – Büchi – ADPDL* f has a tree model if and only if it has a Hintikka tree. ■

It seems that all that remains now is to construct a Büchi automaton that accepts precisely the Hintikka trees for f. Unfortunately, this is impossible (the impossibility follows from results by Rabin [Ra70].) The difficulty comes from condition 8.3, since, using the techniques of [VW84], it is not hard to construct a Büchi automaton that check the other conditions. Thus, rather than use Büchi automata to accept Hintikka trees for *converse – repeat – ADPDL*, we have to use the more powerful *hybrid* automata of Vardi and Stockmeyer [VS85].

A *hybrid* tree automaton H is a pair (A, B), where A is a Rabin tree automaton and B is a Büchi sequential automaton, both over the same alphabet Σ. H *accepts* a tree T if T is accepted by A and, for every infinite path P starting at λ, B rejects the infinite word $T(P)$. We need not concern ourselves here with Rabin automata; it suffices to say that every Büchi automaton can be viewed as a Rabin automaton. The key fact about hybrid automata, proven in [VS85], is that 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 ($O(2n^2)$) in the size of B.

To construct a hybrid automaton that accepts that Hintikka trees of f, we construct a Büchi tree automaton A_f that check all the conditions except for 8.3, and we construct a Büchi sequential automaton B_f that checks for *violations* of condition 8.3. The hybrid automaton $H_f = (A_f, B_f)$ accepts precisely the Hintikka trees of f. While A_f has $O(2^{n^2})$ states, B_f has only $O(n^2)$ states. This yields a decision procedure that runs in nondeterministic time $O(\exp(n^4))$.

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.

[Bu62] J.R. Büchi, "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.

[Da84] R. Danecki, *"Propositional Dynamic Logic with Strong Looping Predicate"*, 1984.

[dB80] J. de Bakker, *Mathematical theory of program correctness*, Prentice hall, 1980.

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

[Ha83] J.Y. Halpern, private communication, 1983.

[HS83a] D. Harel, R. Sherman, "Looping vs. Repeating in Dynamic Logic", *Information and Control* 55(1982), pp. 175-192.

[HS83b] D. Harel, R. Sherman, "Propositional Dynamic Logic of Flowcharts", *Proc. Int. Conf. on Foundations of Computation Theory*, Lecture Notes in Computer Science, vol. 158, Springer-Verlag, Berlin, 1983, pp. 195-206.

[Pa80 Parikh, R.: A completeness result for PDL. *Symp. on Math. Foundations of Computer Science, Zakopane, 1978.*

[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, "Using Graphs to understand PDL", *Proc. Workshop on Logics of Programs*, (D. Kozen, ed.), Yorktown-Heights, Lecture Notes in Computer Science, vol. 131, Springer-Verlag, Berlin, 1982, pp. 387-396.

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

[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.

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

[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.

[VS85] M.Y. Vardi, L. Stockmeyer, "Improved Upper and Lower Bounds for Modal Logics of Programs", To appear in *Proc. 17th ACM Symp. on Theory of Computing*, Providence, May 1985.

[VW84] M. Y. Vardi, P. Wolper, "Automata Theoretic Techniques for Modal Logics of Programs", IBM Research Report, October 1984. A preliminary version appeared in *Proc. ACM Symp. on Theory of Computing*, Wahington, April 1984, pp. 446-456.