INFORMATION AND CONTROL 54, 121–141 (1982)

Propositional Dynamic Logic of Looping and Converse Is Elementarily Decidable*

ROBERT S. STREETT

Computer Science Department,
Boston University, Boston, Massachusetts 02115

Propositional dynamic logic is a formal system for reasoning about the before–after behavior of regular program schemes. An extension of propositional dynamic logic which includes both an infinite looping construct and a converse or backtracking construct is considered and it is proved that the satisfiability problem for this logic is elementarily decidable. In order to establish this result, deterministic two-way automata on infinite trees are defined, and it is shown how they can be simulated by nondeterministic one-way automata. The satisfiability problem for propositional dynamic logic of looping and converse is then reduced to the emptiness problem for these two-way automata.

1. INTRODUCTION

Dynamic logic (Pratt, 1976, 1979; Harel, 1979; Fischer and Ladner, 1979) applies concepts from modal logic to a relational semantics of programs to yield various systems for reasoning about the before-after behavior of programs. Analogous to the modal logic assertions $\Diamond p$ (possibly p) and $\Box p$ (necessarily p) are the dynamic logic constructs $\langle a \rangle p$ and $[a]p$. If a is a program and p is an assertion about the state of a computation, then $\langle a \rangle p$ asserts that after executing a, p can be the case, and $[a]p$ asserts that after executing a, p must be the case.

A dynamic logic includes both a programming language for representing programs and an assertion language for expressing properties of computation states; different dynamic logics result from the selection of different programming and assertion languages. The underlying assertion language of propositional dynamic logic or PDL (Fischer and Ladner, 1979) is the

* This research was originally undertaken for the author's doctoral thesis at the Massachusetts Institute of Technology under the supervision of Professor Albert R. Meyer; it was supported in part by NSF Grant MCS 79 10261. The author wishes to thank Professor Meyer for his generous guidance and support and for originally suggesting the problem. The results of this paper have been presented at the Thirteenth ACM Symposium on the Theory of Computing.

0019-9958/82 $2.00

propositional calculus; its programming language consists of regular expressions over uninterpreted program labels, i.e., the programming primitives are black box programs, and more complicated programs are built up using the nondeterministic control structures of sequencing, choosing, and iterating.

Although PDL can express many interesting properties of programs, Pratt has shown that it is not powerful enough to capture the notion of infinite looping in regular programs (Pratt, 1978). However, by adding a natural formula construct to PDL, we obtain a programming logic strong enough to express a useful propositional notion of infinite looping. The resulting logic is also strong enough to express all formulae of two other propositional logics of programs: Propositional Algorithmic Logic (Mirkowska, 1980) and Temporal Logic of Branching Time (Ben-Ari, Manna, and Pnueli, 1981).

A striking feature of PDL is that it satisfies the following finite model property: an arbitrary (perhaps infinite) model of a PDL formula p can be reduced to a small finite model of p by merging those states which satisfy exactly the same subformulae of p. This property plays a key role in the known decision procedures for PDL (Fischer and Ladner, 1979; Pratt, 1979). This technique does not extend to PDL with looping, since there is a formula which is satisfiable in an infinite model which cannot be reduced to a finite model by merging states. This formula is therefore not equivalent to any PDL formula, and so PDL with looping is strictly more expressive than PDL. Nevertheless, PDL with looping is decidable and does satisfy a finite model property (Streett, 1980, 1982).

Pratt's original formulation of dynamic logic included the programming construct converse (Pratt, 1976). Given a program a, the converse of a is the program which runs a backwards, i.e., which undoes the computation performed by a. PDL with converse satisfies the same finite model property as PDL and the known decision procedures for PDL extend without difficulty to include the converse construct (Fischer and Ladner, 1979; Pratt, 1979).

Looping and converse interact to make PDL with looping and converse significantly different from either PDL with just looping or PDL with just converse. The resulting logic does not satisfy the finite model property: there is a formula satisfiable in an infinite model but not in any finite model. This proves that PDL with both constructs is strictly more expressive than either sublogic. The failure of a logic to satisfy the finite model property is often taken as an indication of undecidability, but in this case the evidence is misleading; PDL with looping and onverse is in fact elementarily decidable.

There is a straightforward proof of the decidability of PDL with looping by embedding it into SnS, the second order theory of several successors (Streett, 1979). (This method was used by Parikh (1978) to prove the decidability of his Second Order Acyclic Process Logic or SOAPL.) The

upper bound on complexity obtained in this way is not elementary, since *SnS* cannot be decided in elementary time (Meyer, 1974). In any case, there does not appear to be a straightforward translation of the converse construct into *SnS*.

Models of formulae of either SOAPL or PDL with looping can be viewed as labelled graphs. These graphs can be unravelled or unwound into tree-structured models in which programs conform to the tree structure, i.e., programs connect nodes only to their descendants in the tree. The translation of these logics into *SnS* depends crucially on this fact. The decidability of *SnS* can be established via a reduction to the emptiness problem of automata on infinite trees (Rabin, 1969). An elementary time decision procedure of PDL with looping can be obtained by directly reducing satisfiability to this emptiness problem, bypassing the translation in *SnS* (Streett 1980, 1982). The reduction involves the construction, for each formula p, of an automaton which accepts, in a sense, models of p. It follows by automata theoretic arguments that every satisfiable formula has a finitely generable model, i.e., a model obtained by unravelling a finite graph. It is not difficult to show that this finite graph is itself a model, so that the introduction of the looping construct does not destroy the finite model property (Streett, 1980, 1982).

Models of formulae of PDL with looping and converse are also labelled graphs and these graphs can also be unwound into tree-structured models. However, unlike the tree models for the previous logics, the presence of the converse construct destroys the relationship between programs and tree structure; programs can link arbitrary nodes of the tree. The presence of such programs prevents a straightforward reduction of PDL with looping and converse to the emptiness problem for automata on infinite trees. However, the semantics of the converse construct suggests a definition of deterministic two-way automata on infinite trees such that the satisfiability problem for the extended logic is reducible to the emptiness problem for these newly defined automata. Decidability of the logic follows from a reduction of the two-way emptiness problem to the ordinary or one-way emptiness problem.

Although there is no finite model property, the models of a formula of PDL with looping and converse are recognizable by a finite automaton. As before, it follows that every satisfiable formula has a finitely generable model, i.e., a model obtained by unravelling a finite graph. Although in general this finite graph is not a model of the original formula, it is a finite representation of a model. This clarifies why the logic is decidable.

2. Syntax, Semantics, and Expressive Power

We let $A, B, C,...,$ denote elements from an infinite set of *atomic programs* and $P, Q, R,...,$ denote elements from an infinite set of *atomic formulae.*

Definition 2.1. The set of programs and the set of formulae are then defined inductively as follows:

(a) Atomic programs are programs.

(b) If a, b are programs, then $a; b, a \cup b, a^*, a^-$ are programs.

(c) Atomic formulae are formulae.

(d) If p, q are formulae, then $\neg p$, $p \vee q$ are formulae.

(e) If a is a program and p is a formula, then $\langle a \rangle p$, Δa are formulae.

The formulae and programs of the original formulation of PDL are those containing no occurrence of the looping construct Δa nor any occurrence of the converse construct a^-.

Definition 2.2. A structure is a triple $S = \langle U, \models_S, <>_S \rangle$, where

(a) U is a nonempty set, the universe of states,

(b) $\models_S$ is a satisfiability relation on the atomic propositions,

(c) $<>_S$ assigns binary relations on states to the atomic programs.

Definition 2.3. Given a structure S, $\models_S$, and $<>_S$ can be extended to arbitrary formulae and programs as follows:

(a) $u \models_S \neg p$ iff not $u \models_S p$,

(b) $u \models_S p \vee q$ iff $u \models_S p$ or $u \models_S q$,

(c) $u \models_S \langle a \rangle p$ iff $\exists v.\ u <a>_S v \ \&\ v \models_S p$,

(d) $u \models_S \Delta a$ iff $\exists u_0, u_1,...$ such that $u_0 = u$ and $\forall n \geqslant 0.\ u_n <a>_S u_{n+1}$.

(e) $u <a; b>_S v$ iff $\exists w.\ u <a>_S w$ and $w <b>_S v$,

(f) $u <a \cup b>_S v$ iff $u <a>_S v$ or $u <b>_S v$,

(g) $u <a^*>_S v$ iff $\exists n \geqslant 0.\ \exists u_0,..., u_n$ such that $u_0 = u$ and $u_n = v$ and $u_0 <a>_S u_1$, $u_1 <a>_S u_2,..., u_{n-1} <a>_S u_n$

(h) $u <a^->_S v$ iff $v <a>_S u$.

If a and b are programs, then $a; b$ is the program which executes first a, then b. The programming connectives $\cup$ and $*$ are nondeterministic; if a and b are programs, then $a \cup b$ is a program which permits a choice of either a or b, and a^* is a program which permits a choice of some number (possibly zero) of iterations of a. If a is a program, then a^- is the converse of a, i.e., it

undoes the computations performed by a (however, since a can take several input states to the same output state, doing a followed by a^- can result in a change of state). If a is a program, then Δa is a formula which is true whenever there is a way to repeatedly execute the program a ad infinitum.

Additional Boolean operations can be defined as abbreviations: $p \wedge q =_{\text{df}} \neg(\neg p \vee \neg q)$, $p \to q =_{\text{df}} \neg p \vee q$, $p \leftrightarrow a =_{\text{df}} (p \to q) \wedge (q \to p)$. If a is a program and p is a formula, then $a \to p$ or $[a]p$, the weakest precondition of a with respect to p, is a formula characterizing exactly those states from which all terminating computations of a lead to final states satisfying p. Weakest preconditions can be defined as follows: $[a]p =_{\text{df}} \neg\langle a\rangle \neg p$. If a is a program and p is a formula, then $p \leftarrow a$, the strongest postcondition of a with respect to p, is a formula characterizing exactly those states which can be reached, via a computation of a, from an initial state satisfying p (de Bakker, 1980). The converse construct can be used to define strongest postconditions as follows: $p \leftarrow a =_{\text{df}} \langle a^-\rangle p$. The formula Δa indicates that the program a^* can diverge, i.e., enter a nonhalting computation. A formula ∞a, which is true of a program a exactly when a can enter a nonhalting computation or infinite loop, can be defined inductively as follows: $\infty(a; b) =_{\text{df}} \infty a \vee \langle a\rangle \infty b$, $\infty(a \cup b) =_{\text{df}} \infty a \vee \infty b$, $\infty(a^*) =_{\text{df}} \Delta a \vee \langle a^*\rangle \infty a$. This looping construct is useful in expressing total correctness of programs.

Remark. Many formulations of PDL include a *test* construct. If p is a formula, then $p?$ is a program which permits program execution to proceed if p is true and abnormally terminates program execution if p is false. The semantics of tests in a structure S are given by the following condition: $u \prec p? \succ_S v$ iff $u = v$ and $u \models_S p$. Informally, $p?$ is equivalent to *if p then skip else abort*. Tests can be used to define strucutred programming constructs: *if p then a else b* $=_{\text{df}} (p?; a) \cup (\neg p?; b)$ and *while p do a* $=_{\text{df}} (p?; a)^*; \neg p?$. Tests are not considered in this paper, but all results continue to hold in the presence of tests. For details, see Streett (1982).

DEFINITION 2.4. If p is a formula and S is a structure, then S is a *model* of p or S *satisfies* p if and only if $u \models_S p$ for some $u \in U$, and p is *satisfiable* if and only if some structure satisfies p. The formula p is *valid* in S if and only if $u \models_S p$ for all $u \in U$, and p is *valid* if and only if p is valid in all structures.

DEFINITION 2.5. A set X of formulae *expresses* a second set Y of formulae if and only if for every formula $p \in Y$ there is a formula $q \in Y$ such that $p \leftrightarrow q$ is valid. The set X is *more expressive* than the set Y if and only if X expresses Y but Y does not express X.

The following theorems rank PDL with looping and converse and some of

its sublogics with respect to expressive power. Theorem 2.7 establishes a property of formulae of PDL with converse which Theorem 2.8 shows is not shared by all formulae of PDL with looping. We conclude that PDL with looping is more expressive than PDL and that PDL with converse does not express PDL with looping. Theorems 2.9 and 2.10 establish that PDL with both constructs is more expressive than PDL with looping. Finally, Theorems 2.12 and 2.13 show that PDL with converse is more expressive than PDL and that PDL with looping does not express PDL with converse, so that PDL with converse and PDL with looping are incomparable in expressive power.

DEFINITION 2.6. Given a formula p and structures $S = \langle U, \models_S, <>_S \rangle$ and $T = \langle V, \models_T, <>_T \rangle$, a *p-homomorphism* from S to T is an onto map $f: U \to V$ such that for all states $u \in U$, $u \models_S p$ iff $f(u) \models_T p$ and for all atomic programs A appearing in p and states $u, v \in U$, $u <A>_S v$ iff $f(u) <A>_T f(v)$. If a p-homomorphism from S to T exists, we call T a *p-quotient* of S.

THEOREM 2.7 (Fischer and Ladner, 1979). PDL *with converse (and hence also* PDL*) satisfies the finite quotient property: every structure has a finite p-quotient for every formula p. A p-quotient can be constructed to have no more than* 2^n *states, where n is the length of p.*

THEOREM 2.8. *The looping construct destroys the finite quotient property; there is a structure with no finite* ΔA*-quotients.*

Proof. Consider an infinite structure S consisting of a single infinite reverse A-chain, i.e., $U = \{u_n\}_{n \geq 0}$ and $u_i <A>_S u_j$ iff $i = j + 1$. Then $u \models_S \neg\Delta A$ for every state u. Suppose f is a ΔA-homomorphism from S to a finite structure T, so that for all $u \in U$, $f(u) \models_T \neg\Delta A$. Since T is finite we must have $f(u) = f(v)$ for some $u \neq v$, implying $f(u) <A; A^*>_T f(u)$ and $f(u) \models_T \Delta A$, a contradiction. ∎

THEOREM 2.9 (Streett, 1980, 1982). PDL *with looping satisfies the finite model property; every satisfiable formula is satisfied in a finite model.*

THEOREM 2.10. PDL *with both looping and converse does not satisfy the finite model property; there is a satisfiable formula which is not satisfied in any finite model.*

Proof. Consider the satisfiable formula $\Delta A \wedge \neg\langle A^* \rangle \Delta(A^-)$. If $u_0 \models_S \Delta A \wedge \neg\langle A^* \rangle \Delta(A^-)$, then $u_0 \models_S \Delta A$ and $u_0 \models_S \neg\langle A^* \rangle \Delta(A^-)$. Hence there is an infinite A chain $u_0 <A>_S u_1 \cdots u_n <A>_S u_{n+1} \cdots$. If $u_i = u_j$ for any $i < j$, then $u_i \models_S \Delta(A^-)$ and so $u_0 \models_S \langle A^* \rangle \Delta(A^-)$, a contradiction. So all the

u_i are distinct. Hence, $\Delta A \wedge \neg\langle A^* \rangle \Delta(A^-)$ is satisfiable only in infinite models. ∎

DEFINITION 2.11. A structure S is a *one-to-one structure* if for all atomic programs A, the relation $\prec A \succ_S$ is one-to-one, i.e., for all states $u, v, w \in U$, if $u \prec A \succ_S v$ and $w \prec A \succ_S v$ then $u = w$.

THEOREM 2.12. PDL *with looping satisfies the one-to-one model property; every satisfiable formula is satisfied in a one-to-one model.*

Proof. Given any structure $S = \langle U, \models_S, \prec \succ_S \rangle$, define a structure $T = \langle U^+, \models_T, \prec \succ_T \rangle$, where U^+ is the set of nonempty sequences of states from U. For all atomic propositions P and states $v \in U^+$, let $v \models_T P$ iff *last* $(v) \models_S P$. For all atomic programs A and states $u, v \in V$, let $u \prec A \succ_T v$ iff $v = u\,last(v)$ and $last(u) \prec A \succ_S last(v)$. By construction T is one-to-one. It is not difficult to show that for all formulae p and states $v \in U^+$, $v \models_T p$ iff $last(v) \models_S p$. ∎

THEOREM 2.13. *The converse construct destroys the one-to-one model property; there is a satisfiable formula of* PDL *with converse which is not satisfied in any one-to-one model.*

Proof. Consider the satisfiable formula $P \wedge \langle A \rangle \langle A^- \rangle \neg P$. Suppose $u \models_S P \wedge \langle A \rangle \langle A^- \rangle \neg P$, where S is one-to-one. Then $u \models_S P$ and there is a state v such that $uA \succ_S v$ and $v \models_S \langle A^- \rangle \neg P$, so that there must be a state w such that $w \prec A \succ_S v$ and $w \models_S \neg P$. Since S is one-to-one, $u = w$. But this is impossible, since we have $u \models_S P$ and $w \models_S \neg P$. ∎

PDL with looping is also strong enough to express all formulae of the Propositional Algorithmic Logic of Mirkowska (1980) and the Temporal Logic of Branching Time of Ben-Ari, Manna, and Pnueli (1981). For details, see Streett (1982).

3. TWO-WAY AUTOMATA ON INFINITE TREES

Automata on infinite trees, called *one-way* automata in this section to distinguish them from the *two-way* variant defined below, have been extensively studied (Rabin, 1969; Hossley and Rackoff, 1972). We briefly review some fundamental definitions.

DEFINITION 3.1. The set $T_N = \{0, 1, ..., N-1\}^*$ of sequences of the first N nonnegative integers can be viewed as an infinite N-ary tree, in which the empty sequence Λ is the root and each sequence (or node) $x \in T_N$ has as its successors the sequences $x0, ..., x(N-1)$.

DEFINITION 3.2. A *finite (infinite) one-way path* through T_N is a finite (infinite) sequence $\pi = \{x_n\}$ of elements of T_N, such that for all n, x_{n+1} is a successor of x_n.

DEFINITION 3.3. If Σ is a finite alphabet, then an *infinite N-ary Σ-tree* is a function $f: T_N \to \Sigma$.

DEFINITION 3.4. A *(nondeterministic) one-way automaton A* on infinite N-ary Σ-trees is a tuple $\langle S, s_0, M, G\rangle$, where

(a) S is the set of states,

(b) $s_0 \in S$ is the initial state,

(c) $M: S \times \Sigma \to Powerset\ (S^N)$ is the next state function,

(d) $G \subseteq Powerset\ (S)$ is a set of accepting subsets.

DEFINITION 3.5. A *run* of A on an infinite N-ary Σ-tree f is a function $\rho: T_N \to S$ such that $\rho(\Lambda) = s_0$ and for all $x \in T_N$, $\langle \rho(x(0),..., \rho(x(n-1))\rangle \in M(\rho(x), f(x))$.

DEFINITION 3.6. If ρ is a run of A on f and π is an infinite one-way path, then $\mathrm{Inf}(\rho, \pi) = \{s \in S \mid \rho(x) = s$ for infinitely many x on $\pi\}$.

DEFINITION 3.7. An automaton A *accepts* an infinite N-ary Σ-tree f if and only if there is a run ρ of A on f such that for all infinite one-way paths π, $\mathrm{Inf}(\rho, \pi) \in G$.

THEOREM 3.8 (Rabin, 1969; Hossley and Rackoff, 1972). *The emptiness problem for an infinite tree automaton A, i.e., the problem of deciding whether or not A accepts any tree at all, is elementarily decidable.*

Analogously to two-way automata on finite strings, we can define two-way automata on infinite trees. Two-way automata compute along all infinite paths through a tree, i.e., i.e., computations begin at all the nodes of the tree and branch in all directions, including back towards the root. It is technically convenient to allow two-way automata to distinguish the root from all other nodes. Theorem 3.30 shows how to simulate *deterministic* two-way automata by nondeterministic one-way automata; it is an open problem whether this result can be extended to *nondeterministic two-way* automata. First, however, two-way infinite paths through infinite trees are defined, and some simple results proved about the structure of such paths.

DEFINITION 3.9. Two nodes x and y of T_N are *neighbors* when either x is a successor of y or y is a successor of x. For $0 \leqslant n \leqslant N-1$, the nth neighbor of x is xn; the Nth neighbor of x is its predecessor if it exists.

DEFINITION 3.10. A *finite* (*infinite*) *two-way path* on T_N is a finite (infinite) sequence $\{x_n\}$ of elements of T_N such that all n, x_n and x_{n+1} are neighbors. Let P_N denote the set of finite paths on the tree T_N. If $\pi = \{x_n\}_{1 \leqslant n \leqslant L}$ and $\tau = \{x_n\}_{L+1 \leqslant n \leqslant M}$ are two finite paths such that x_L and x_{L+1} are neighbors, then the *concatenation* of π and τ is π; $\tau = \{x_n\}_{1 \leqslant n \leqslant M}$ (defined similarly if τ is an infinite path). The relation $\pi < \tau$ holds if and only if $\tau = \pi$; σ for some nonempty path σ. A *loop* on x is a finite path $\{x_n\}_{1 \leqslant n \leqslant N}$ such that $x_1 = x_N = x$. A *simple loop* is a loop x; π; x such that π does not contain x. A *singleton* is a path consisting of a single element. An infinite path π is *cyclic* on x if and only if x occurs infinitely often in π; π is *acyclic* if and only if it is not cyclic on any x.

LEMMA 3.11. *If x; π; x is a simple loop, then π is a loop.*

Proof. Since x; π; x is a path from x to itself, π must begin and end with neighbors of x. Any path, however, which connects two distinct neighbors of x must include x. Hence, if π does not include x, π must begin and end with the same neighbor of x. ∎

LEMMA 3.12. *If $\pi = \{x_n\}_{n \geqslant 0}$ is an infinite acyclic path, then there is an infinite one-way path $\{y_n\}_{n \geqslant 0}$ such that $\pi = \sigma; \tau_0; \ldots; \tau_n; \ldots$, where each τ_n is a loop on y_n.*

Proof. The path π must contain a least element x. Let σ be a (possibly empty) initial segment of π preceding some occurrence of x in π. Let y_0 be x and let τ_0 be that segment of π which extends from σ to include the last occurrence of x in π, so that τ_0 is a loop on y_0. Inductively, given y_n and $\tau_n = \{x_m\}_{L \leqslant m \leqslant M}$, let $y_{n+1} = x_{M+1}$ and let τ_{n+1} be that segment of π which extends from $\sigma; \tau_0; \ldots; \tau_n$ to include the last occurrence of y_{n+1} in π, so that τ_{n+1} is a loop on y_{n+1}. The reader can verify that $\{y_n\}_{n \geqslant 0}$ is an infinite one-way path. ∎

DEFINITION 3.13. A *deterministic two-way automaton on infinite N-ary Σ-trees* is a tuple $A = \langle S, s_0, s_1, M, G \rangle$, where

(a) S is a finite set of states,

(b) $s_0 \in S$ is the initial state for the root,

(c) $s_1 \in S$ is the initial state for nonroot nodes,

(d) $M: S \times \Sigma \to S^{N+1}$ is the state map; for $s \in S$ and $\sigma \in \Sigma$, let $M(s, \sigma) = \langle M_0(s, \sigma), \ldots, M_N(s, \sigma) \rangle$. Informally, if A is in state s on a node labelled σ, then A will be in state $M_n(s, \sigma)$ on the nth neighbor of that node.

(e) $G \subseteq \mathit{Powerset}\ (S)$ is a collection of acceptable sets of states. Infor-

mally, A accepts a tree if for every infinite two-way path π, G contains the set of states entered infinitely often along π.

DEFINITION 3.14. The *run* of a two-way automaton A on an infinite N-ary Σ-tree f is the function $\rho: P_N \to S$ such that

(1) If π is the singleton path consisting of the root, $\rho(\pi) = s_0$.

(2) If π is a singleton path consisting of a nonroot node, $\rho(\pi) = s_1$.

(3) If π is a path ending in x and y is the nth neighbor of x, $\rho(\pi; y) = M_n(\rho(\pi), f(x))$.

DEFINITION 3.15. If ρ is the run of A on f and π is an infinite path, then $\text{Inf}(\rho, \pi) = \{s \in S \mid \rho(\tau) = s \text{ for infinitely many finite paths } \tau < \pi\}$.

DEFINITION 3.16. A two-way automaton A *accepts* an infinite N-ary Σ-tree f if and only if for all infinite two-way paths π, $\text{Inf}(\rho, \pi) \in G$, where ρ is the run of A on f.

Lemma 3.12 shows that an infinite path π can take only two forms: either π loops endlessly on a single node or else π passes through all the nodes of an infinite forward path, possibly looping on each one. This suggests that a one-way automata might be able to simulate a two-way automata by successively guessing state information about the loops on each node. This method of simulation is successful because it is possible for an automaton to check that the guesses include information about all possible loops.

DEFINITION 3.17. If S is a set of states, then a *circuit* is a triple $\langle s, X, t\rangle$, where $s, t \in S$ and $X \subseteq S$. $C_S = \textit{Powerset}\ (S \times \textit{Powerset}\ (S) \times S)$ is the collection of sets of circuits. Intuitively, a circuit represents the state history of a two-way automata as it passes through a loop: s and t are the initial and final states and X is the set of intermediate states. A circuit of the form $\langle s, \varnothing, s\rangle$ represents the instantaneous state of an automaton.

DEFINITION 3.18. Given an automaton A and a tree f, a *plan* for A on f is an infinite N-ary C_S-tree g such that for all $x \in T_N$:

(a) $\langle s_0, \varnothing, s_0\rangle \in g(\Lambda)$,

(b) $\langle s_1, \varnothing, s_1\rangle \in g(x)$, for $x \neq \Lambda$,

(c) if $\langle s, \varnothing, s\rangle \in g(x)$ and y is the nth neighbor of x, then $\langle M_n(s, f(x)), \varnothing, M_n(s, f(x))\rangle \in g(y)$,

(d) if $\langle s, X, t\rangle \in g(x)$ and $\langle t, Y, u\rangle \in g(x)$ with $X, Y \neq \varnothing$, then $\langle s, X \cup \{t\} \cup Y, u\rangle \in g(x)$, in which case the resulting circuit is called the *join* of the original two,

(e) if $\langle s, \varnothing, s\rangle \in g(x)$, y is the nth neighbor of x, x is the mth neighbor of y, and $\langle M_n(s, f(x)), Y, u\rangle \in g(y)$, then $\langle s, X \cup \{t, u\}, M_m(s, f(y))\rangle \in g(x)$, in which case the resulting circuit is called the *expansion* of the first one.

Conditions (a)–(e) are intended to force a plan to include circuits for all possible loops through a tree, but they do not rule out the presence of spurious circuits which do not correspond to any loop. The least or minimal plan, however, contains precisely the circuits for all loops.

LEMMA 3.19. *For each automaton A and tree f, there is a plan $g_{\min}$ for A on f such that for all plans g for A on f and nodes x, $g_{\min}(x) \subseteq g(x)$.*

Proof. Define $g_{\min}$ as the pointwise intersection of all plans for A on f. ∎

DEFINITION 3.20. Given a plan g for a tree f and an infinite forward path $\{x_n\}$, a *series* is an infinite sequence $\{\langle s_n, X_n, t_n\rangle\}$ of linked circuits on $\{x_n\}$, i.e., for all n, $\langle s_n, X_n, t_n\rangle \in g(x_n)$ and $s_{n+1} = M(t_n, f(x_n))$ if x_{n+1} is the mth neighbor of x_n.

DEFINITION 3.21. If ζ is a series, then $\mathrm{Inf}(\zeta) = \{s \mid$ for infinitely many circuits $\langle t, X, u\rangle$ in ζ, $s \in X \cup \{t, u\}\}$.

DEFINITION 3.22. A plan g for an automaton $A = \langle S, s_0, s_1, M, G\rangle$ is *good* if and only if

(a) for all x and $\langle s, X, s\rangle \in g(x)$, $X \cup \{s\} \in G$,

(b) for all series ζ on infinite forward paths, $\mathrm{Inf}(\zeta) \in G$.

The two goodness conditions correspond to the two forms, cyclic and acyclic, of infinite two-way paths. A circuit of the form $\langle s, X, s\rangle$ indicates that the automaton can cycle endlessly through the set $X \cup \{s\}$ of states while travelling over a cyclic path, while a series describes the state history of the automaton on an acyclic path.

LEMMA 3.23. *There is a good plan for an automaton A on a tree f if and only if the minimal plan is good.*

Proof. The goodness conditions ensure that any plan included in a good plan is also good, hence the minimal plan (which is included in every plan) must be good if any plan is good. ∎

The next series of lemmas shows that the minimal plan contains precisely the circuits for all loops. In what follows let $g_{\min}$ be the minimal plan for an automaton A on a tree f and let ρ be the run of A on f.

LEMMA 3.24. *For all x and $\langle s, X, t\rangle \in g_{\min}(x)$, there is a path π ending in x such that $\rho(\pi) = s$.*

Proof. If a circuit appears in the minimal plan, then there must be a derivation of this fact by the rules (a)–(e) in Definition 3.18. The result follows by induction on the structure of such derivations. ∎

LEMMA 3.25. *For all x and paths π ending in x, $\langle \rho(\pi), \varnothing, \rho(\pi)\rangle \in g_{\min}(x)$.*

Proof. By induction on the length of paths, using conditions (a)–(c) from Definition 3.18. ∎

LEMMA 3.26. *For all x and $\langle s, X, t\rangle \in g_{\min}(x)$, there is a loop π on x such that for all paths of the form $\tau; \pi$ with $\rho(\tau; x) = s$, $X = \{\rho(\mu) \mid \tau; x < \mu < \tau; \pi\}$ and $t = \rho(\tau; \pi)$.*

Proof. For circuits of the form $\langle s, \varnothing, x\rangle$, the required loop is the singleton x. If $\langle s, X, t\rangle$ is the join of two circuits $\langle t, Y, u\rangle$ and $\langle u, Z, v\rangle$, then inductively there must be loops $x; \sigma; x$ and $x; \tau; x$ for the component circuits (since $Y, Z \neq \varnothing$, these loops cannot be singletons). The required loop for the join is $x; \sigma; x; \tau; x$. If $\langle s, X, t\rangle$ is the expansion of a circuit $\langle t, Y, y\rangle$ on a neighbor y of x, then inductively there must be a loop σ on y for $\langle t, Y, u\rangle$. The required loop for the expansion is $x; \sigma; x$. ∎

LEMMA 3.27. *For all paths $\tau; \pi$ ending in a loop π on a node x, there is a circuit $\langle s, X, t\rangle \in g_{\min}(x)$ such that $s = \rho(\tau; x)$, $X = \{\rho(\mu) \mid \tau; x < \mu < \tau; \pi\}$, and $t = \rho(\tau; \pi)$.*

Proof. If π is the singleton x, the required circuit $\langle \rho(\tau; x), \varnothing, \rho(\tau; x)\rangle \in g_{\min}(x)$ by Lemma 3.25. If $\pi = x; \sigma; x$ with σ a loop on a neighbor y of x, then inductively there must be a circuit for σ on y. The required circuit for π on x is an expansion of the circuit for σ on y. If σ is not a loop, then by Lemma 3.11 it must contain x, i.e., $\sigma = \sigma_0; x; \sigma_1$. Inductively there must be circuits for $x; \sigma_0; x$ and $x; \sigma_1; x$. The required circuit for π is the join of these two circuits. ∎

LEMMA 3.28. *An automaton A accepts a tree f if and only if the minimal plan for A on f is good.*

Proof. First, suppose A rejects f. Then there is an infinite path π such that $\mathrm{Inf}(\rho, \pi) \notin G$. If π is cyclic on x, then $\pi = \tau_0; \sigma; \tau_1$, where σ is a loop on x so large that $\rho(\tau_0; x) = \rho(\tau_0; \sigma)$ and $\{\rho(\tau_0; x), \ldots, \rho(\tau_0; \sigma)\} = \mathrm{Inf}(\rho, \pi)$. By Lemma 27, $g(x)$ will violate condition (a) in Definition 3.22. If, on the other hand, π is acyclic, then by Lemma 3.12 there is an infinite forward path $\{x_n\}$ such that $\pi = \sigma; \tau_0; \tau_1; \ldots$, where each τ_n is a loop on x_n. It is straightforward

to select a series ζ of circuits for $g_{\min}$ on $\{x_n\}$ violating condition (b) in Definition 3.22.

Now suppose that the minimal plan is not good. Then either there is a node x with a bad circuit or an infinite forward path with a bad series. If there is a bad circuit $\langle s, X, s\rangle$ on x, use Lemmas 3.24 and 3.26 to construct a cyclic path π such that $\mathrm{Inf}(\rho, \pi) = X \cup \{s\} \notin G$. If there is a bad series ζ, use Lemmas 3.24 and 3.26 to construct an acyclic path π such that $\mathrm{Inf}(\rho, \pi) = \mathrm{Inf}(\zeta) \notin G$. ∎

DEFINITION 3.29. If f and g are infinite trees, define the *product tree* $f \times g$ by $(f \times g)(x) = \langle f(x), g(x)\rangle$.

THEOREM 3.30. *For every deterministic two-way automaton A there is a nondeterminitic one-way automaton B accepting exactly the same trees. Further, B can be constructed in time elementary in the size of A.*

Proof. It is straightforward to construct, in time elementary in the size of A, first, a one-way tree automaton C which accepts infinite $(S \times C_S)$-trees $f \times g$ exactly when g is a plan for f, and second, a nondeterministic automaton D on infinite sequences of circuits which accepts precisely the sequences violating the goodness conditions. McNaughton (1966) gives an elementary time construction for a deterministic automaton on infinite sequences which accepts the sequences rejected by a given automaton. Let E be the result of applying this construction to D, and let F be the deterministic tree automaton which simulates E down every forward path, so that F accepts exactly the good plans. The desired automaton B, given an input tree f, nondeterministically guesses a possible plan g and simulates F on g and C on $f \times g$. By Lemmas 3.23 and 3.28, A and B accept the same trees. ∎

Remark. The domain C_S has size doubly exponential in the size of A's state space, and McNaughton's construction involves a further doubly exponential blowup, so that B has size quadruply exponential in the size of A.

4. THE SATISFIABILITY PROBLEM

In this section the automata theoretic result of the previous section is used to obtain an elementary decision procedure for PDL with looping and converse. First, however, we precisely define the notions of a subformula of a formula and an execution sequence of a program.

DEFINITION 4.1. The *Fischer–Ladner closure* of a formula p is the least set of formulae $FL(p)$ such that

(a) $p \in FL(p)$,
(b) if $\neg q \in FL(p)$, then $q \in FL(p)$,
(c) if $q \vee r \in FL(p)$, then $q, r \in FL(p)$,
(d) if $\langle a \rangle q \in FL(p)$ then $q \in FL(p)$,
(e) if $\langle a; b \rangle q \in FL(p)$, then $\langle a \rangle \langle b \rangle q \in FL(p)$,
(f) if $\langle (a; b)^- \rangle q \in FL(p)$, then $\langle b^-; a^- \rangle q \in FL(p)$,
(g) if $\langle a \cup b \rangle q \in FL(p)$, then $\langle a \rangle q, \langle b \rangle q \in FL(p)$,
(h) if $\langle (a \cup b)^- \rangle q \in FL(p)$, then $\langle a^- \cup b^- \rangle q \in FL(p)$,
(i) if $\langle a^* \rangle q \in FL(p)$, then $q, \langle a \rangle \langle a^* \rangle q \in FL(p)$,
(j) if $\langle (a^*)^- \rangle q \in FL(p)$, then $\langle (a^-)^* \rangle q \in FL(p)$,
(k) if $\Delta a \in FL(p)$, then $\langle a \rangle \Delta a \in FL(p)$.

LEMMA 4.2. *If p is a formula of length n, then $FL(p)$ contains at most n formulae.*

Proof. A straightforward extension of the corresponding proof for PDL (Fischer and Ladner, 1979). ∎

DEFINITION 4.3. The elements of $FL(p)$ are called the *subformulae* of p, but note that $\langle a \rangle \langle a^* \rangle q$ and $\langle a \rangle \Delta a$ are, by the above definition, subformulae of $\langle a^* \rangle q$ and Δa, respectively. A subformula of p of the form $\langle a \rangle q$ is called a *diamond subformula* of p.

DEFINITION 4.4. A *literal program* is either an atomic program or the converse of an atomic program. The *inverse* of a literal program A is A^-; the *inverse* of A^- is A.

Programs with converse are extended regular expressions, and each program denotes a regular set of strings of literal programs, the set of its execution sequences. Note that the execution sequences of a^- can be obtained by inverting all literal programs in the reversals of execution sequences of a.

DEFINITION 4.5. The set, $L(a)$, of *execution sequences* of a program a, is defined inductively as follows:

(a) $L(A) = \{A\}$,
(b) $L(a; b) = L(a); L(b)$,
(c) $L(a \cup b) = L(a) \cup L(b)$,
(d) $L(a^*) = (L(a))^*$,
(e) $L(A^-) = \{A^-\}$,

(f) $L((a;b)^-) = L(b^-;a^-)$,

(g) $L((a \cup b)^-) = L(a^- \cup b^-)$,

(h) $L((a^*)^-) = L((a^-)^*)$,

(i) $L((a^-)^-) = L(a)$.

LEMMA 4.6. *For all structures* $S = \langle U, \models_S, <>_S \rangle$ *and programs* a, $u <a>_S v$ *if and only if there is an execution sequence* $b_1 \cdots b_k \in L(a)$ *and a sequence of states* $\{u_n\}_{0 \leqslant n \leqslant k}$ *such that* $u_0 = u$, $u_k = v$ *and* $u_n <b_{n+1}>_S u_{n+1}$ *for* $0 \leqslant n \leqslant k$.

Proof. By structural induction on programs. ∎

If p is a satisfiable formula, Theorem 4.9 shows that p has a special model, called a scheme, which is easily transformed into a tree suitable as input to a two-way automaton. A scheme is a tree-like structure in which p is satisfied at the root and diamond subformulae of p are satisfied along specified paths, enabling the truth value of these subformulae to be checked deterministically.

DEFINITION 4.7. A structure $S = \langle U, \models_S, <>_S \rangle$ is a *tree structure* if $U = T_N$ for some N and for all atomic programs A and states u, v, if $u <A>_S v$, then u and v are neighbors.

DEFINITION 4.8. If p is a formula with diamond subformulae $\langle a_1 \rangle q_1, \ldots, \langle a_N \rangle q_N$, then a *scheme* for p is a tree structure $S = \langle T_{N+1}, \models_S, <>_S \rangle$ such that $\Lambda \models_S p$ and for all states x, if $x \models_S \langle a_n \rangle q_n$ then $\exists y.$ $((y = x \vee \exists m \geqslant 0.\ y = xn0^m) \wedge x <a_n>_S y \wedge y \models_S q_n)$.

THEOREM 4.9. *Every satisfiable formula has a scheme.*

Proof. Suppose $u_0 \models_S p$, where $S = \langle U, \models_S, <>_S \rangle$. We construct a map $\varphi: T_{N+1} \to U$ inductively as follows: Let $\varphi(\Lambda) = u_0$. Inductively, if $\varphi(x) = u$, then we consider, for each n, whether $u \models_S \langle a_n \rangle q_n$. If not, let $\varphi(xn0^m)$ be arbitrary for all m. If so, then there is a state v such that $u <a_n>_S v \wedge v \models_S q_n$. By Lemma 4.6, there is a sequence of states $\{u_i\}_{0 \leqslant i \leqslant k}$ and an execution sequence $b_1 \cdots b_k \in L(a_n)$ such that $u_0 = u$, $u_n = v$, and $u_i <b_{i+1}>_S u_{i+1}$ for $0 \leqslant i < k$.

For $1 \leqslant i \leqslant k$, let $\varphi(xn0^{i-1}) = u_i$. For $i > k$, let $\varphi(xn0^{i-1})$ be chosen arbitrarily. Finally, given φ, define a structure $T = \langle T_{N+1}, \models_T, <>_T \rangle$ by letting $x \models_T P$ if and only if $\varphi(x) \models_S P$ and letting $x <A>_T y$ if and only if x and y are neighbors and $\varphi(x) <A>_S \varphi(y)$. By construction T is a scheme for p. ∎

Schemes are easily transformed into trees suitable for input to automata on infinite trees. The trees obtained in this way are recognizable by an automaton; this fact leads immediately to a decision procedure for PDL with looping and converse.

DEFINITION 4.10. If p is a formula, let $LP(p)$ denote the set of literal programs appearing in p and let $\Sigma_p = Powerset\ (FL(p) \cup LP(p))$.

DEFINITION 4.11. Given a scheme $S = \langle T_{N+1}, \models_S, < >_S \rangle$ for a formula p, the *image* of S is the $N+1$-ary Σ_p-tree f such that for all $x \in T_{N+1}$, $f(x) = \{q \in FL(p) \mid x \models_S q\} \cup \{a \in LP(p) \mid y <a>_S x$, where y is the predecessor of $x\}$. An *image for* p is an image of a scheme for p.

DEFINITION 4.12. Given a Σ_p-tree f, a program a *matches* a path $\pi = \{x_i\}_{0 \leqslant i \leqslant n}$ if and only if there is an execution sequence $b_1 \cdots b_n \in L(a)$ such that for $1 \leqslant i \leqslant n$,

(a) if x_i is a successor of x_{i-1}, then $b_i \in f(x_i)$,

(b) if x_i is the predecessor of x_{i-1}, then the inverse of $b_i \in f(x_i)$.

DEFINITION 4.13. Given a Σ_p-tree f, a program a *repeatedly matches* an infinite path $\{x_n\}_{n \geqslant 0}$ if and only if there is a infinite, increasing sequence of indices $\{i_j\}_{j \geqslant 0}$ such that $i_0 = 0$ and a matches $\{x_n\}_{i_{j-1} \leqslant n \leqslant i_j}$ for $j \geqslant 1$.

LEMMA 4.14. *A Σ_p-tree f is an image for p if and only if the following conditions are satisfied*:

(a) $p \in f(\Lambda)$,

(b) *for* $\neg q \in FL(p)$, $\neg q \in f(x)$ *if and only if* $q \notin f(x)$,

(c) *for* $q \vee r \in FL(p)$, $q \vee r \in f(x)$ *if and only if* $q \in f(x)$ *or* $r \in f(x)$,

(d) *if* $\langle a_n \rangle q_n \in f(x)$, *then there is an initial segment* π *of the infinite path* x; $\{xn0^m\}_{m \geqslant 0}$ *such that* a_n *matches* π *and* $q_n \in f(\text{last}(\pi))$,

(e) *if* $\langle a_n \rangle q_n \notin f(x)$, *then for all finite paths* π *starting at* x, *either* a_n *does not match* π *or* $q_n \notin f(\text{last}(\pi))$,

(f) *for* $\Delta a \in FL(p)$, $\Delta a \in f(x)$ *if and only if* $\langle a \rangle \Delta a \in f(x)$,

(g) *for* $\Delta a \in FL(p)$, *if* a *match the singleton path* x, *then* $\Delta a \in f(x)$,

(h) *fr* $\Delta a \in FL(p)$, *if* $\Delta a \notin f(x)$, *then for all infinite paths* π *starting at* x, a *does not repeatedly match* π.

Proof. We leave it to the reader to verify that an image for p satisfies (a)–(h). Conversely, given a $N+1$-ary Σ_p-tree f satisfying (a)–(h), we can define a two-way tree structure $S = \langle T_{N+1}, \models_S, < >_S \rangle$ by letting $x \models_S P$ iff

$P \in f(x)$ and $x\langle A\rangle_S y$ iff either y is a successor of x and $A \in f(x)$ or y is the predecessor of x and $A^- \in f(x)$. The reader can verify that f is the image of S. A structural induction on formulae establishes that for all $x \in T_{N+1}$ and $q \in FL(p)$, $x \models_S q$ iff $q \in f(x)$.

The base case, q an atomic subromula P, follows from the construction of S. Conditions (b) and (c) handle the Boolean constructs; conditions (d) and (e) the diamond subformulae. If q is a delta subformula Δa, then $(x \models_S \Delta a) \rightarrow (\Delta a \in f(x))$ follows for programs with the empty execution sequence by condition (g) and for other programs by condition (h), and $(\Delta a \in f(x)) \rightarrow (x \models_S \Delta a)$ follows from a reduction via condition (f) to the case of diamond subformulae. By condition (a), $\Lambda \models_S p$, and by condition (d), for $1 \leqslant n \leqslant N$, if $x \models_S \langle a_n\rangle q_n$, then $\exists y.\ ((y = x \vee \exists m \geqslant 0.\ y = xn0^m) \wedge x\langle a_n\rangle_S y \wedge y \models_S q_n)$. Hence S is a scheme for p. ∎

THEOREM 4.15. *Given a formula p, there is a deterministic two-way tree automaton A_p which accepts exactly the images for p. Further, A_p can be constructed in time elementary in the length of p.*

Proof. By Lemma 4.14, it is sufficient to construct an automaton accepting exactly the $N+1$-ary Σ_p-trees satisfying the conditions (a)–(h) of that lemma, where N is the number of diamond subformulae of p. It is straightforward to construct an automaton B with four states (two start states, an accepting state, and a failure state) which accepts exactly the trees satisfying conditions (a)–(c), (f), and (g). For $1 \leqslant n \leqslant N$, let A_n be a deterministic automaton on finite strings which accepts the regular set $L(a_n)$. Let C_n be an automaton on infinite trees which, for every node x in the tree labelled with $\langle a_n\rangle q_n$ runs the automaton A_n down the path $x; \{xn0^m\}_{m \geqslant 0}$, looking for an initial segment which is matched by the program a_n and ends in a node labelled q_n. Let D_n be an automaton on infinite trees which, for every node x in the tree not labelled with $\langle a_n\rangle q_n$, runs the automaton A_n down every path starting with x, rejecting the tree if a_n matches any finite path starting with x and ending with a node labelled q_n.

Given an automaton recognizing a regular set X not containing the empty string, there is a construction, due to McNaughton (1966), of a deterministic automaton on infinite strings which accepts exactly the infinite strings which cannot be parsed as infinite sequences of strings from X. For $\Delta a \in FL(p)$, let E_a be the result of applying McNaughton's construction to an automaton accepting $L(a) - \{\Lambda\}$. Let F_a be an automaton on infinite trees which, for every node x not labelled with Δa, runs the automaton E_a down every path from x in order to reject any tree containing a path from x which a repeatedly matches. Finally, the automaton B and the C_n's, D_n's, and F_a's can be combined in a crossproduct construction to yield the desired A_p. ∎

Remark. McNaughton's construction involves a double exponential blowup and is applied to deterministic automata constructed from regular expressions occurring in p. It follows that the number of states of A_p is at worst triply exponential in the length of the formula p.

THEOREM 4.16. *The satisfiability problem for* PDL *with looping and converse is elementarily decidable.*

Proof. Given a formula p, Theorem 4.15 constructs a two-way automaton A_p on infinite $N+1$-ary trees such that A_p accepts some tree if and only if p is satisfiable. By Theorem 3.30, there is an equivalent one-way automaton B on infinite $N+1$-ary trees. It is straightforward to construct a one-way automaton C on infinite binary trees, whose emptiness problem is equivalent to B's. The emptiness problem for one-way automata on infinite binary trees is elementarily decidable (Rabin, 1969; Hossley and Rackoff, 1972). ∎

Remark. The construction of the two-way automaton A_p runs in time at worst triple exponential in the length of p; the one-way simulation by B entails a quadruply exponential blowup. The emptiness problem for one-way automata is decidable in time exponential in the number of states of the automaton tested. It follows that the above decision procedure runs in time at worst octuply exponential in the length of the formula tested. The observant reader will note that two-way automata are used to handle the converse construct and that one-way automata suffice for converse-free PDL with looping. This observation leads to a triply exponential time decision procedure for that logic (Streett, 1980, 1982).

Rabin (1969) has shown that every nonempty automaton recognizable set of infinite trees contains a finitely generable tree, i.e., an infinite tree which can be obtained by unwinding a finite graph. Although PDL with looping and converse does not satisfy the finite model property, Rabin's result shows that every satisfiable formula has a finite representation. In the case of converse-free formulae, however, it is possible to transform the generating graph for an image for the formula into a finite model (Streett, 1980, 1982).

5. CONCLUSIONS AND OPEN PROBLEMS

Analysis of the elementary time decision procedure given in the preceding chapter shows that it runs in time at worst octuply exponential in the length of the formula tested. There is a large gap between this upper bound on the complexity of the satisfiability problem and the following lower bound established by Fischer and Ladner for PDL.

THEOREM 5.1 (Fischer and Ladner, 1979). *There is a constant $c > 1$ such that* PDL (*and hence its extensions*) *cannot be decided in time c^n, where n is the length of the formula tested.*

Pratt (1982) and Kozen (1982) have recently defined two distinct propositional versions of the mu-calculus (Park, 1969, 1973; de Bakker and de Roever, 1973; Hitchcock and Park, 1973). Both versions satisfy the finite model property and are decidable in exponential time. Pratt's logic can express PDL with converse, and Kozen's logic can express PDL with Δa for some but not all programs a. Since PDL with looping and converse does not satisfy the finite model property, it is not subsumed by either mu-calculus. It is an open problem whether the apparently incompatible features of the two versions can be reconciled, perhaps yielding an exponential time decision procedure for a still more powerful logic.

Since PDL is decidable, it has an uninteresting complete recursive axiomatisation: the set of all valid formulae. However, one would still like to find a simple and natural complete axiomatisation. In the case of PDL, a completeness proof for the following set of axioms was first announced by Segerberg (1977); the first complete proof is due to Parikh (1978).

Axioms:

(1) All the tautologies of the propositional calculus

(2) $[a](p \to q) \to ([a]p \to [a]q)$

(3) $[a;b]p \leftrightarrow [a][b]p$

(4) $[a \cup b]p \leftrightarrow [a]p \,\&\, [b]p$

(5) $[a^*]p \to p \,\&\, [a]p$

(6) $[a^*]p \to [a^*][a^*]p$

(7) $[a^*](p \to [a]p) \to (p \to [a^*]p)$

Rules of Inference:

(Modus ponens) If p and $p \to q$ are theorems, then q is a theorem.

(Generalization) If p is a theorem, then so is $[a]p$.

In addition, Parikh (1978) has shown that adding additional axioms

(8) $p \to [a]\langle a^- \rangle p$

(9) $p \to [a^-]\langle a \rangle p$

to the above axiomatisation for PDL yields a complete axiomatisation for PDL with converse. A natural question to ask is whether there are one or more axioms concerning the Δ construct which, when added to the above complete axiomatisations for PDL with or without converse, yield complete axiomatisations for logics with looping.

Conjecture. The following two axioms

(10) $\Delta a \to \langle a \rangle \Delta a$

(11) $[a^*](p \to \langle a \rangle p) \to (p \to \Delta a)$

are sufficient to produce complete axiomatisations for PDL with looping and PDL with both looping and converse.

This conjecture is strongly supported by Kozen's complete axiomatization of his propositional mu-calculus (Kozen, 1982).

REFERENCES

BEN-ARI, M., MANNA, Z., AND PNUELI, A. (1981), The temporal logic of branching time, *in* "Eighth ACM Symposium on the Principles of Programming Languages," pp. 164–176.

DE BAKKER, J. (1980), "Mathematical Theory of Program Correctness," Prentice-Hall, New York.

DE BAKKER, J., AND DE ROEVER, W. P. (1973), A calculus for recursive program schemes, *in* "First International Colloquium on Automata, Languages and Programming," pp. 167–196.

FISCHER, M. J., AND LADNER, R. E. (1979), Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18**, 194–211.

HAREL, D. (1979), "First Order Dynamic Logic," Springer-Verlag Lecture Notes in Computer Science No. 68, Berlin/New York.

HITCHCOCK, P., AND PARK, D., Induction rules and termination proofs, *in* "First International Colloquium on Automata, Languages and Programming," pp. 225–252.

HOSSLEY, R., AND RACKOFF, C. W. (1972), The emptiness problem for automata on infinite trees, *in* "Thirteenth IEEE Symposium on Switching and Automata Theory," pp. 121–124.

KOZEN, D. (1982), Results on the propositional mu-calculus, *in* "Ninth International Colloquium on Automata, Languages, and Programming," pp. 348–359.

MCNAUGHTON, R. (1966), Testing and generating infinite sequences by a finite automaton, *Inform. and Control* **9**, 521–530.

MEYER, A. R. (1974), Weak monadic second order theory of successor Is not elementary recursive, *in* "Boston Logic Colloquium," Springer-Verlag Lecture Notes in Mathematics, No. 453, Berlin/New York.

MIRKOWSKA, G. (1980), Complete axiomatizations of algorithmic properties of program schemes with bounded nondeterministic program schemes, *in* "Twelfth ACM Symposium on the Theory of Computing," pp. 14–21.

PARIKH, R. (1978a), A completeness result for propositonal dynamic logic, *in* "Symposium on the Mathematical Foundations of Computer Science," Springer-Verlag Lecture Notes in Computer Science, No. 24.

PARIKH, R. (1978b), A decidability result for a second order process logic, *in* "Nineteenth IEEE Symposium on Foundations of Computer Science," pp. 177–183.

PARK, D. (1969), Fixpoint induction and proofs of program properties, *in* "Machine Intelligence 5," Edinburgh Univ. Press, Edinburgh.

PARK, D. (1976), Finiteness is mu-ineffable, *Theoret. Comput. Sci.* **3**, 176–181.

PRATT, V. R. (1976), Semantical considerations on Floyd–Hoare logic *in* "Seventeenth IEEE Symposium on Foundations of Computer Science," pp. 109–121.

PRATT, V. R. (1978), "Applications of Modal Logic to Programming," MIT LCS Technical Memo TM-116, MIT Press, Cambridge, Mass.

PRATT, V. R. (1979), Models of program logics, *in* "Twentieth IEEE Symposium on the Foundations of Computer Science," pp. 115–122.

PRATT, V. R. (1982), A decidable mu-calculus: Preliminary report, *in* "Twenty-second ACM Symposium on Foundations of Computer Science," pp. 421–427.

RABIN, M. O. (1969), Decidability of second order theories and automata on infinite trees, *Trans. Amer. Math. Soc.* **141**, 1–35.

SEGERBERG, K. (1977), A completeness theorem in the modal logic of programs (preliminary report), *Notices Amer. Math. Soc.* **24**, A–522.

STREETT, R. S. (1979), "Propositional Dynamic Logic and Program Divergence," M. S. thesis proposal, Dept. of EECS, Massachusetts Institute of Technology.

STREETT, R. S. (1980), "A Propositional Dynamic Logic for Reasoning About Program Divergence," M. S. thesis, Dept. of EECS, Massachusetts Institute of Technology.

STREETT, R. S. (1981), Propositional dynamic logic of looping and converse, *in* "Thirteenth ACM Symposium on the Theory of Computing," pp. 375–381.

STREET, R. S. (1982), "Propositional Dynamic Logic of Looping and Converse," Ph. D. thesis, LCS Technical Report TR-263, Massachusetts Institute of Technology.

Printed in Belgium