LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-263

PROPOSITIONAL DYNAMIC LOGIC OF LOOPING AND CONVERSE

Robert S. Streett

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

This blank page was inserted to preserve pagination.

ACKNOWLEDGEMENTS

During my four years at MIT, my thesis supervisors, Albert Meyer and Vaughan Pratt, were always generous with their time, giving me much expert guidance and warm encouragement. I am deeply grateful. Many thanks to Michael Sipser and Neil Immerman for so readily and amiably agreeing to be readers. The students of the Theory Group were, without exception, always helpful and supportive. Extra thanks are due to Andy Moulton for so cheerfully enduring my constant conversational forays into his office.

Loving care, essential for the continued mental health of thesis-fixated graduate students, was provided by someone very special: Denise.

This research was funded in part by the National Science Foundation under grant MCS 7910261.

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

[19] M. O. Rabin, "Automata on Infinite Trees and the Synthesis Problem", Hebrew University Dept. of Mathematics Technical Report No. 37, 1970.

[20] K. Segerberg, "A Completeness Theorem in the Modal logic of Programs" (preliminary report), *Notices of the American Mathematical Society, 24,* 6, A-522, 1977.

[21] R. S. Streett, "Propositional Dynamic Logic and Program Divergence", M. S. thesis proposal, Dept. of EECS, MIT, 1979.

[22] R. S. Streett, *A Propositional Dynamic Logic for Reasoning About Program Divergence,* M. S. thesis, Dept. of EECS, MIT, 1980.

[23] R. S. Streett, "Propositional Dynamic Logic of Looping and Converse", *Proceedings of the 13th ACM Symposium on the Theory of Computing,* 375-381, 1981.

1 Introduction

Dynamic logic [5, 6, 15, 16] 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* [5, 6, 16] is the propositional calculus; its programming language consists of regular expressions over uninterpreted program labels and tests, i.e., the programming primitives are black box programs, and more complicated programs are built up using the nondeterministic control structures of sequencing, testing, 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 [16]. However, by adding a natural formula construct *delta* 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: Mirkowska's *Propositional Algorithmic Logic* (*PAL*) [12] and Ben-Ari's, Manna's, and Pnueli's *Unified Logic of Branching Time* (*UB*) [1].

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* [5, 17]. This technique does not extend to *delta-PDL*, since there is a formula which is satisfiable in an infinite model which cannot be reduced to a finite model by merging states. This *delta-PDL* formula is therefore not equivalent to any *PDL* formula, and so *delta-PDL* is strictly more expressive than *PDL*. Nevertheless, we shall see that *delta-PDL* is decidable and does satisfy a finite model property.

Pratt's original formulation of dynamic logic included the programming construct *converse* [15]. Given a program a, the *converse* of a is the program which runs a backwards, i.e., which undoes all the computations performed by a. *Converse-PDL*, the extension of *PDL* to include the *converse* construct, satisfies the same finite model property as *PDL* and the known decision procedures for *PDL* extend without difficulty to *converse-PDL* [5, 17].

Bibliography

[1] M. Ben-Ari, Z. Manna, and A. Pnueli, "The Temporal Logic of Branching Time", *Proceedings of the 8th ACM Symposium on the Principles of Programming Languages*, 164-176, 1981.

[2] J. de Bakker, *Mathematical Theory of Program Correctness*, Prentice-Hall, 1980.

[3] E. W. Dijkstra, "Guarded Commands, Nondeterminacy, and Formal Derivation of Programs", *Communications of the ACM, 18*, 8, 1975.

[4] E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[5] M. J. Fischer and R. E. Ladner, "Propositional Dynamic Logic of Regular Programs", *Journal of Computer Science and Systems, 18*, 194-211, 1979.

[6] D. Harel, *First Order Dynamic Logic, Springer-Verlag Lecture Notes in Computer Science 68*, 1979.

[7] C. A. R. Hoare, "An Axiomatic Basis for Computer Programming", *Communications of the ACM, 8*, 576-580, 1969.

[8] R. Hossley and C. W. Rackoff, "The Emptiness Problem for Automata on Infinite Trees", *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, 121-124, 1972.

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

[10] A. Meyer, "Weak Monadic Second Order Theory of Successor Is Not Elementary Recursive", *Boston Logic Colloquium, Springer-Verlag Lecture Notes in Mathematics 453*, 1974.

[11] A. Meyer and K. Winklmann, "On the Expressive Power of Dynamic Logic", *Proceedings of the 11th ACM Symposium on the Theory of Computing*, 167-175, 1979.

[12] G. Mirkowska, "Complete Axiomatizations of Algorithmic Properties of Program Schemes with Bounded Nondeterministic Program Schemes", *Proceedings of the 12th ACM Symposium on the Theory of Computing*, 14-21, 1980.

[13] R. Parikh, "A Completeness Result for Propositional Dynamic Logic", *Proceedings of the Symposium on the Mathematical Foundations of Computer Science, Springer-Verlag Lecture Notes in Computer Science 24*, 1978.

[14] R. Parikh, "A Decidability Result for a Second Order Process Logic", *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, 177-183, 1978.

[15] V. R. Pratt, "Semantical Considerations on Floyd-Hoare Logic", *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, 109-121, 1976.

[16] V. R. Pratt, *Applications of Modal Logic to Programming*, MIT LCS Technical Memo TM-116, 1978.

[17] V. R. Pratt, "Models of Program Logics", *Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science*, 115-122, 1979.

The two constructs *delta* and *converse* interact to make *delta-converse-PDL* significantly different from either *delta-PDL* or *converse-PDL*. *Delta-converse-PDL* 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 *delta-converse-PDL* is strictly more expressive than either *delta-PDL* or *converse-PDL*. The failure of a logic to satisfy the finite model property is often taken as an indication of its undecidability, but in this case the evidence is misleading; *delta-converse-PDL* is in fact elementarily decidable, viz., decidable in time bounded by an eightfold composition of exponential functions.

There is a straightforward proof of the decidability of *delta-PDL* by embedding it into *SnS*, the second order theory of several successors [21]. (This method was used by Parikh to prove the decidability of a logic which he called *Second Order Acyclic Process Logic* (*SOAPL*) [14].) The upper bound on the complexity of *delta-PDL* obtained in this way is not elementary, since *SnS* cannot be decided in elementary time [10]. In any case, there does not appear to be a straightforward embedding of *delta-converse-PDL* into *SnS*.

Models of *delta-PDL* and *SOAPL* formulae 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 [18]. A quadruply exponential time decision procedure for *delta-PDL* can be obtained by directly reducing *delta-PDL* satisfiability to this emptiness problem, bypassing the translation into *SnS* [22]. The reduction involves the construction, for each formula p, of an automaton which accepts, in some 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 *delta-PDL* does satisfy the finite model property. The quadruply exponential upper bound on the computational complexity of *delta-PDL* can be improved by an exponential factor by showing that the automata used to decide *delta-PDL* satisfiability belong to a special class whose emptiness problem is exponentially easier than the general case.

Models of *delta-converse-PDL* formulae are also labelled graphs and these graphs can also be unwound into tree-structured models. However, unlike the tree models for the previous logics, programs in *delta-converse-PDL* tree models do not conform to the underlying tree structure; programs can link arbitrary nodes of the tree. The presence of such programs prevents a straightforward reduction of *delta-converse-PDL* 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 *delta-converse-PDL* is reducible to the emptiness problem for these newly defined automata. The decidability of *delta-converse-PDL* follows from a reduction of the two-way emptiness problem to the ordinary or one-

In addition, Parikh showed that adding additional axioms

(9) $p \rightarrow [a]\langle a^{-}\rangle p$

(10) $p \rightarrow [a^{-}]\langle a\rangle p$

to the above complete axiomatisation for *PDL* yields a complete axiomatisation for *converse-PDL* [13]. A natural question to ask is whether there is are one or more axioms concerning the Δ construct which, when added to the above complete axiomatisations for *PDL* and *converse-PDL*, yield complete axiomatisations for *delta-PDL* and *delta-converse-PDL*.

Conjecture: The following two axioms

(11) $\Delta a \leftrightarrow \langle a\rangle \Delta a$

(12) $[a^*](p \rightarrow \langle a\rangle p) \rightarrow (p \rightarrow \Delta a)$

are sufficient to produce complete axiomatisations for *delta-PDL* and *delta-converse-PDL*.

The complexity theory results in this thesis have depended very heavily on results concerning finite automata on infinite trees. Below are two interesting open problems concerning two-way automata.

Open Problem: Can nondeterministic two-way automata be simulated by one-way automata?

Open Problem: How many states are required to simulate a two-way automaton with a one-way automaton? In particular, is there, for infinitely many n, a two-way automaton with n states which cannot be simulated by a one-way automaton with less than 2^n (or 2^{2^n} or $2^{2^{2^n}}$ or $2^{2^{2^{2^n}}}$) states?

way emptiness problem.

Although *delta-converse-PDL* does not satisfy the finite model property, the models of a *delta-converse-PDL* formula 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 *representation* of a model, so that *delta-converse-PDL* satisfies a finite representation property. This clarifies why the logic is decidable.

Most of the results in this thesis which concern *delta-PDL* originally appeared, in different form, in the author's Master's thesis [22]. A preliminary version of the results in this thesis concerning *delta-converse-PDL* appeared in the *Proceedings of the Thirteenth ACM Symposium on the Theory of Computing* [23].

6 Conclusions and Open Problems

The main results of this thesis are elementary recursive decision procedures (i.e., algorithms which run in time $O(\exp^m n)$ for some m, where n is the length of the input) for *delta-PDL* and *delta-converse-PDL*. The existence of these algorithms establishes upper bounds on the computational time complexity of the satisfiability problem for these logics. Unfortunately, the best lower bound for these logics is the following one proved by Fischer and Ladner for *PDL*.

Theorem 6.1 [6]: 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.

The large gaps between the best known upper and lower bounds, doubly exponential in the case of *delta-PDL* and septuply exponential in the case of *delta-converse-PDL*, leave room for further work in the complexity theory of these logics.

Open Problem: What are the exact computational complexities of *delta-PDL* and *delta-converse-PDL*? In particular, does either or both require doubly exponential time to decide?

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 [20]; the first complete proof to appear is due to Parikh [13].

Axioms:

(1) All the tautologies of the propositional calculus
(2) $[a](p \rightarrow q) \rightarrow ([a]p \rightarrow [a]q)$
(3) $[a;b]p \leftrightarrow [a][b]p$
(4) $[a \cup b]p \leftrightarrow [a]p \ \& \ [b]p$
(5) $[a^*]p \rightarrow p \ \& \ [a]p$
(6) $[a^*]p \rightarrow [a^*][a^*]p$
(7) $[a^*](p \rightarrow [a]p) \rightarrow (p \rightarrow [a^*]p)$
(8) $[p?]q \leftrightarrow (p \rightarrow q)$

Rules of Inference:

(Modus ponens) If p and $p \rightarrow q$ are theorems, then q is a theorem.
(Generalization) If p is a theorem, then so is $[a]p$.

2 Syntax, Semantics, and Expressive Power

In this chapter we formally define the syntax and semantics of *delta-converse-PDL* (which contains *PDL*, *delta-PDL*, and *converse-PDL* as sublogics). We then show how a large number of logical constructs used in proving program correctness can be expressed in *delta-converse-PDL*. We next prove some relationships between *delta-converse-PDL*, its various sublogics, and two other propositional logics of programs, the *Propositional Algorithmic Logic* (*PAL*) of Mirkowska [12] and the *Unified Temporal Logic of Branching Time* (*UB*) of Ben-Ari, Manna, and Pnueli [1].

We are given a set Π_0 whose elements are called *atomic programs* and a set Φ_0 whose elements are called *atomic formulae*. Capital letters *A, B, C, . . .* from the beginning of the alphabet will be used as variables over Π_0, and capital letters *P, Q, R, . . .* from the middle of the alphabet will be used as variables over Φ_0.

The set of programs, Π, and the set of formulae, Φ, of *delta-converse-PDL* are then defined inductively (note the use of letters *a, b, c, . . .* as variables over Π and *p, q, r, . . .* as variables over Φ):

Π: (1) $\Pi_0 \subseteq \Pi$

(2) If $a, b \in \Pi$ then $a;b,\ a \cup b,\ a^*,\ a^- \in \Pi$

(3) If $p \in \Phi$ then $p? \in \Pi$

Φ: (1) $\Phi_0 \subseteq \Phi$

(2) If $p \in \Phi$ then $\neg p \in \Phi$

(3) If $a \in \Pi$ and $p \in \Phi$ then $\langle a \rangle p,\ \Delta a \in \Phi$

The sublogics of *delta-converse-PDL* are defined as follows. The formulae and programs of *converse-PDL* are those not containing any occurrence of Δa. The formulae and programs of *delta-PDL* are those not containing any occurrence of a^-. The formulae and programs of *PDL* are those containing neither Δa nor a^-.

Definition: A structure is a triple $S = \langle U, \models_S, \langle\rangle_S \rangle$ where

(1) U is a non-empty set, the universe of states.

(2) $\models_S$ is a satisfiability relation on the atomic propositions, i.e. a predicate on $U \times \Pi_0$.

(3) $\langle\rangle_S$ assigns binary relations on states to the atomic programs.

Definition: A structure $S = \langle U, \models_S, \langle\rangle_S \rangle$ is a *tree structure* if and only if U is a tree and for all states u and v and atomic programs A, $u \langle A \rangle_S v$ only if u and v are neighbors in the tree, i.e., either v is a successor of u or vice versa. The tree structure S is *one-way* if and only if for all states u

Note: The above proof does not extend to *delta-converse-PDL*, since in general programs with converse can repeatedly fit infinite paths in the generating graph of a finitely-generable image without fitting the corresponding paths in the complete image.

and v and atomic programs A, $u\langle A\rangle_S v$ only if v is a successor of u.

Definition: Given a structure S, $\models_S$ and $\langle\rangle_S$ can be extended to arbitrary formulae and programs as follows:

(1) $u \models_S \neg p$ iff not $u \models_S p$.

(2) $u \models_S \langle a\rangle p$ iff $\exists$v. $u\langle a\rangle_S v$ & $v \models_S p$.

(3) $u \models_S \Delta a$ iff $\exists u_0, u_1, \ldots$ such that $u_0 = u$ and $\forall n \geq 0$. $u_n\langle a\rangle_S u_{n+1}$.

(4) $u\langle a;b\rangle_S v$ iff $\exists w$. $u\langle a\rangle_S w$ and $w\langle b\rangle_S v$.

(5) $u\langle a \cup b\rangle_S v$ iff $u\langle a\rangle_S v$ or $u\langle b\rangle_S v$.

(6) $u\langle a^*\rangle_S v$ iff $u\langle a\rangle_S{}^* v$.

(7) $u\langle a^-\rangle_S v$ iff $v\langle a\rangle_S u$.

(8) $u\langle p?\rangle_S v$ iff $u = v$ and $u \models_S p$.

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 p is a formula, then the program $p?$ can be thought of as an abbreviation for *if p then skip else abort,* i.e., it permits execution to proceed if p is true and interrupts execution if p is false. 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 ouput state, doing a followed by a^- can take a state to some other state besides itself). If a is a program, then Δa is a formula which is true whenever there is a way to repeatedly execute the program a without stopping.

The primitive constructs of *delta-converse-PDL* can be used to define many other interesting constructs as abbreviations. For example:

A correctness assertion: $[a]p =_{df} \neg\langle a\rangle\neg p$

Boolean operators:
$p \,\&\, q =_{df} \langle p?\rangle q$
$p \rightarrow q =_{df} [p?]q$
$p \vee q =_{df} \neg p \rightarrow q$
$p \leftrightarrow q =_{df} (p \rightarrow q) \,\&\, (q \rightarrow p)$

Propositional constants:
$true =_{df} P \vee \neg P$
$false =_{df} P \,\&\, \neg P$

Rabin [8, 19] 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 *delta-converse-PDL* does not satisfy the finite model property, Rabin's result shows that every satisfiable *delta-converse-PDL* formula has a finite representation. In the case of *delta-PDL* formulae, however, it is possible to transform the generating graph for an image for the formula into a finite model.

Theorem 5.11: For all *delta-PDL* formulae p, if p is satisfiable, then p has a finite model.

Proof: If p is satisfiable, then by the preceding results, there is a scheme $S = \langle T_{N+1}, \models_S, \langle\rangle_S \rangle$ for p whose image f is finitely generable. Hence there is a finite subtree T of T_{N+1} and a generating map J: $front(T) \rightarrow int(T)$ such that $f = f \circ J^*$. Define a finite structure $R = \langle T, \models_R, \langle\rangle_R \rangle$ as follows. For $x \in T$ and P an atomic program, let $x \models_R P$ iff $x \models_S P$. For x and $y \in T$ and A an atomic program, let $x\langle A\rangle_R y$ iff either $x \in int(T)$ and $x\langle A\rangle_S y$ or $x \in front(T)$ and $J(x)\langle A\rangle_S y$. We will prove, by structural induction on formulae, that for all $y \in T_{N+1}$ and q a subformula of p, $y \models_S q$ if and only if $J^*(y) \models_R q$.

If q is an atomic subformula P, then $y \models_S P$ iff $J^*(y) \models_S P$, since the image of S is generated by T and J. By the definition of R, $J^*(y) \models_S P$ iff $J^*(y) \models_R P$. If q is a negated subformula, then $y \models_S q$ iff $J^*(y) \models_R q$ follows from the inductive hypothesis and the definition of negation.

If q is a diamond subformula $\langle a\rangle r$, then suppose $y \models_S \langle a\rangle r$. Then by *Lemma 5.2* there must be an execution sequence $b_1 \cdot \cdot \cdot b_k \in L(a;r?)$ and a sequence $\{y_n\}_{0 \leq n \leq k}$ of elements of T_{N+1} such that $y_0 = y$ and for $0 \leq n < k$, $y_n\langle b_{n+1}\rangle_S y_{n+1}$. We leave it to the reader to verify that for $0 \leq n < k$, $J^*(y_n)\langle b_{n+1}\rangle_R J^*(y_n)$, so that $J^*(y = y_0) \models_R \langle a\rangle r$.

Conversely, suppose $x = J^*(y)$ and $x \models_R \langle a\rangle r$. Then there must be an execution sequence $b_1 \cdot \cdot \cdot b_k \in L(a;r?)$ and a sequence $\{x_n\}_{0 \leq n \leq k}$ of elements of T such that $x_0 = x$ and for $0 \leq n < k$, $x_n\langle b_{n+1}\rangle_S x_{n+1}$. Inductively define a sequence $\{y_n\}_{0 \leq n \leq k}$ of elements of T_{N+1} as follows. Let $y_0 = y$ and having defined y_n, define y_{n+1} in accord with the relationship between x_n and x_{n+1}. If b_{n+1} is a test, then $x_{n+1} = x$, so let $y_{n+1} = y_n$. Otherwise, b_{n+1} is an atomic program (since p is *converse*-free), and x_{n+1} is a successor, the *mth* say, of x_n if $x_n \in int(T)$, or of $J(x_n)$ if $x_n \in front(T)$. In this case let y_{n+1} be the *mth* successor of y. It is now straightforward to prove that $J^*(y_n) = x_n$ for $0 \leq n \leq k$, and that $y_n\langle b_{n+1}\rangle_S y_{n+1}$ for $0 \leq n < k$. Hence, $y \models_S \langle a\rangle r$.

If q is a delta subformula Δa, then $y \models_S \Delta a$ if and only if $J^*(y) \models_R \Delta a$ follows by an argument almost identical to the previous one for diamond subformulae. We conclude that $\Lambda \models_R p$, since $\Lambda \models_S p$ and p is a subformula of p. Therefore the structure R is a finite model of p. ∎

Program constants: $skip =_{df} true?$
$abort =_{df} false?$

Deterministic control structures:

$$if\ p\ then\ a\ else\ b =_{df} (p?;a) \cup (\neg p?;b)$$
$$while\ p\ do\ a =_{df} (p?;a)^*;\neg p?$$

Dijkstra's guarded commands [3]:

$$IF\ p \rightarrow a \,[]\, q \rightarrow b\ FI =_{df} (p?;a) \cup (q?;b)$$
$$DO\ p \rightarrow a \,[]\, q \rightarrow b\ OD =_{df} ((p?;a) \cup (q?;b))^*;(\neg p \,\&\, \neg q)?$$

de Bakker's weakest preconditions [2]:

$$a \rightarrow p =_{df} [a]p$$

de Bakker's strongest postconditions [2]:

$$a \leftarrow p =_{df} \langle a^{-} \rangle p$$

Hoare's partial correctness assertions [7]:

$$p\{a\}q =_{df} p \rightarrow [a]q$$

A well-foundedness or convergence assertion:

$$\nabla a =_{df} \neg \Delta a$$

An infinite looping assertion [6, 11, 16], defined inductively:

$$\infty A =_{df} false$$
$$\infty(a;b) =_{df} \infty a \vee \langle a \rangle \infty b$$
$$\infty(a \cup b) =_{df} \infty a \vee \infty b$$
$$\infty(a^*) =_{df} \langle a^* \rangle \infty a \vee \Delta a$$
$$\infty(p?) =_{df} false$$

(Alternatively, one can amend the syntax by adding the ∞A's to Π_0, allowing structures to decide arbitrarily which primitive programs loop and which do not.)

Dijkstra's weakest precondition operator [4]:

$$wp(a, p) =_{df} [a]p \,\&\, \langle a \rangle true \,\&\, \neg \infty a$$

Definition: If $p \in \Phi$ 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 *satisfiability problem* for *delta-converse-PDL* is the problem of deciding whether or not an arbitrary *delta-converse-PDL* formula is satisfiable.

converse-PDL satisfiability can be decided in time $O(\exp^8 k)$, where k is the length of the formula tested. ∎

Theorem 5.9: Given a *delta-PDL* formula p of length k, there is a deterministic complemented pairs automaton A_p, with no more than $O(\exp \exp k)$ states and $O(\exp k)$ pairs, which accepts exactly the images of one-way schemes for p. Furthermore, A_p can be constructed in time $O(\exp \exp k)$.

Proof: The proof is very similar to that of *Theorem 5.7*. By *Corollary 5.6*, it is sufficient to construct an automaton accepting exactly the $N+1$-ary Σ_p-trees satisfying the conditions (1)-(7) and an extra condition: (8) $f(x)$ contains no negative literals, for all x. It is straightforward to construct a complemented pairs automaton B with three states (a start state, an accepting state, and a failure state) and one pair which accepts exactly the trees satisfying conditions (1), (2), (5), (6), and (8). On the assumption that condition (8) is fulfilled, only forward paths need be considered to check conditions (4) and (7). It is not difficult to construct complemented pairs automata C_n and D_n which check conditions (3) and (4) respectively and which have exactly one pair and no more than $O(\exp k)$ states.

Given a deterministic m state automaton recognizing a regular set X (not containing the empty string) over an alphabet Σ, a construction of McNaughton's [9] yields a deterministic pairs automaton on infinite strings, with $O(\exp m)$ states and $O(m)$ pairs which accepts exactly the infinite strings in $\Sigma^*;X^\infty$. Since McNaughton's machine is a deterministic pairs automaton on infinite strings, it can be viewed as a complemented pairs automaton accepting exactly the infinite strings *not* in $\Sigma^*;X^\infty$.

For $\Delta a \in cl(p)$, let E_a be the complemented pairs automaton resulting from applying the above construction to a deterministic automaton accepting $\{\eta_0 b_1 \cdots b_k \eta_k \in C(a) \mid k \geq 1$ and $\Delta a \notin \eta_0\}$. Let F_a be an automaton on infinite trees which, runs the automaton E_a down every path from the root in order to reject any tree containing a node x such that $\Delta a \notin f(x)$ and an infinite path from x which a repeatedly fits. Each F_a can be constructed to have no more than $O(\exp \exp k)$ states and $O(\exp k)$ pairs.

Finally, the automaton B and the C_n's, D_n's, and F_a's can be combined in a cross-product construction to yield the desired A_p. A_p has no more than $O(\exp \exp k)$ states and $O(\exp k)$ pairs and can be constructed in time $O(\exp \exp k)$. ∎

Theorem 5.10: The satisfiability problem for *delta-PDL* is decidable in time $O(\exp^3 k)$, where k is the length of the formula tested.

Proof: Given a formula p of length k, *Theorem 5.7* constructs a complemented pairs automaton A_p on infinite $N+1$-ary trees with no more than $O(\exp \exp k)$ states and $O(\exp k)$ pairs such that A_p accepts some tree if and only if p is satisfiable. By *Theorem 3.8*, the emptiness problem for A_p can be decided in time $O(\exp^3 k)$. ∎

Definition: If $p \in \Phi$ and S is a structure, then 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: 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 *delta-converse-PDL* and some of its sublogics with respect to expressive power. *Theorem 2.1*, due to Fischer and Ladner, establishes a property of *PDL* and *converse-PDL* formulae which *Theorems 2.2* and *2.3* show is not shared by all *delta-PDL* and *delta-converse-PDL* formulae. We conclude that *delta-PDL* is more expressive than *PDL*, that *delta-converse-PDL* is more expressive than either *delta-PDL* or *converse-PDL*, and that *converse-PDL* does not express *delta-PDL*. Finally, *Theorem 2.4* shows that *converse-PDL* is more expressive than *PDL* and that *delta-PDL* does not express *converse-PDL*, so that *converse-PDL* and *delta-PDL* are incomparable in expressive power.

Theorem 2.1 [5]: *Converse-PDL* (and hence also *PDL*) satisfies the collapsing finite model property: every model of a formula cannot be collapsed to a finite model by identifying states. The resulting finite model has at most 2^n states, where n is the length of the formula.

Theorem 2.2: *Delta-PDL* does not satisfy the collapsing finite model property; there is a formula with an infinite model which cannot be collapsed to a finite structure without altering the truth value of the formula at some state.

Proof: Consider an infinite structure S with an infinite reverse A-chain (i.e., a sequence $\{u_n\}_{n\geq 0}$ of states such that $u_{n+1}\langle A\rangle_S u_n$ for all n), but no infinite forward A-chains (i.e., sequences $\{u_n\}_{n\geq 0}$ of states such that $u_n\langle A\rangle_S u_{n+1}$ for all n). Then for every state u along the reverse A-chain, $u \models_S \neg\Delta A$. However, S cannot be collapsed to a finite structure T without identifying two distinct states, u and v say, on the chain. If w is the collapse of u and v in T, then $w\langle A;A^*\rangle_T w$, and hence $w \models_T \Delta A$. ∎

Theorem 2.3: *Delta-converse-PDL* 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 $\Delta A \;\&\; \neg\langle A^*\rangle\Delta(A^-)$. If $u_0 \models_S \Delta A \;\&\; \neg\langle A^*\rangle\Delta(A^-)$, 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 $u_0\langle A\rangle_S u_1 \cdots u_n\langle A\rangle_S u_{n+1} \cdots$. 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, $\Delta A \;\&\; \neg\langle A^*\rangle\Delta(A^-)$ is satisfiable only in infinite models. ∎

Proof: A straightforward extension of the preceding proof. ∎

Theorem 5.7: Given a *delta-converse-PDL* formula p of length k, there is a deterministic two-way tree automaton A_p which accepts exactly the images for p. Further, A_p need have no more than $O(\exp \exp k)$ states and can be constructed in time $O(\exp \exp k)$.

Proof: By *Lemma 5.5*, it is sufficient to construct an automaton accepting exactly the $N+1$-ary Σ_p-trees satisfying the conditions (1)-(7), where $N \leq k$ 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 (1), (2), (5), and (6).

For $1 \leq n \leq N$, let A_n be a deterministic automaton on finite strings which accepts the regular set $C(a_n;q_n?)$. The A_n's can be constructed to have no more than $O(\exp k)$ states. 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 $x;\{xn0^m\}_{m \geq 0}$, looking for an initial segment which the program $a_n;q_n?$ fits. 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 $a_n;q_n?$ fits any finite path starting with x. The C_n's and D_n's can be constructed to have no more than $O(\exp k)$ states.

Given a deterministic m state automaton recognizing a regular set X not containing the empty string, there is a construction, due to McNaughton [9], of a deterministic automaton on infinite strings, with no more than $O(\exp m)$ states, which accepts exactly the infinite strings not in X^∞. For $\Delta a \in cl(p)$, let E_a be the result of applying McNaughton's construction to a deterministic automaton accepting $\{\eta_0 b_1 \cdots b_k \eta_k \in C(a) \mid k \geq 1\}$. 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 fits. F_a can be constructed to have no more than $O(\exp \exp k)$ states.

Finally, the automaton B and the C_n's, D_n's, and F_a's can be combined in a cross-product construction to yield the desired A_p. A_p has no more than $O(\exp \exp k)$ states and can be constructed in time $O(\exp \exp k)$. ∎

Theorem 5.8: The satisfiability problem for *delta-converse-PDL* is decidable in time $O(\exp^8 k)$, where k is the length of the formula tested.

Proof: Given a formula p of length k, *Theorem 5.7* constructs a two-way automaton A_p on infinite $N+1$-ary trees with no more than $O(\exp \exp k)$ states such that A_p accepts some tree if and only if p is satisfiable. By *Theorem 5.10*, there is an equivalent one-way automaton B on infinite $N+1$-ary trees with no more than $O(\exp^6 k)$ states. It is straightforward to construct a one-way automaton C on infinite binary trees with no more than $O(N+1 \exp^6 k) = O(\exp^6 k)$ states, whose emptiness problem is equivalent to B's. The emptiness problem for one-way automata on infinite binary trees is decidable in time $O(\exp \exp m)$, where m is the number of states [8, 18]. Therefore, *delta-*

We shall prove later (see *Lemma 5.3*) that *delta-converse-PDL* satisfies a tree model property; every satisfiable *delta-converse-PDL* formula has a tree model. For *delta-PDL* a stronger property holds: every satisfiable *delta-PDL* formula has a one-way tree model (see *Corollary 5.4*).

Theorem 2.4: *Converse-PDL* (and hence also *delta-converse-PDL*) does not satisfy the one-way tree model property; there is a satisfiable *converse-PDL* formula which is not satisfied in any one-way tree model.

Proof: Consider the satisfiable formula $P \,\&\, \langle A\rangle\langle A^{-}\rangle\neg P$. Suppose $u \models_S P \,\&\, \langle A\rangle\langle A^{-}\rangle\neg P$, where S is a one-way tree model. Then $u \models_S P$ and there is an immediate successor v of u such that $v \models_S \langle A^{-}\rangle\neg P$, so that there must be a state w such that $w\langle A\rangle_S v$ and $w \models_S \neg P$. Since S is a one-way tree model, w must be the parent of v, so $w = u$. But this is impossible, since we have $u \models_S P$ and $w \models_S \neg P$. ∎

The remainder of this chapter relates the expressive power of *delta-PDL* to that of two other propositional logics of programs: the *Propositional Algorithmic Logic* (*PAL*) of Mirkowska [12] and the *Unified Temporal Logic of Branching Time* (*UB*) of Ben-Ari, Manna, and Pnueli [1]. *UB* is an intensional logic of programs, as opposed to *PDL* and *PAL*, which are extensional. Programs appear explicitly in the formulae of *PDL* and *PAL*, and different formulae can refer to completely different programs. The formulae of a temporal logic do not explicitly refer to programs; rather, every formula is taken to refer to a single program, which is fixed by the choice of a *UB*-structure.

Definition: The formulae, Π_{UB}, of *UB*, are defined inductively as follows:

(1) $\Pi_0 \subseteq \Pi_{UB}$
(2) If $p, q \in \Pi_{UB}$, then $\neg p$, $p \vee q$, $\exists Xp$, $\exists Fp$, $\exists Gp \in \Pi_{UB}$

Definition: A *UB*-structure is a tuple $S = \langle U, \models_S, \Rightarrow_S\rangle$ where U is a set of states, $\models_S$ is a satisfiability relation on the atomic propositions, and $\Rightarrow_S$ is a *total* binary relation on U (i.e., for every state u there is at least one state v such that $u \Rightarrow_S v$).

Definition: Given a *UB*-structure $S = \langle U, \models_S, \Rightarrow_S\rangle$, $\models_S$ can be extended to all *UB* formulae as follows.

(1) $u \models_S \neg p$ iff not $u \models_S p$.
(2) $u \models_S p \vee q$ iff $u \models_S p$ or $u \models_S q$.
(3) $u \models_S \exists Xp$ iff $\exists v.\ u \Rightarrow_S v$ and $v \models_S p$.
(4) $u \models_S \exists Fp$ iff $\exists v.\ u \Rightarrow_S^* v$ and $v \models_S p$
(5) $u \models_S \exists Gp$ iff there is an infinite sequence $\{u_n\}_{n\geq 0}$ of states such that $u_0 = u$ and for all n, $u_n \models_S p$ and $u_n \Rightarrow_S u_{n+1}$.

The logic *UB* is a temporal logic of discrete branching time; given a program a, the binary relation $\Rightarrow_S$ relates computation states at time t to possible computation states at the next time $t + 1$.

(2) if x_{i+1} is the predecessor of x_i, then the inverse of $b_{i+1} \in f(x_i)$.

Remark: A program a fits a singleton path x if and only if there is a compressed execution sequence $\eta \in C(a)$, consisting of a single set of subformulae of p, such that $\eta \subseteq f(x)$. If f is the image of a one-way scheme and if a is a *converse*-free program, then a can fit only forward paths and only condition (1) is needed to determine the forward paths which a fits.

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

Lemma 5.5: A Σ_p-tree f is an image for p if and only if the following conditions are satisfied.

(1) $p \in f(\Lambda)$.

(2) for $\neg q \in cl(p)$, $\neg q \in f(x)$ if and only if $q \notin f(x)$.

(3) if $\langle a_n\rangle q_n \in f(x)$, then there is an initial segment π of the infinite path $x;\{xn0^m\}_{m\geq 0}$ such that $a_n;q_n?$ fits π.

(4) if $\langle a_n\rangle q_n \notin f(x)$, then for all finite paths π starting at x, $a_n;q_n?$ does not fit π.

(5) for $\Delta a \in cl(p)$, $\Delta a \in f(x)$ if and only if $\langle a\rangle \Delta a \in f(x)$.

(6) for $\Delta a \in cl(p)$, if a fits the singleton path x, then $\Delta a \in f(x)$.

(7) for $\Delta a \in cl(p)$, if $\Delta a \notin f(x)$, then for all infinite paths π starting at x, a does not repeatedly fit π.

Proof: We leave it to the reader to verify that an image for p satisfies (1) - (7). Conversely, given a $N+1$-ary Σ_p-tree f satisfying (1) - (7), we can define a two-way tree structure $S = \langle T_{N+1}, \models_S, \langle\rangle_S\rangle$ 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. We proceed, using structural induction on formulae and conditions (2) - (7), to establish that for all $x \in T_{N+1}$ and $q \in cl(p)$, $x \models_S q$ iff $q \in f(x)$.

If q is an atomic subformula P, then $x \models_S P$ iff $P \in f(x)$ follows from the definition of S. If q is a negated subformula $\neg r$, then $x \models_S \neg r$ iff $\neg r \in f(x)$ follows from condition (2). If q is a diamond subformula $\langle a_n\rangle q_n$, then $(x \models_S \langle a_n\rangle q_n) \rightarrow (\langle a_n\rangle q_n \in f(x))$ follows from condition (4), and $(\langle a_n\rangle q_n \in f(x)) \rightarrow (x \models_S \langle a_n\rangle q_n)$ follows from condition (3). If q is a delta subformula Δa, then $(x \models_S \Delta a) \rightarrow (\Delta a \in f(x))$ follows from conditions (4), (6), and (7), and $(\Delta a \in f(x)) \rightarrow (x \models_S \Delta a)$ follows from conditions (3) and (5). By condition (1), $\Lambda \models_S p$, and by condition (3), for $1 \leq n \leq N$, if $x \models_S \langle a_n\rangle q_n$, then $\exists y.\ x \leq y < xn0^\infty$ & $x\langle a_n;q_n?\rangle_S y$. Hence S is a scheme for p. ∎

Corollary 5.6: If p is a *delta-PDL* formula, then a Σ_p-tree f is a one-way image for p if and only if conditions (1) - (7) above are satisfied and, for all x, $f(x)$ contains no negative literals.

The formula $\exists Xp$ is true in a state at time t if that state can become, at time $t + 1$, a state in which p is true. The formula $\exists Fp$ is true in a state at time t if that state is or can become, at some later time $t + n$, a state in which p is true. The formula $\exists Gp$ is true in a state at time t if from that state there is an infinite sequence of successive states in which p is true. We can define three dual formulae: $\forall Xp =_{df} \neg\exists X\neg p$, $\forall Fp =_{df} \neg\exists F\neg p$, and $\forall Gp =_{df} \neg\exists G\neg p$. The formula $\forall Xp$ is true in a state if p is true in all possible next states. The formula $\forall Fp$ is true in a state if p is true in that state and in all possible future states. The formula $\forall Gp$ is true in a state if, from that state, every chain of successive states contains a state in which p is true.

Definition: Let A be a fixed atomic program. Let $\dagger$: $\Pi_{UB} \rightarrow \Pi$ be a translation defined as follows.

(1) $P\dagger = P$
(2) $(\neg p)\dagger = \neg(p\dagger)$
(3) $(p \vee q)\dagger = (p\dagger \vee q\dagger)$
(4) $(\exists Xp)\dagger = \langle A\rangle(p\dagger)$
(5) $(\exists Fp)\dagger = \langle A^*\rangle(p\dagger)$
(6) $(\exists Gp)\dagger = \Delta((p\dagger)?;A)$

Definition: If $S = \langle U, \models_S, \Rightarrow_S\rangle$ is a *UB*-structure, then let $S\dagger = \langle U, \models_{S\dagger}, \langle\rangle_{S\dagger}\rangle$ be any structure in which $\models_{S\dagger} = \models_S$ and $\langle A\rangle_{S\dagger} = \Rightarrow_S$.

Theorem 2.5: *UB* is embeddable in *delta-PDL*; if p is a *UB* formula satisfied at a state u in a *UB*-structure S, then $u \models_{S\dagger} p\dagger$. Further, p has a *UB*-model if and only if $[A^*]\langle A\rangle true \;\&\; p\dagger$ is satisfiable.

Proof: By structural induction on formulae. ∎

Propositional Algorithmic Logic is very similar to *PDL*. One major difference is that the semantics of programs in *PAL* is defined in terms of computation sequences rather than binary relations as in *PDL* (one might say that *PAL* has an operational semantics and *PDL* a denotational semantics). The other major difference is that *PAL* contains a powerful total correctness assertion for nondeterministic programs, $\Box(a)p$, which is true when every execution sequence of a terminates in a state in which p is true. Since the truth value of $\Box(a)p$ depends on the presence or absence of nonterminating execution sequences of a, *PDL* does not express *PAL*. *Delta-PDL*, however, does express *PAL*.

arbitrarily. Finally, given φ, define a structure $T = \langle T_{N+1}, \models_T, \langle\rangle_T \rangle$ by letting $x \models_T P$ if and only if $\varphi(x) \models_S P$ and letting $x\langle A\rangle_T y$ if and only if x and y are neighbors and $\varphi(x)\langle A\rangle_S \varphi(y)$. By construction T is a scheme for p. ∎

Corollary 5.4: Every satisfiable *delta-PDL* formula has a one-way scheme.

Proof: Given a satisfiable delta-PDL formula p, construct the map φ as in the proceeding proof, but define $T = \langle T_{N+1}, \models_T, \langle\rangle_T \rangle$ by letting $x \models_T P$ if and only if $\varphi(x) \models_S P$ and letting $x\langle A\rangle_T y$ if and only if y is a successor of x and $\varphi(x)\langle A\rangle_S \varphi(y)$. By construction T is a one-way scheme for p. ∎

Schemes are easily transformed into trees suitable for input to automata on infinite trees. The trees obtained in this way are automaton recognizable; this fact leads immediately to decision procedures for *delta-PDL* and *delta-converse-PDL*.

Definition: If p is a *delta-converse-PDL* formula, Π_p denotes the set of literals appearing in p. Let $\Sigma_p = \mathit{Powerset}(\mathit{cl}(p) \cup \Pi_p)$.

Definition: Given a scheme $S = \langle T_{N+1}, \models_S, \langle\rangle_S \rangle$ for a *delta-converse-PDL* formula p, the *image* of S is the $N+1$-ary Σ_p-tree f such that for all $x \in T_{N+1}$, $f(x) = \{q \in \mathit{cl}(p) \mid x \models_S q\} \cup \{a \in \Pi_p \mid y\langle a\rangle_S x$ where y is the predecessor of $x\}$. An *image for p* is an image of a scheme for p.

Remark: If the scheme S is one-way and if f is the image of S, then for all x, $f(x)$ contains no negative literals.

It is technically convenient to define a version of execution sequences in which all subsequences of tests are compressed into single sets of formulae. Note that it is no more difficult for a finite automaton to recognize the compressed execution sequences of a program than the ordinary execution sequences: if the latter set is accepted by a n state automaton on finite strings, then so is the former.

Definition: Given a formula p, a *compressed* (with respect to p) execution sequence is a sequence $\eta_0 b_1 \eta_1 \cdots \eta_{n-1} b_n \eta_n$ of alternating literals and sets of subformulae of p, beginning and ending with sets. The set of compressed execution sequences for a program a is $C(a) = \{\eta_0 b_1 \eta_1 \cdots \eta_{n-1} b_n \eta_n \mid$ there exists $q_{01}? \cdots q_{0k_0}? b_1 q_{11}? \cdots q_{1k_1}? b_2 \cdots b_n q_{n1}? \cdots q_{nk_n}? \in L(a)$, where each b_i is a literal, such that $\eta_i = \{q_{i1}, \ldots, q_{ik_i}\}$, for $0 \leq i \leq n\}$.

Definition: Given a Σ_p-tree f, a program a *fits* a path $\pi = \{x_i\}_{0 \leq i \leq n}$ if and only if there is an compressed execution sequence $\eta_0 b_1 \eta_1 \cdots \eta_{n-1} b_n \eta_n \in L(a)$ such that for $0 \leq i < n$, $\eta_i \subseteq f(x_i)$ and for $0 \leq i < n$,

(1) if x_{i+1} is a successor of x_i, then $b_{i+1} \in f(x_{i+1})$.

Definition: The set of programs, Π_{PAL}, and the set of formulae, Φ_{PAL}, of *PAL* are defined inductively as follows.

Π_{PAL}: (1) $\Pi_0 \subseteq \Pi_{PAL}$

(2) If $a, b \in \Pi_{PAL}$, then $a;b$, $a \cup b$, $a^* \in \Pi_{PAL}$

(3) If $p \in \Phi_{PAL}$ and $a, b \in \Pi_{PAL}$, then $p?$, *if* p *then* a *else* b, *while* p *do* $a \in \Pi_{PAL}$

Φ_{PAL}: (1) $\Phi_0 \subseteq \Phi_{PAL}$

(2) If $p, q \in \Phi_{PAL}$, then $\neg p \in \Phi_{PAL}$

(3) If $a \in \Pi_{PAL}$ and $p \in \Phi_{PAL}$, then $\Diamond(a)p$, $\Box(a)p \in \Phi_{PAL}$

Definition: If S is a structure, then a *configuration* is a pair $\langle u, \pi \rangle$, where u is a state of S and $\pi = \langle a_1, \ldots, a_k \rangle$ is a (possibly empty) sequence of programs. The configuration $\langle u, \pi \rangle$ is *final* if and only if π is empty.

Definition: Given a structure S, $\models_S$ can be extended to arbitrary *PAL* formulae and a binary relation $\Rightarrow_S$ on configurations can be defined as follows. If $\langle u, \pi \rangle \Rightarrow_S^* \langle v, \tau \rangle$, then we say that $\langle u, \pi \rangle$ *yields* $\langle v, \tau \rangle$. If $\langle u, \pi \rangle$ is not final and in addition there is no configuration $\langle v, \tau \rangle$ such that $\langle u, \pi \rangle \Rightarrow_S \langle v, \tau \rangle$, then $\langle u, \pi \rangle$ is a *failing* configuration.

(1) $u \models_S \neg p$ iff not $u \models_S p$.

(2) $u \models_S \Diamond(a)p$ iff $\langle u, \langle a, p? \rangle \rangle$ yields a final configuration.

(3) $u \models_S \Box(a)p$ iff $\langle u, \langle a, p? \rangle \rangle$ yields neither a failing configuration nor an infinite chain of configurations.

(4) $\langle u, \langle A, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle v, \langle a_1, \ldots, a_k \rangle \rangle$ iff $u \langle A \rangle_S v$.

(5) $\langle u, \langle a;b, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle a, b, a_1, \ldots, a_k \rangle \rangle$.

(6) $\langle u, \langle a \cup b, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle c, a_1, \ldots, a_k \rangle \rangle$ iff $c = a$ or $c = b$.

(7) $\langle u, \langle a^*, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle a_1, \ldots, a_k \rangle \rangle$.

(8) $\langle u, \langle a^*, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle a, a^*, a_1, \ldots, a_k \rangle \rangle$.

(9) $\langle u, \langle p?, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle a_1, \ldots, a_k \rangle \rangle$ iff $u \models_S p$.

(10) $\langle u, \langle$*if* p *then* a *else* $b, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle c, a_1, \ldots, a_k \rangle \rangle$ iff either $u \models_S p$ and $c = a$ or $u \models_S \neg p$ and $c = b$.

(11) $\langle u, \langle$*while* p *do* $a, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle a_1, \ldots, a_k \rangle \rangle$ iff $u \models_S p$.

(12) $\langle u, \langle$*while* p *do* $a, a_1, \ldots, a_k \rangle \rangle \Rightarrow_S \langle u, \langle a,$ *while* p *do* $a, a_1, \ldots, a_k \rangle \rangle$ iff $u \models_S \neg p$.

Definition: If a is a *delta-converse-PDL* program, then $L(a)$, the set of *execution sequences* of a, is defined inductively as follows:

(1) $L(A) = \{A\}$
(2) $L(a;b) = L(a);L(b)$
(3) $L(a \cup b) = L(a) \cup L(b)$
(4) $L(a^*) = (L(a))^*$
(5) $L(q?) = \{q\}$
(6) $L(A^-) = \{A^-\}$
(7) $L((a;b)^-) = L(b^-;a^-)$
(8) $L((a \cup b)^-) = L(a^- \cup b^-)$
(9) $L((a^*)^-) = L((a^-)^*)$
(10) $L((q?)^-) = \{q\}$
(11) $L((a^-)^-) = L(a)$

Lemma 5.2: For all structures $S = \langle U, \models_S, \langle\rangle_S \rangle$ and programs a, $u\langle a\rangle_S v$ if and only if there is an execution sequence $b_1 \cdot \cdot \cdot b_k \in L(a)$ and a sequence of states $\{u_n\}_{0 \le n \le k}$ such that $u_0 = u$, $u_k = v$ and $u_n \langle b_{n+1} \rangle_S u_{n+1}$ for $0 \le n < k$.

Proof: By structural induction on programs. ∎

If p is a satisfiable *delta-converse-PDL* formula, *Theorem 5.3* shows that p has a special tree model, called a scheme, which is easily transformed into a tree suitable as input to a two-way automaton. A scheme is a tree structure in which p is satisfied at the root and diamond subformulae of p are satisfied along specific paths. If p is *converse*-free, i.e., a *delta-PDL* formula, then *Corollary 5.4* shows that p has a one-way scheme, i.e., a scheme which is a one-way tree structure.

Definition: If p is a *delta-converse-PDL* 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 $S = \langle T_{N+1}, \models_S, \langle\rangle_S \rangle$ such that $\Lambda \models_S p$ and for all states x, if $x \models_S \langle a_n\rangle q_n$ then $\exists y.\ x \le y < xn0^\infty$ & $x\langle a_n;q_n?\rangle_S y$.

Theorem 5.3: Every satisfiable *delta-converse-PDL* formula has a scheme.

Proof: Suppose $u_0 \models_S p$, where $S = \langle U, \models_S, \langle\rangle_S \rangle$. We construct a a map φ: $T_{N+1} \rightarrow U$ inductively as follows. Let $\varphi(\Lambda) = u_0$. Inductively, if x is in V and $\varphi(x) = u$, 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 $u\langle a_n;q_n?\rangle_S v$. By *Lemma 5.2*, there is a sequence of states $\{u_i\}_{0 \le i \le k}$ and an execution sequence $b_1 \cdot \cdot \cdot b_k \in L(a_n;q_n?)$ such that $u_0 = u$, $u_n = v$, and and $u_i\langle b_{i+1}\rangle_S u_{i+1}$ for $0 \le i < k$. Let m be the number of literals in $b_1 \cdot \cdot \cdot b_k$. For $1 \le i \le m$, let $\varphi(xn0^{i-1}) = u_j$, where j is the index of the i^{th} literal in $b_1 \cdot \cdot \cdot b_k$. For $i > m$, let $\varphi(xn0^{i-1})$ be chosen

Remark: Note that $\Diamond(a)p$ and $\Box(a)p$ are not dual to one another, i.e, $\Box(a)p$ is not equivalent to $\neg\Diamond(a)\neg p$. Note also that $\Box(\textit{if } p \textit{ then } a \textit{ else } b)q$ is sometimes true and sometimes false, but that $\Box((p?;a)\cup(\neg p?;b))q$ is always false, since
$\langle u, \langle((p?;a)\cup(\neg p?;b));q?\rangle\rangle$ yields $\langle u, \langle p?, a, q?\rangle\rangle$ and $\langle u, \langle \neg p?, a, q?\rangle\rangle$, one of which must be a failing configuration. Hence *if p then a else b* cannot be defined, in *PAL*, to be an abbreviation of $(p?;a)\cup(\neg p?;b)$. Similarly, *while p do a* cannot be defined, in *PAL*, to be an abbreviation of $(p?;a^*);\neg p?$.

Definition: For each *PAL* program a, define a *PAL* formula *fail(a)*) as follows.

(1) $\textit{fail}(A) = \neg\Diamond(A)\textit{true}$
(2) $\textit{fail}(a;b) = \textit{fail}(a) \vee \Diamond(a)\textit{fail}(b)$
(3) $\textit{fail}(a\cup b) = \textit{fail}(a) \vee \textit{fail}(b)$
(4) $\textit{fail}(a^*) = \Diamond(a^*)\textit{fail}(a)$
(5) $\textit{fail}(p?) = \neg p$
(6) $\textit{fail}(\textit{if } p \textit{ then } a \textit{ else } b) = (p \;\&\; \textit{fail}(a)) \vee (\neg p \;\&\; \textit{fail}(b))$
(7) $\textit{fail}(\textit{while } p \textit{ do } a) = \Diamond((p?;a)^*)(p \;\&\; \textit{fail}(a))$

Lemma 2.6: For all structures S, states u, and *PAL* programs a, $u \models_S \textit{fail}(a)$ if and only if $\langle u, \langle a\rangle\rangle$ yields a failing configuration.

Proof: By structural induction on programs. ∎

Definition: Let ‡ be a translation from *PAL* formulae and programs to *delta-PDL* formulae and programs defined as follows.

(1) $P‡ = P$
(2) $(\neg p)‡ = \neg(p‡)$
(3) $(\Diamond(a)p)‡ = \langle a‡\rangle(p‡)$
(4) $(\Box(a)p)‡ = \neg((\textit{fail}(a;p?))‡ \vee \infty(a‡))$
(5) $A‡ = A$
(6) $(a;b)‡ = (a‡);(b‡)$
(7) $(a\cup b)‡ = (a‡)\cup(b‡)$
(8) $(a^*)‡ = (a‡)^*$
(9) $(p?)‡ = (p‡)?$
(10) $(\textit{if } p \textit{ then } a \textit{ else } b)‡ = \textit{if } p‡ \textit{ then } a‡ \textit{ else } b‡$
(11) $(\textit{while } p \textit{ do } a)‡ = \textit{while } p‡ \textit{ do } a‡$

5 Satisfiability and Finite Models

In this chapter the automata theoretic results of the previous two chapters are used to obtain decision procedures for *delta-PDL* and *delta-converse-PDL*. The notion of a finitely generable tree is then employed to establish a finite model theorem for *delta-PDL* and a finite representation theorem for *delta-converse-PDL*. First, however, we precisely define the informal notions of the subformulae of a formula and the execution sequences of a program.

Definition: If p is a *delta-converse-PDL* formula, then $cl(p)$, the *Fischer-Ladner closure* of p, is the least set of formulae such that

(1) $p \in cl(p)$
(2) if $\neg q \in cl(p)$, then $q \in cl(p)$
(3) if $\langle A\rangle q \in cl(p)$ or $\langle A^{-}\rangle q \in cl(p)$, then $q \in cl(p)$
(4) if $\langle a;b\rangle q \in cl(p)$, then $\langle a\rangle\langle b\rangle q \in cl(p)$
(5) if $\langle (a;b)^{-}\rangle q \in cl(p)$, then $\langle b^{-};a^{-}\rangle q \in cl(p)$
(6) if $\langle a \cup b\rangle q \in cl(p)$, then $\langle a\rangle q$, $\langle b\rangle q \in cl(p)$
(7) if $\langle (a\cup b)^{-}\rangle q \in cl(p)$, then $\langle a^{-} \cup b^{-}\rangle q \in cl(p)$
(8) if $\langle a^{*}\rangle q \in cl(p)$, then q, $\langle a\rangle\langle a^{*}\rangle q \in cl(p)$
(9) if $\langle (a^{*})^{-}\rangle q \in cl(p)$, then $\langle (a^{-})^{*}\rangle q \in cl(p)$
(10) if $\langle r?\rangle q \in cl(p)$, then r, $q \in cl(p)$
(11) if $\langle (r?)^{-}\rangle q \in cl(p)$, then $\langle r?\rangle q \in cl(p)$
(12) if $\Delta a \in cl(p)$, then $\langle a\rangle \Delta a \in cl(p)$

Lemma 5.1: If p is a *delta-converse-PDL* formula of length n, then $cl(p)$ contains at most n formulae.

Proof: A straightforward extension of the corresponding proof for *PDL* [7]. ∎

Definition: The elements of $cl(p)$ are called the *subformulae* of p; this can be misleading, since $\langle a\rangle\langle a^{*}\rangle q$ 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: Abusing predicate calculus terminology, we define a *literal* to be either an atomic program or the converse of an atomic program. Atomic programs will sometimes be called *positive* literals and converses of atomic programs *negative* literals. The *inverse* of a positive literal A is A^{-}; the *inverse* of a negative literal A^{-} is A.

Programs in *delta-converse-PDL* are extended regular expressions over literals and tests, so each program denotes a regular set, the set of its execution sequences.

Lemma 2.7: For all structures S, states u of S, and *PAL* programs a, $\langle u, \langle a \rangle\rangle$ yields $\langle v, \langle\rangle\rangle$ if and only $u\langle a\ddagger\rangle_S v$.

Proof: By structural induction on programs. ∎

Lemma 2.8: For all structures S, states u of S, and *PAL* programs a, $\langle u, \langle a \rangle\rangle$ yields an infinite chain if and only $u \models_S \infty(a\ddagger)$.

Proof: By structural induction on programs. ∎

Theorem 2.9: *PAL* is embeddable in *delta-PDL*, i.e., for all structures S, states u of S, and *PAL* formulae p, $u \models_S p$ if and only $u \models_S p\ddagger$.

Proof: Follows directly from *Lemmas 2.6, 2.7,* and *2.8.* ∎

Proof: It is easy to construct, in time $O(\exp \exp m)$, a one-way automaton C, with no more than $O(\exp \exp m)$ states, which accepts an infinite $(\Sigma \times C_S)$-tree $f \times g$ exactly when g is a plan for f. It is straightforward to construct, also in time $O(\exp \exp m)$, a nondeterministic automaton D on infinite strings, with no more than $O(\exp \exp m)$ states, which, when run along an infinite forward path of an infinite N-ary C_S-tree g, accepts exactly when that path violates either of the two conditions for goodness. McNaughton gives a construction which, given a nondeterministic automaton on infinite strings with k states, produces, in time $O(\exp \exp k)$, a deterministic automaton on infinite strings, with no more than $O(\exp \exp k)$ states, which accepts exactly the complement of the set of strings accepted by the original automaton [9]. Let E be the result of applying McNaughton's construction to D; let F be that automaton on infinite trees which runs E down every infinite forward path, so that F accepts g exactly when g is good. Finally, the desired automaton B, given an input tree f: $T_N \rightarrow \Sigma$, nondeterministically guesses a map g: $T_N \rightarrow C_S$ while simultaneously running the automata C on $f \times g$ and F on g. By *Lemmas 4.4* and *4.9*, A and B accept the same trees. The automaton B has no more than $O(\exp^4 m)$ states and can be constructed from A in time $O(\exp^4 m)$. ∎

3 One-Way Automata on Infinite Trees

Automata on infinite trees, called *one-way* automata in this chapter to distinguish them from the two-way automata defined in the next chapter, have been extensively studied [8, 18, 19]. We briefly review the fundamental definitions and theorems.

Definition: The set $T_N = \{0, 1, \ldots, N\text{-}1\}^*$ of strings of the first N nonnegative integers can be viewed as an infinite N-ary tree, in which the empty string Λ is the root and each string (or node) $x \in T_N$ has as its successors the strings $x0, \ldots, x(N\text{-}1)$. The descendant relation is the reflexive transitive closure of the successor relation; we write $y \geq x$ when y is a descendant of x (alternatively, we can write $x \leq y$ and say that x is an ancestor of y).

Definition: A *finite* (*infinite*) *forward 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: If Σ is a finite alphabet, then an *infinite N-ary* Σ*-tree* is a function f: $T_N \rightarrow \Sigma$.

Definition: A (*nondeterministic*) *one-way automaton* A on infinite N-ary Σ-trees is a tuple $\langle S, s, M, G \rangle$ where

S is the set of states.
$s \in S$ is the initial state.
M: $S \times \Sigma \rightarrow$ *Powerset*(S^N) is the next state function.
$G \subseteq$ *Powerset*(S) is a set of accepting subsets.

Definition: A *run* of A on an infinite N-ary Σ-tree f is a function ρ: $T_N \rightarrow S$ such that $\rho(\Lambda) = s$ and for all $x \in T_N$, $\langle \rho(x0), \ldots, \rho(x(n\text{-}1)) \rangle \in M(\rho(x), f(x))$.

Definition: If ρ is a run of A on f and π is an infinite forward path, then $Inf(\rho, \pi) = \{q \in S \mid \rho(x) = q$ for infinitely many x on $\pi\}$.

Definition: 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 forward paths π, $Inf(\rho, \pi) \in G$.

Theorem 3.1: The emptiness problem for an N-ary infinite tree automaton A with m states, i.e., the problem of deciding whether or not A accepts any tree at all, can be decided in time $O(\exp \exp mN)$. ∎

Proof: Given an m state automaton on infinite N-ary trees, it is a straightforward exercise to construct an $O(mN)$ state automaton on infinite binary trees, such that the two automata have equivalent emptiness problems. Hossley and Rackoff [8] give a decision procedure for the emptiness problem for automata on infinite binary trees which runs in time $O(\exp \exp n)$, where n is the number of states of the automaton tested. ∎

these two circuits (since $Y, Z \neq \emptyset$, the loops cannot be singletons). The required loop for the join is $x;\sigma;x;\tau;x$. In the case of rule (5), $\langle s, X, t\rangle$ is the expansion of a circuit $\langle t, Y, u\rangle \in g_{min}(y)$, where y is a neighbor of x. Inductively, there is a loop π on y for $\langle t, Y, u\rangle$. The required loop for the expansion is $x;\pi;x$. ∎

Lemma 4.8: For all paths $\tau;\pi$ ending in a loop π on x, $\rho(\tau;x \mid \tau;\pi) \in g_{min}(x)$.

Proof: By induction on the length of π. Let $s = \rho(\tau;x)$. If π is the singleton x, then by *Lemma 4.6*, $\rho(\tau;x \mid \tau;\pi) = \langle s\rangle \in g_{min}(x)$. If $\pi = x;\mu;x$ where μ is a loop on a neighbor y of x, then inductively, $\rho(\tau;x;y \mid \tau;x;\mu) \in g_{min}(y)$. Then, by rule (5) for plans, $\rho(\tau;x \mid \tau;x;\pi) \in g_{min}(x)$. If μ is not a loop, then by *Lemma 4.1*, μ contains x, i.e. $\mu = \varphi;x;\psi$. Inductively, $\rho(\tau;x \mid \tau;x;\varphi;x)$, $\rho(\tau;x;\varphi;x \mid \tau;\pi) \in g_{min}(x)$. Then, by rule (4) for plans, $\rho(\tau;x \mid \tau;\pi) \in g_{min}(x)$. ∎

Lemma 4.9: The automaton A accepts an infinite tree f if and only if the minimal plan g_{min} for A on f is good.

Proof: First, suppose A does not accept f. Then there is an infinite path π such that $Inf(\rho, \pi) \notin G$, where ρ is the run of A on f. If π is cyclic on x, then $\pi = \mu;\sigma;\tau$ where σ is a loop on x and $\rho(\mu;x) = \rho(\mu;\sigma) \in \rho(\mu;x, \mu;\sigma) = Inf(\rho, \pi)$. Then, by *Lemma 4.8*, $\rho(\mu;x \mid \mu;\sigma) \in g_{min}(x)$, where $\rho(\mu;x) \in \rho(\mu;x, \mu;\sigma) \notin G$, so g_{min} is not good. If, on the other hand, π is acyclic, then by *Lemma 4.2* there is an infinite forward path $\{x_n\}$ such that $\pi = \sigma;\tau_0; \ldots ;\tau_n; \ldots$, where each τ_n is a loop on x_n. Let $\zeta = \{\rho(\tau_1; \ldots ;\tau_{n-1};x_n \mid \tau_1; \ldots ;\tau_n)\}_{n\geq 0}$. We leave it to the reader to show that ζ is a series for g_{min} on $\{x_n\}$, but that $Sum(\zeta) \notin G$, so that g_{min} is not good. Conversely, suppose that g_{min} is not good. Then either there is a node x and a circuit $\langle s, X, s\rangle \in g_{min}(x)$ such that $s \in X \notin G$ or there is an infinite forward path $\{x_n\}$ and a series $\zeta = \{\langle s_n, X_n, t_n\rangle\}$ for g_{min} on $\{x_n\}$ such that $Sum(\zeta) \notin G$. If the first case holds, then by *Lemma 4.7*, there is a loop $x;\pi;x$ such that for all paths τ ending in x, if $\rho(\tau) = s$, then $\rho(\tau, \tau;\pi;x) = X$ and $\rho(\tau;\pi;x) = s$. By *Lemma 4.5*, there is a path τ ending in x such that $\rho(\tau) = s$. Let $\mu = \tau;\pi;x;\pi;x;\pi;x; \cdots$ We leave it to the reader to show that A rejects f, because $Inf(\rho, \mu) = X \notin G$. If the the second case holds, then, by *Lemma 4.5*, there is an infinite path $\mu = \tau_1;\tau_2; \cdots$ such that for all n, τ_n is a loop on x_n and $\langle s_n, X_n, t_n\rangle = \rho(\tau_1; \cdots ;\tau_{n-1};x_n \mid \tau_1; \cdots ;\tau_n)$. Then $Inf(\rho, \mu) = Sum(\zeta) \notin G$, so that A rejects f in this case also. ∎

Definition: If f is an infinite N-ary Σ_0-tree and g is an infinite N-ary Σ_1-tree, then the *product tree* $f \times g$ is an infinite N-ary $(\Sigma_0 \times \Sigma_1)$-tree defined by $(f \times g)(x) = \langle f(x), g(x)\rangle$.

Theorem 4.10: Given a deterministic two-way automaton A with m states, there is a nondeterministic one way automaton B with no more than $O(\exp^4 m)$ states which accepts exactly the trees accepted by A. Further, B can be constructed in time $O(\exp^4 m)$.

The decision procedure for the emptiness problem depends crucially on the fact that every nonempty set of trees accepted by an automaton contains a finitely generable tree, i.e., a tree obtained by unwinding a finite graph. In chapter 5 we will use this fact to establish a finite model property for *delta-PDL* and a finite representation property for *delta-converse-PDL.*

Definition: A *frontier* of T_N is a maximal incomparable subset X of T_N, i.e., a subset X such that every element of T is either a descendant or an ancestor of some member of X, but no member of X is the descendant of any other member of X.

Definition: A *finite subtree* of T_N is a subset T of T_N such that $T = \{x \in T_N \mid x \leq y \text{ for some } y \in X\}$, where X is a frontier. The frontier of T, *front*(T), is X, and the interior of T, *int*(T), is $T - \mathit{front}(T)$. A *finite* N-ary Σ-tree is a map f: $T \rightarrow \Sigma$, where T is a finite subtree of T_N.

Definition: Given an automaton A on infinite N-ary Σ-trees and a finite N-ary Σ-tree f: $T \rightarrow \Sigma$, a run of A on f is a function ρ: $T \rightarrow S$ such that $\rho(\Lambda) = s$ and for all $x \in \mathit{int}(T)$, $\langle \rho(x0), \ldots, \rho(x(n\text{-}1))\rangle \in M(\rho(x), f(x))$.

Definition: A *generating map* for a finite subtree T of T_N is a function J: *front*(T) $\rightarrow$ *int*(T). Every generating map defines a unique function J^*: $T_N \rightarrow T$ as follows:

$$\begin{aligned} J^*(\Lambda) &= \Lambda \\ J^*(xn) &= J^*(x)n \text{ if } J^*(x) \in \mathit{int}(T), \\ &= J(J^*(x))n \text{ if } J^*(x) \in \mathit{front}(T). \end{aligned}$$

Definition: An infinite Σ-tree f is *finitely generable* if and only if there is a finite subtree T of T_N and a generating map J such that $f = f \circ J^*$.

Theorem 3.2 [8, 19]: If an automaton accepts at least one tree, then it accepts a finitely generable tree.

Below we present an alternative formulation of automata on infinite trees. Pairs automata are equivalent to ordinary automata in the following sense: for every ordinary automaton, there is a pairs automaton which accepts exactly the same trees, and conversely.

Definition: If $\Omega = \{\langle L_n, U_n\rangle\}_{1 \leq n \leq k}$ is a finite sequence of pairs of subsets of some set S, then let $F_\Omega = \{X \subseteq S \mid X \cap L_n = \emptyset \ \& \ X \cap U_n \neq \emptyset \text{ for some } n\}$. Let $G_\Omega = \mathit{Powerset}(S) - F_\Omega = \{X \subseteq S \mid X \cap U_n \neq \emptyset \rightarrow X \cap L_n \neq \emptyset \text{ for all } n\}$. Note that G_Ω is closed under unions, i.e., if $X, Y \in G_\Omega$, then $X \cup Y \in G_\Omega$.

Definition: A *pairs automaton* [8, 18, 19] A is a tuple $\langle S, s, M, \Omega\rangle$, where S, s, and M are defined as for an ordinary automaton and $\Omega = \{\langle L_n, U_n\rangle\}_{1 \leq n \leq k}$ is a finite sequence of pairs of subsets of S. A *run* of A on a tree f is defined exactly as for an ordinary automaton. The pairs automaton A *accepts* f if and only if there is a run ρ of A on f such that for all infinite forward

infinite paths. A circuit of the form $\langle s, X, s\rangle$ with $s \in X$ indicates that A can cycle endlessly through the set X of states while travelling over a cyclic path, while a series describes the state history of A on an acyclic path. *Lemma 4.9* will show that the minimal plan g_{min} for A on f is good exactly when A accepts f. Note that goodness is preserved under inclusion, i.e., if g and h are two infinite N-ary C_S-trees such that h is good and $\forall x \in T_N.\ g(x) \subseteq h(x)$, then g is good. The lemma below follows immediately.

Lemma 4.4: There is a good plan for A on f if and only if g_{min} is good.

Proof: The minimal plan g_{min} is included in every plan for A on f, so g_{min} must be good if any plan for A on f is good. ∎

The next series of lemmas show that the minimal plan g_{min} contains precisely the circuits for all loops.

Lemma 4.5: For all $x \in T_N$, if $\langle s, X, t\rangle \in g_{min}(x)$, then there is a path π ending in x such that $\rho(\pi) = s$.

Proof: If $\langle s, X, t\rangle \in g_{min}(x)$ then there must be a derivation of this fact by rules (1) - (5) for plans. We proceed by induction on the structure of derivations. For case (1), the required path is the singleton x. If $\langle s\rangle \in g_{min}(x)$ by rule (3), then there is a circuit $\langle t\rangle \in g_{min}(y)$, where $t = M_n(t, f(y))$ and y is the n^{th} neighbor of x. By induction there is a path τ ending in y such that $\rho(\tau) = t$. The required path for $\langle s\rangle$ is $\tau;x$. Similarly for case (2). If $\langle s, X, t\rangle \in g_{min}(x)$ by rule (4), then there is a circuit $\langle s, Y, u\rangle \in g_{min}(x)$ such that $Y \cup \{u\} \subseteq X$. By induction there is a path π ending in x such that $\rho(\pi) = s$. If $\langle s, X, t\rangle \in g_{min}(x)$ by rule (5), then $\langle s\rangle \in g_{min}(x)$. By induction there is a path π ending in x such that $\rho(\pi) = s$. ∎

Lemma 4.6: For all $x \in T_N$ and for all paths π ending in x, $\langle\rho(\pi)\rangle \in g_{min}(x)$.

Proof: We proceed by induction on the length of paths. If π is a singleton, then $\rho(\pi) = s_0$ and $\langle s_0\rangle \in g_{min}(x)$ by rule (1). If $\pi = \tau;n$, where τ ends in Λ, then $\rho(\pi) = L_n(\rho(\tau), f(\Lambda))$ and $\langle\rho(\pi)\rangle \in g_{min}(x)$ by rule (2). Finally, if $\pi = \tau;x$, where τ ends in $y \neq \Lambda$ and x is the n^{th} neighbor of y, then $\rho(\pi) = M_n(\rho(\tau), f(y))$ and $\langle\rho(\pi)\rangle \in g_{min}(x)$ by rule (3). ∎

Lemma 4.7: For all $x \in T_N$, if $\langle s, X, t\rangle \in g_{min}(x)$ then there is a loop π on x such that for all paths of the form $\tau;\pi$, if $\rho(\tau;x) = s$ then $\rho(\tau;x, \tau;\pi) = X$ and $\rho(\tau;\pi) = t$.

Proof: If $\langle s, X, t\rangle \in g_{min}(x)$ then there must be a derivation of this fact by rules (1) - (5) for plans. We proceed by induction on the structure of derivations. For the cases (1) - (3), the required loop is the singleton x. In the case of rule (4), $\langle s, X, t\rangle$ is the join of two circuits $\langle t, Y, u\rangle \in g_{min}(x)$ and $\langle v, Z, w\rangle \in g_{min}(x)$, such that $Y, Z \neq \emptyset$. Inductively, there are loops $x;\sigma;x$ and $x;\tau;x$ for

paths π, $Inf(\rho, \pi) \in F_\Omega$ (i.e., $\notin G_\Omega$).

Pairs automata as defined above will not be used in this thesis. However, by reversing the standard definition of acceptance, we obtain a new type of automaton, the *complemented pairs automaton.* In chapter 5 we will use complemented pairs automata to decide the satisfiability of *delta-PDL* formulae.

Definition: A *complemented pairs automaton* A is a tuple $\langle S, s, M, \Omega\rangle$, where S, s, M, and $\Omega = \{\langle L_n, U_n\rangle\}_{1\leq n\leq k}$ are defined as for a pairs automaton. A *run* of A on a tree f is defined exactly as for a pairs automaton. However, the complemented pairs automaton A *accepts* f if and only if there is a run ρ of A on f such that for all infinite forward paths π, $Inf(\rho, \pi) \in G_\Omega$ (i.e., $\notin F_\Omega$).

The fact that G_Ω is always closed under unions permits a simplified decision procedure for the emptiness problem for complemented pairs automata. The interested reader should compare the procedure below with that of Hossley and Rackoff [8] in order to fully appreciate the similarities and differences. Note that the running time of the procedure below depends both on the number of states and the number of pairs of the automata tested. In chapter 5 we will use complemented pairs automata where k, the number of pairs, is $O(\log m)$, where m is the number of states. The procedure below decides the emptiness problem for such automata in time $O(\exp\ m)$, as opposed to time $O(\exp\ \exp\ m)$ for Hossley's and Rackoff's more general procedure.

Definition: A string $q_1 \cdots q_m \in S^*$ is *good* with respect to a complemented pairs automaton $A = \langle S, s, M, \Omega\rangle$ if and only if $\exists i < m.\ q_i = q_m\ \&\ \{q_{i+1}, \ldots, q_m\} \in G_\Omega$.

Lemma 3.3: The set of strings which are good with respect to a complemented pairs automaton with m states and k pairs is accepted by a deterministic automaton on finite strings of size at worst $O(\exp\ \exp(k+\log m))$.

Proof: It is straightforward to construct a nondeterministic automaton on finite strings, with no more than $O(m \times 2^k)$ states, which accepts exactly the good strings. Applying the Rabin-Scott powerset construction yields the required deterministic automaton. ∎

Definition: A finite N-ary Σ-tree f: $T \rightarrow \Sigma$ is *good* (with respect to A) if there is a run ρ of A on f such that for all $x = n_1 \cdots n_k$ in the frontier of T, $\rho(\Lambda)\rho(n_1)\rho(n_1n_2) \cdots \rho(x)$ is good.

Lemma 3.4: The set of good trees for a complemented pairs automaton with m states and k pairs is accepted by a deterministic automaton on finite N-ary trees with no more than $O(\exp\ \exp(k+\log m))$ states.

Proof: Let B be the deterministic automaton on finite strings guaranteed by the preceding lemma. The desired tree automaton, given a tree f, simulates A on f in order to construct a run of A on f, while simultaneously using B to check every path of this run. ∎

an automaton, so we abbreviate $\langle s, \emptyset, s\rangle$ to $\langle s\rangle$.

Notation: If ρ: $P_N \rightarrow S$ and $\tau, \pi \in P_N$, then $\rho(\tau, \pi) = \{\rho(\mu) \mid \tau < \mu < \pi\}$ and $\rho(\tau \mid \pi) = \langle\rho(\tau), \rho(\tau, \pi), \rho(\pi)\rangle$.

Definition: 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 $x \in T_N$:

(1) $\langle s_0\rangle \in g(x)$

(2) if $\langle s\rangle \in g(\Lambda)$, then $\langle L_n(s, f(\Lambda))\rangle \in g(n)$

(3) if $x \neq \Lambda$ and $\langle s\rangle \in g(x)$ and y is the n^{th} neighbor of x, then $\langle M_n(s, f(x))\rangle \in g(y)$

(4) if $\langle s, X, t\rangle \in g(x)$ and $\langle t, Y, u\rangle \in g(x)$ with $X, Y \neq \emptyset$, 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.

(5) if $\langle s\rangle \in g(x)$, y is the n^{th} neighbor of x, x is the m^{th} neighbor of y, $t = L_n(s, f(\Lambda))$ if $x = \Lambda$ or $M_n(s, f(x))$ otherwise, $v = L_n(u, f(\Lambda))$ if $y = \Lambda$ or $M_n(s, f(x))$ otherwise, and $\langle t, X, u\rangle \in g(y)$, then $\langle s, X \cup \{t,u\}, v\rangle \in g(x)$, in which case the resulting circuit is called the *expansion* of the first one.

The above five conditions 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 circuits which do not correspond to any loop. It will be shown, however, that the least or minimal plan contains precisely the circuits for all loops.

Lemma 4.3: 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 $x \in T_N$, $g_{min}(x) \subseteq g(x)$.

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

Definition: Given a plan g and an infinite forward path $\{x_n\}$, a *series* for g on $\{x_n\}$ is an infinite sequence of circuits $\{\langle s_n, X_n, t_n\rangle\}$ such that for all n, $\langle s_n, X_n, t_n\rangle \in g(x_n)$ and $s_{n+1} = M_m(t_n, f(x_n))$ (or $L_m(t_n, f(\Lambda))$ if $x_n = \Lambda$) if x_{n+1} is the m^{th} neighbor of x_n.

Definition: If ζ is a sequence of circuits, then $Sum(\zeta) = \{s \in S \mid s \in X \cup \{t, u\}$, for infinitely many $\langle t, X, u\rangle$ on $\zeta\}$.

Definition: An infinite N-ary C_S-tree g is *good* if and only if

(1) for all $x \in T_N$, if $\langle s, X, s\rangle \in g(x)$ and $s \in X$, then $X \in G$.

(2) for all infinite forward paths $\{x_n\}$ and series $\zeta = \{\langle s_n, X_n, t_n\rangle\}$ for g on $\{x_n\}$, $Sum(\zeta) \in G$.

The two conditions for goodness correspond to the two forms, cyclic and acyclic, of

Corollary 3.5: The goodness problem for a complemented pairs automaton A on infinite N-ary trees with m states and k pairs, i.e., the problem of deciding whether A has a good tree, is decidable in time at worst $O(N^3 \text{ x } \exp\exp(k+\log m)$.

Proof: The preceding lemma shows that the goodness problem for A is equivalent to the emptiness problem for an automaton B on finite N-ary trees of size at worst $O(\exp\exp(k+\log m))$. It is straightforward to construct an automaton C on finite binary trees, of size at worst $O(N \text{ x } \exp\exp(k+\log m)$, such that C and B have equivalent emptiness problems. Rabin [R69] gives a decision procedure for the emptiness problem for automata on finite binary trees which runs in time $O(n^3)$, where n is the number of states of the automaton tested. ∎

Theorem 3.6: If A accepts a tree, then A has a good tree.

Proof: Suppose A accepts the infinite N-ary Σ-tree f. Let ρ: $T_N \rightarrow S$ be an accepting run of A on f. We claim that for all infinite forward paths π, there is an $x = n_1 \cdots n_k$ on π such that $\rho(\Lambda) \cdots \rho(x)$ is a good string. For if $\pi = \{x_n\}_{n\geq 0}$ is an infinite forward path, then $X = \text{Inf}(\rho, \pi) \in G_\Omega$. For all n, let $q_n = \rho(x_n)$. Let $i = \min\{n \mid \forall m \geq n.\ q_m \in X\}$. Let $j = \min\{n > i \mid q_n = q_i \ \&\ \{q_{i+1}, \ldots, q_n\} = X\}$. Let $x = x_1 \cdots x_j$. Then $\rho(\Lambda) \cdots \rho(x) = q_0 \cdots q_i \cdots q_j$ with $q_i = q_j$ and $\{q_{i+1}, \ldots, q_j\} = X$. So $\rho(\Lambda) \cdots \rho(x)$ is a good string.

Let $T = \{x \in T_N \mid \forall y < x.\ \rho(\Lambda) \cdots \rho(y) \text{ is not good}\}$. We leave it to the reader to establish that T is a finite subtree of T_N and that f restricted to T is a good tree. ∎

Theorem 3.7: If A has a good tree, then A accepts some tree.

Proof: Suppose g is a good tree where g: $T \rightarrow \Sigma$ and T is a finite subtree of T_N. Let σ be a run of A on g which makes g good. Then $\sigma(\Lambda) \cdots \sigma(x)$ is a good string for all $x \in \mathit{front}(T)$, i.e., there exists a $y < x$ such that if $x = yn_1 \cdots n_k$ then $\sigma(x) = \sigma(y)$ and $\{\sigma(y), \sigma(yn_1), \ldots, \sigma(yn_1 \cdots n_{k-1})\} \in G_\Omega$. Define a generating map J: $\mathit{front}(T) \rightarrow \mathit{int}(T)$ by $J(x) = y$. Note that for all $x \in T$, $J^*(x) = x$, and that for $x \notin T$, $J^*(x) < x$. Define f: $T_N \rightarrow \Sigma$ by $f = g \circ J^*$; i.e., f is the finitely generable tree generated by g and J. Similarly, extend σ to T_N by defining $\rho = \sigma \circ J^*$. We leave it to the reader to prove that ρ is a run of A on f.

We claim that ρ is an accepting run of A on f. For suppose $\pi = \{x_n\}_{n\geq 0}$ is an infinite forward path. Let $y_n = J^*(x_n)$ for $n \geq 0$ and let $Y = \mathit{Inf}(J^*, \pi)$. The interior of T is finite, so $\exists i.\ Y = \{y_n \mid n \geq i\}$. Also, by the definition of J and J^*, y_{m+1} is either a successor or an ancestor of y_m for all m. Let $Z = \{y \in \mathit{int}(T) \mid J(zn) \leq y \leq z \text{ for some } z \in Y,\ zn \in \mathit{front}(T),\ J(zn) \in Y\} = \{y \in \mathit{int}(T) \mid y_{m+1} \leq y \leq y_m \text{ for some } m \geq i\}$.

We claim that $Y = Z$. For suppose that $y_k \notin Z$ for some $k \geq i$. We shall show that for all $m \geq i$, if $y_m > y_k$, then $y_{m+1} > y_k$. For suppose $y_m > y_k$ for some $m \geq i$. We know that y_{m+1} is either a successor or an ancestor of y_m. If y_{m+1} is an ancestor of y_m, then $y_{m+1} \leq y_m$ and $y_k < y_m$ imply that either $y_{m+1} \leq y_k$ or $y_{m+1} > y_k$. But $y_{m+1} \leq y_k$ and $y_k < y_m$ imply that $y_k \in Z$,

Definition: A *deterministic two-way automaton on infinite N-ary* Σ*-trees* is a tuple $A = \langle S, s_0, L, M, G \rangle$, where

(1) S is a finite set of states.

(2) $s_0 \in S$ is the initial states.

(3) L: $S \times \Sigma \rightarrow S^N$ is the next state map for the root; for $s \in S$ and $\sigma \in \Sigma$, let $L(s, \sigma) = \langle L_0(s, \sigma), \ldots, L_N(s, \sigma) \rangle$. Informally, if A is in state s on the root, labelled σ, then A will be in state $L_n(s, \sigma)$ on the node n.

(3) M: $S \times \Sigma \rightarrow S^{N+1}$ is the next state map for non-root nodes; 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 n^{th} neighbor of that node.

(4) $G \subseteq$ *Powerset*(S) is a collection of acceptable sets of states. Informally, A accepts a tree if for every infinite path π, G contains the set of states entered infinitely often along π.

Definition: The *run* of a two-way automaton A on an infinite N-ary Σ-tree f is the function ρ: $P_N \rightarrow S$ such that

(1) If π is a singleton, $\rho(\pi) = s_0$.

(2) If π is a path ending in Λ, $\rho(\pi;n) = L_n(\rho(\pi), f(\Lambda))$.

(3) If π is a path ending in $x \neq \Lambda$ and y is the n^{th} neighbor of x, $\rho(\pi;y) = M_n(\rho(\pi), f(x))$.

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

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

Lemma 4.2 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, looping (perhaps trivially) 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: If S is a set of states, then a *circuit* is an element $\langle s, X, t \rangle$ where $s, t \in S$ and $X \subseteq S$. The collection of sets of circuits is denoted by C_S. 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 $\langle s, \emptyset, s \rangle$ represents the instantaneous state of

contradicting the hypothesis, so $y_{m+1} > y_k$. If y_{m+1} is a successor of y_m, then $y_{m+1} > y_k$ also. Hence, for all $m \geq i$, if $y_m > y_k$, then $y_{m+1} > y_k$. Therefore, for all $m \geq i$, if $y_m > y_k$, then for all $l \geq m$, $y_l > y_k$. Therefore, if $\exists y \in Y.\ y > y_k$, then $\forall y \in Y.\ y > y_k$. But y_{k+1} is either a successor or an ancestor of y_k. But if y_{k+1} is a successor of y_k, then $\exists y \in Y.\ y > y_k$, implying $\forall y \in Y.\ y > y_k$, implying $y_k > y_k$, a contradiction. And if y_{k+1} is an ancestor of y_k, then $y_{k+1} \leq y_k$ and $y_k \leq y_k$, implying $y_k \in Z$, contradicting the hypothesis. Therefore, $\forall k \geq i.\ y_k \in Z$, i.e., $Y \subseteq Z$.

Conversely, suppose that $z \in Z$, but $z \notin Y$. Then for some $k \geq i$, $y_{k+1} \leq z \leq y_k$. But $y_{k+1} = z$ or $y_k = z$ contradicts the hypothesis that $z \notin Y$, so $y_{k+1} < z < y_k$. We shall show that for all $m \geq i$, if $y_{m+1} > z$, then $y_m > z$. For suppose $y_{m+1} > z$ for some $m \geq i$. We know that y_{m+1} is either a successor or an ancestor of y_m. If y_{m+1} is a successor of y_m, then $y_{m+1} > z$ implies that $y_m \geq z$. But $y_m = z$ implies that $z \in Y$, contradicting the hypothesis, so $y_m > z$. If y_{m+1} is an ancestor of y_m, then $y_m > z$ also.

Hence, for all $m \geq i$, if $y_{m+1} > z$, then $y_m > z$. Therefore, for all $m \geq i$, if $y_m > z$, then for $i \leq l \leq m$, $y_l > z$. Since $Y = \{y \mid y_m = y \text{ for infinitely many } m\}$, if $\exists y \in Y.\ y > z$, then $\forall y \in Y.\ y > z$. But $y_k, y_{k+1} \in Y$, yet $y_{k+1} < z < y_k$, a contradiction. Therefore, $Z \subseteq Y$. This concludes the proof that $Y = Z$.

$$\begin{aligned}
\text{Hence, } \mathrm{Inf}(\rho, \pi) &= \{\sigma(y) \mid y \in Y\} = \{\sigma(y) \mid y \in Z\} \\
&= \{\sigma(y) \mid \ y_{m+1} \leq y \leq y_m \text{ for some } m \geq i\} \\
&= \{\sigma(y) \mid \ J(zn) \leq y \leq z \text{ for some } z \in Y,\ zn \in \mathit{fron}(T),\ J(zn) \in Y\} \\
&= \bigcup_{z \in Y, zn \in \mathit{fron}(T), J(zn) \in Y} \{\sigma(y) \mid \ J(zn) \leq y \leq z\}.
\end{aligned}$$

By the construction of J, each set $\{\sigma(y) \mid J(zn) \leq y \leq z\} \in G_\Omega$. Since G_Ω is closed under unions, $\mathrm{Inf}(\rho, \pi) \in G_\Omega$. Since $\mathrm{Inf}(\rho, \pi) \in G_\Omega$ for all infinite forward paths π, ρ is an accepting run for A on f. Therefore A accepts f. ∎

Theorem 3.8: The emptiness problem for complemented pairs automata on infinite N-ary trees with m states and k pairs can be decided in time at worst $O(N^3 \times \exp\exp(k+\log m))$.

Proof: The two preceding theorems show the equivalence of the emptiness and goodness problems for complemented pairs automata. The result follows immediately from *Corollary 3.5.* ∎

4 Two-Way Automata on Infinite Trees

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., 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 4.10* shows how to simulate *deterministic* two-way automata by nondeterministic one-way automata; we do not know whether this result can be extended to *nondeterministic* two-way automata. First, however, infinite trees and paths through infinite trees are defined, and some simple results proved about the structure of paths.

Definition: Recall that T_N is an infinite N-ary tree. 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 $0 \leq n \leq N\text{-}1$, the n^{th} neighbor of x is xn; if x is the successor of y, then y is the N^{th} neighbor of x.

Definition: A *finite* (*infinite*) *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 $\pi = \{x_n\}_{1 \leq n \leq L}$ and $\tau = \{x_n\}_{L+1 \leq n \leq M}$ are two finite paths such that x_L and x_{L+1} are neighbors, then the *concatenation* of π and τ is $\pi;\tau = \{x_n\}_{1 \leq n \leq M}$ (defined similarly if τ is an infinite path). The relation $\pi < \tau$ holds if and only if $\tau = \pi;\sigma$ for some nonempty path σ. A *forward path* is a path $\{x_n\}$ such that x_{n+1} is a successor of x_n for all n. A *loop* on x is a finite path $\{x_n\}_{1 \leq n \leq N}$ such that $x_1 = x_N = x$. A *simple loop* is a loop $x;\pi;x$ 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 4.1: If $x;\pi;x$ is a simple loop, then π is a loop.

Proof: Since $x;\pi;x$ 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 4.2: If $\pi = \{x_n\}_{n \geq 0}$ is an infinite acyclic path, then there is an infinite forward path $\{y_n\}_{n \geq 0}$ such that $\pi = \sigma;\tau_0;\ \ldots\ ;\tau_n;\ \ldots$, where each τ_n is a loop on y_n.

Proof: Clearly, 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 $\tau_n = \{x_m\}_{L \leq m \leq M}$, 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 $\{y_n\}_{n \geq 0}$ is an infinite forward path. ∎

CS-TR Scanning Project
Document Control Form

Date : 8/18/95

Report # LCS-TR-263

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 36 (42-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☒ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☒ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

(A) IMAGE MAP: (1-36) PAGES #'ED 2-37
(37-42) SCANCONTROL, COVER, PRINTER'S NOTES, TARGETS (3)
(B) COVER IS BLUE WITH WHITE TYPE.

Scanning Agent Signoff:
Date Received: 8/18/95 Date Scanned: 9/22/95 Date Returned: 9/28/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 9/29/1995

M.I.T. Libraries
Document Services