MIT/LCS/TM-198

THE PROPOSITIONAL DYNAMIC LOGIC OF DETERMINISTIC, WELL-STRUCTURED PROGRAMS

Joseph Y. Halpern
John H. Reif

March 1981

The Propositional Dynamic Logic of Deterministic, Well-Structured Programs

Joseph Y. Halpern,
Mathematics Department,
Harvard University,
Cambridge, Massachusetts 02138.

John H. Reif,
Aiken Computation Laboratory,
Harvard University,
Cambridge, Massachusetts 02138.

March, 1981.

Abstract: We consider a restricted propositional dynamic logic, Strict Deterministic Propositional Dynamic Logic (SDPDL), which is appropriate for reasoning about deterministic well-structured programs. In contrast to PDL, for which the validity problem is known to be complete in deterministic exponential time, the validity problem for SDPDL is shown to be polynomial space complete. We also show that SDPDL is less expressive than PDL. The results rely on structure theorems for models of satisfiable SDPDL formulas, and the proofs give insight into the effects of nondeterminism on intractability and expressiveness in program logics.

KEYWORDS: strict deterministic propositional dynamic logic, propositional dynamic logic, decision procedure, polynomial space complete, expressiveness, well-structured programs.

This research was partially supported by grants from the National Science and Engineering Research Council, NSF Grants MCS80-10707 and MCS79-21024, and ONR Grant N00014-80C0647.

1. Introduction

The major issue in logics of programs is finding a language appropriate for reasoning about programs. We want a language which has sufficient power to enable us to express in a natural way the kinds of properties we would like to prove about programs, such as correctness, termination, and equivalence, and yet is sufficiently tractable to admit an efficient decision procedure. With this in mind, Fischer and Ladner introduced Propositional Dynamic Logic (PDL), a logic based on modal logic. It was shown to have a decision procedure complete in deterministic exponential time and many other desirable properties (cf. [FL, Pr3, KP]).

In analogy to regular languages, PDL makes use of the nondeterministic program constructors * and ∪. However, in programming languages used today, the programs are *deterministic.* Indeed, historically, much of the research in logics of programs has dealt only with deterministic programs (cf. the work of Salwicki and his coworkers in Algorithmic Logic; eg. [Sal] and [Mir]). One way of excluding nondeterminism from PDL is by restricting the use of * and ∪ so that they only occur in contexts which yield deterministic, well-structured programs, equivalent to those built up from the atomic programs using the constructs **while ... do ... od** and **if ... then ... else ... fi**. Strict Deterministic PDL (SDPDL) is the restriction of PDL to formulas where programs are of this sort.

Two natural questions arise: (1) Does the restriction to deterministic programs give us an easier decision procedure, and (2) does it lead to a loss of expressive power; i.e. are there notions which we can express in PDL which are not expressible in SDPDL? The answer to both questions turns out to be yes.

In section 5, we give a procedure for deciding if an SDPDL formula is satisfiable which runs in polynomial space and show that the decision problem is polynomial space hard. In section 6 we show that SDPDL is less expressive than PDL, thus answering an open question of Harel ([Har]). Both the algorithm and expressibility results are based on structure theorems for models of SDPDL formulas which show that if a formula p is satisfiable, it is satisfiable in a tree model with only polynomially many nodes at each depth. In fact, given any tree model for p, we show that we can always find a subtree with only polynomially many nodes at each depth which is also a model of p (cf. Theorems 4.12, 4.19).

These proofs give us insight into the effects of nondeterminism on intractability and expressiveness in program logics. Essentially they show that a deterministic (SDPDL) program cannot examine every node of a full binary tree, while a nondeterministic program can. The first author has shown (in [Hal]) that this situation holds even in the first order case. Thus first

order regular dynamic logic can be shown to be more expressive than its deterministic counterpart, answering another open question of [Har]; (cf. [MW], which studies the quantifier-free case). By contrast, Meyer and Tiuryn have shown (in [MT]) that nondeterminism does not lead to more expressive power in the case of first order dynamic logic with recursively enumerable programs, since a deterministic r.e. program can do a breadth-first search of a tree.

The results similar to our Theorems 5.1 and 5.3 on polynomial space completeness result have been announced independently by Chlebus ([Ch]).

2. Syntax and Semantics

SDPDL is in fact a restriction of Deterministic PDL (DPDL), the logical theory with the same syntax as PDL but its semantics restricted so that in each state an atomic program specifies at most one successor state. DPDL, like PDL, is known to have a decision procedure which is complete in exponential time (cf. [BHP]). We briefly review the syntax and semantics of PDL and DPDL.

2.1 *Syntax*: The alphabet for PDL (as well as DPDL), $\mathcal{A}$, consists of a set Φ_0, whose elements are called atomic formulas, a set Σ_0, whose elements are called atomic programs, and the symbols $\cup$, ;, $*$, ?, $\neg$, $<$, $>$, (,).

The set of programs, Σ, and the set of formulas, Φ, are defined inductively using the following rules:

1. any atomic program in Σ_0 is a program;
2. if a and b are programs, then so are (a;b), ($a \cup b$), and a^*; (we will occasionally omit the parentheses)
3. any atomic formula in Φ_0 is a formula;
4. if p is a formula and a is a program, then $\neg p$ and $\langle a \rangle p$ are formulas;
5. If p is a formula, then p? is a program.

We also use the following abbreviations: $p \wedge q$ for $\langle p? \rangle q$, $p \vee q$ for $\neg(\neg p \wedge \neg q)$, $p \rightarrow q$ for $\neg p \vee q$, $p \equiv q$ for $(p \rightarrow q) \wedge (q \rightarrow p)$, and $[a]p$ for $\neg\langle a \rangle \neg p$.

The *size* of a formula p, written $|p|$, is the length of p regarded as a string over $\mathcal{A}$.

2.2 *Notation*: We will normally reserve P, Q, R, ... for members of Φ_0, and A, B, C, ... for members of Σ_0. The letters p, q, r, ... denote formulas, while the letters a, b, c, ... denote programs.

2.3 *Definition*: A *PDL structure* M is a triple (S, π, ρ) where S is a set whose elements are called *states*, $\pi: \Phi \to \mathcal{P}(S)$ is an assignment of formulas to sets of states, and $\rho: \Sigma \to \mathcal{P}(S \times S)$ is a mapping of programs into binary relations on S which satisfies the following constraints:

1. $\rho(a;b) = \rho(a) \circ \rho(b)$ (composition of relations)
2. $\rho(a \cup b) = \rho(a) \cup \rho(b)$ (union of relations)
3. $\rho(a^*) = (\rho(a))^* = \bigcup_n \rho(a^n)$ (reflexive and transitive closure)
4. $\rho(p?) = \{(s, s) \mid p \in \pi(s)\}$

A *DPDL structure* satisfies in addition:

5. For all $A \in \Sigma_0$, $\rho(A)$ defines a partial function; i.e. if $(s, t), (s, t') \in \rho(A)$, then $t = t'$.

If $p \in \Phi$, then we can view $\pi(p)$ as the set of states in which p is true. And if $a \in \Sigma$, then $\rho(a)$ is the input-output relation of program a, i.e., $(u, v) \in \rho(a)$ means that by starting in state u and running program a we can halt in state v.

The *size* of a structure $M = (S, \pi, \rho)$ is the cardinality of S.

2.4 *Definition*: A *(D)PDL model* is a (D)PDL structure (S, π, ρ) satisfying the following additional constraints on π:

6. $\pi(\neg p) = S - \pi(p)$
7. $\pi(\langle a \rangle p) = \{s \in S \mid \exists t((s, t) \in \rho(a) \text{ and } t \in \pi(p))\}$

2.5 *Remarks*: 1. Given $\pi': \Phi_0 \to \mathcal{P}(S)$, $\rho': \Sigma_0 \to \mathcal{P}(S \times S)$, we can always uniquely extend π' to $\pi: \Phi \to \mathcal{P}(S)$ and ρ' to $\rho: \Sigma \to \mathcal{P}(S \times S)$ so that conditions 1-4, 6, and 7 hold. Moreover, if ρ' satisfies condition 5, then so does ρ. Thus, for a (D)PDL model, π and ρ are completely defined by their actions on the primitive formulas and programs.

2. We will say *t is an a-successor of s* in a structure if $(s, t) \in \rho(a)$. In a DPDL model, each $s \in S$ has at most one A-successor for all $A \in \Sigma_0$. Any (D)PDL model $M = (S, \pi, \rho)$ can be viewed as a directed graph, with the nodes labelled by states in S. We join s to t by an edge labelled A iff $(s, t) \in \rho(A)$. Note that we allow multiple edges between s and t, each labelled with a distinct program of Σ_0. The graph together with π uniquely defines M.

2.6 *Definitions*: Let $M = (S, \pi, \rho)$. Then

1. $M, s \models p$ (p is *true* in $s \in S$) iff $p \in \pi(s)$,
2. $M \models p$ (p is *satisfiable* in M) iff, for some $s \in S$, we have $M, s \models p$,
3. a formula p is *(D)PDL satisfiable* iff for some (D)PDL model M, $M \models p$,
4. $\models_{(D)} p$ (p is *(D)PDL valid*) iff for all (D)PDL models $M = (S, \pi, \rho)$ and all $s \in S$, we have $M, s \models p$.

Note that p is valid iff $\neg p$ is not satisfiable.

2.7 Now we are ready to define SDPDL. We would like to guarantee that the only programs which appear inside boxes and diamonds are deterministic ones. We do this by defining the set of SDPDL programs, Σ_s, to be simply the DPDL programs in which $\cup$ and $*$ appear only in constructs of the form $((p?;a) \cup (\neg p?;b))$ and $((p?;a)^*;\neg p?)$, and we abbreviate these to **if** p **then** a **else** b **fi** and **while** p **do** a **od** respectively. This restricted class of programs clearly corresponds to the well known **while** programs. The SDPDL formulas, Φ_s, are those formulas of Φ involving only programs of Σ_s. The semantics of SDPDL are the same as those of DPDL. For convenience , we will assume that Φ_0 always contains the distinguished formula *true*, with $\pi(true) = S$.

For $p \in \Phi_s$, let $|p|_s$ be the length of p measured as a string over $\Phi_0 \cup \Sigma_0 \cup \{$**if**, **then**, **else**, **fi**, **while**, **do**, **od**, (,), <, >, ;$\}$. We omit the subscript s when it is clear from context.

The following two lemmas describes the basic relationships among the programs and formulas of Σ_s and Φ_s. While the proofs are trivial, the results will be used throughout this paper and the reader should understand them thoroughly before going on.

2.8 *Lemma*: Let $M = (S, \pi, \rho)$ be a DPDL structure.

1. $\rho($**if** p **then** a **else** b **fi**$) = \{(s,t) |\ (s,t) \in \rho(a)$ and $s \in \pi(p)\} \cup \{(s,t) |\ (s,t) \in \rho(b)$ and $s \in \pi(\neg p)\}$.
2. $\rho($**while** p **do** a **od**$) = \{(s,t) |\ \exists s_0 \ldots \exists s_k (s_0 = s, s_k = t,$ for all $i < k$ $((s_i, s_{i+1}) \in \rho(a)$ and $s_i \in \pi(p))$, and $s_k \in \pi(\neg p)\}$.
3. For all $a \in \Sigma_s$, $\rho(a)$ is a partial function.

Proof: Parts 1, and 2 are immediate from the definitions in 2.3, 2.4 and 2.7. Part 3 follows from parts 1 and 2 by induction on the structure of programs. We omit details. ∎

2.9 *Lemma*: The following are valid formulas of SDPDL augmented by $\wedge$:

1. $\models \langle a;b\rangle q \equiv \langle a\rangle\langle b\rangle q$
2. $\models \langle$**if** p **then** a **else** b **fi**$\rangle q \equiv ((p \wedge \langle a\rangle q) \vee (\neg p \wedge \langle b\rangle q))$
3. $\models \langle$**while** p **do** a **od**$\rangle q \equiv ((\neg p \wedge q) \vee (p \wedge \langle a\rangle\langle$**while** p **do** a **od**$\rangle q))$
4. $\models \langle p?\rangle q \equiv p \wedge q$
5. $\models \neg\langle a\rangle p \equiv (\langle a\rangle\neg p \vee \neg\langle a\rangle \mathit{true})$
6. $\models \neg\langle a;b\rangle p \equiv \neg\langle a\rangle\langle b\rangle p$
7. $\models \neg\langle$**if** p **then** a **else** b **fi**$\rangle q \equiv ((p \wedge \neg\langle a\rangle q) \vee (\neg p \wedge \neg\langle b\rangle q))$
8. $\models \neg\langle$**while** p **do** a **od**$\rangle q \equiv ((\neg p \wedge \neg q) \vee (p \wedge \neg\langle a\rangle\langle$**while** p **do** a$\rangle q))$
9. $\models \neg\langle p?\rangle q \equiv \neg p \vee \neg q$
10. $\models \langle a\rangle(p \vee q) \equiv \langle a\rangle p \vee \langle a\rangle q$
11. $\models \langle a\rangle(p \wedge q) \equiv (\langle a\rangle p \wedge \langle a\rangle q)$
12. $\models \neg\neg p \equiv p$
13. If $\models p \equiv q$, then $\models \langle a\rangle p \equiv \langle a\rangle q$.

Proof: Immediate from the definitions and Lemma 2.8.3. Note that the fact that $\rho(a)$ defines a partial function for all $a \in \Sigma_s$ is crucial to the proof of parts 5 and 10. These are not valid formulas of DPDL. ∎

In what follows we always assume, unless explicitly stated otherwise, that all programs and formulas are in Σ_s and Φ_s respectively.

2.10 *Depth of Testing*: Let $DPDL_0$ be all the DPDL formulas with no occurrences of *tests* (programs of the form p?). Let $DPDL_{i+1}$ be those DPDL formulas where, if p? occurs as a program in the formula, $p \in DPDL_i$. Note $DPDL = \cup_i DPDL_i$. Let $SDPDL_i = DPDL_i \cap SDPDL$. Thus, for example, we have

$\langle(\langle(q?;A)^*;\neg q?\rangle p)?;B \cup (\neg\langle(q?;A)^*;\neg q?\rangle p)?;C\rangle \mathit{true} \equiv$
$\langle$**if** $\langle$**while** q **do** A **od od**$\rangle$p **then** B **else** C **fi**$\rangle \mathit{true} \in SDPDL_2$.

Finally, let $DPDL_{pt}$ (*poor test DPDL*) be the variant of DPDL where $\wedge$ is allowed as a primitive operation (i.e. if p, q are formulas, then $p \wedge q$ is a formula rather than being an abbreviation of $\langle p?\rangle q$) and the only tests p? allowed are where p is a Boolean combination of primitive predicates. Again $SDPDL_{pt}$ (*poor test* SDPDL) = $DPDL_{pt} \cap SDPDL$.

2.11 *Remark*: For technical reasons we have not allowed $\wedge$ as a primitive operator in DPDL. Thus our hierarchy is somewhat different from that defined in [B] or [Har]. For example, the formula $P \wedge Q$ is in $DPDL_0$ in the hierarchy a la Harel, but is easily seen not to be in $DPDL_0$ as we have defined it. However, it is in our $DPDL_1$ since it is short for $\langle P?\rangle Q$. The same holds true

at higher levels in the hierarchy. And our $DPDL_{pt}$ is called $DPDL_{0.5}$ in [Har] and [B], but that choice of name seemed inappropriate here since in fact it is *not* less expressive than our $DPDL_1$.

3. FL_s-Closures and Tableaux

3.1 The FL_s-closure of an SDPDL formula p_0, $FL_s(p_0)$, is a slight modification of $FL(p_0)$, (cf. [FL]) the Fischer-Ladner closure of a DPDL formula p_0. It is the least set F such that $p_0 \in F$ and

1. $\langle a\rangle p \in F \rightarrow p \in F$
2. $\langle a;b\rangle q \in F \rightarrow \langle a\rangle\langle b\rangle q \in F$
3. $\langle$**if** p **then** a **else** b **fi**$\rangle q \in F \rightarrow \neg p, p, \langle a\rangle q, \langle b\rangle q \in F$
4. $\langle$**while** p **do** a **od**$\rangle q \in F \rightarrow \neg p, p, \langle a\rangle\langle$**while** p **do** a **od**$\rangle q, q \in F$
5. $\langle p?\rangle q \in F \rightarrow p, q \in F$
6. $\neg p \in F \rightarrow p \in F$
7. $\neg\langle a\rangle p \in F \rightarrow \neg p \in F$ for $a \in \Sigma_0$
8. $\neg\langle a;b\rangle p \in F \rightarrow \neg\langle a\rangle\langle b\rangle p \in F$
9. $\neg\langle$**if** p **then** a **else** b **fi**$\rangle q \in F \rightarrow \neg p, \neg\langle a\rangle q, \neg\langle b\rangle q \in F$
10. $\neg\langle$**while** p **do** a **od**$\rangle q \in F \rightarrow \neg p, \neg q, \neg\langle a\rangle\langle$**while** p **do** a **od**$\rangle q \in F$
11. $\neg\langle p?\rangle q \in F \rightarrow \neg p, \neg q \in F.$

3.2 *Theorem*: If $|p_0|_s = n$, then $|FL_s(p_0)| \leq 2n$

Proof: Let G be the least set containing p_0 satisfying conditions 1-6. Then a slight modification of the proof of Lemma 3.2 in [FL] shows $|G| \leq n$. It is easy to check that $G' = G \cup \neg G$ (where $\neg G = \{\neg p \mid p \in G\}$) satisfies 1-11 and $|G'| \leq 2n$. Since $FL_s(p_0)$ is the least such set, we must have $|FL_s(p_0)| \leq 2n$. ∎

3.3 *Definition*: If $p_0 \in \Phi_s$, let $\Sigma_0(p_0) = \{A \in \Sigma_0 \mid A \text{ appears in } p_0\}$. Let $\Sigma_s(p_0)$ be the least set containing $\Sigma_0(p_0)$ such that if a, b $\in \Sigma_s(p_0)$ and $p \in FL_s(p_0)$ then **if** p **then** a **else** b **fi**, **while** p **do** a **od**, and p? are all in $\Sigma_s(p_0)$.

3.4 An *SDPDL tableau for* p_0 is a DPDL structure $M = (S, \pi, \rho)$ such that $\pi(p_0) \neq \phi$ and for all $s \in S$ the following conditions hold for all formulas of $FL_s(p_0)$:

1. It is not the case that $M, s \models p$ and $M, s \models \neg p$
2. $M, s \models \neg\neg p \rightarrow M, s \models p$
3. $M, s \models \langle a;b\rangle q \rightarrow M, s \models \langle a\rangle\langle b\rangle q$
4. $M, s \models \langle$**if** p **then** a **else** b **fi**$\rangle q \rightarrow$ [($M, s \models p$ and $M, s \models \langle a\rangle q$) or ($M, s \models \neg p$ and $M, s \models \langle b\rangle q$)]

5. $M, s \models \langle$**while** p **do** a **od**$\rangle q \rightarrow [(M, s \models p$ and $M, s \models \langle a \rangle \langle$**while** p **do** a **od**$\rangle q)$ or $M, s \models \neg p$ and $M, s \models q)]$
6. $M, s \models \langle p? \rangle q \rightarrow (M, s \models p$ and $M, s \models q)$
7. $M, s \models \neg\langle a;b \rangle q \rightarrow M, s \models \neg\langle a \rangle\langle b \rangle q$
8. $M, s \models \neg\langle$**if** p **then** a **else** b **fi**$\rangle q \rightarrow [(M, s \models p$ and $M, s \models \neg\langle a \rangle q)$ or $(M, s \models \neg p$ and $M, s \models \neg\langle b \rangle q)]$
9. $M, s \models \neg\langle$**while** p **do** a **od**$\rangle q \rightarrow [(M, s \models \neg p$ and $M, s \models \neg q)$ or $(M, s \models p$ and $M, s \models \neg\langle a \rangle\langle$**while** p **do** a **od**$\rangle q)]$
10. $M, s \models \neg\langle p? \rangle q \rightarrow (M, s \models \neg p$ or $M, s \models \neg q)$
11. $M, s \models \langle A \rangle q \rightarrow \exists t((s,t) \in \rho(A)$ and $M, t \models q)$
12. $M, s \models \neg\langle A \rangle q \rightarrow [\exists t((s,t) \in \rho(A)$ and $M, t \models \neg q)$ or $\forall t((s,t) \notin \rho(A))]$
13. $M, s \models \langle$**while** p **do** a **od**$\rangle q \rightarrow \exists t((s,t) \in \rho($**while** p **do** a **od**$)$ and $M, t \models q)$

The following Lemma is a modification of Lemma 1 in [Pr2] (cf. Lemma 3.7 in [BHP]):

3.5 *Lemma*: A formula p_0 is satisfiable in an SDPDL model iff there is an SDPDL tableau for p_0 such that the two structures have isomorphic graphs.

Proof: It is clear that any SDPDL model satisfying p_0 is automatically a tableau. For the converse we need the following:

3.6 *Sublemma* : Let $M = (S, \pi, \rho)$ be a tableau then

(a) For all programs a, if $M, s \models \langle a \rangle p$ then $\exists t((s,t) \in \rho(a)$ and $M, t \models p)$.

(b) For all programs a, if $M, s \models \neg\langle a \rangle p$ then $\forall t((s,t) \in \rho(a) \rightarrow M, t \models \neg p)$.

Proof: We proceed by induction on the structure of programs. The only case which presents any complications is $M, s \models \neg\langle$**while** p **do** a **od**$\rangle q$. Suppose, in order to obtain a contradiction, that $\exists t((s,t) \in \rho($**while** p **do** a **od**$)$ and $M, t \models q)$. By Lemma 2.8, there exist $s_0, s_1, \ldots, s_k$, with $s_0 = s$, $s_k = t$, such that for all $i < k$, $(M, s_i \models p$ and $(s_i, s_{i+1}) \in \rho(a))$ and $M, s_k \models \neg p$. It is easy to show by induction on i, using condition 9 in the definition of tableau and the main induction assumption, that $M, s_i \models \neg\langle$**while** p **do** a **od**$\rangle q$ for all $i \leq k$. But this leads to a contradiction since $M, s_k \models \neg p$ and $M, s_k \models q$. ∎

Returning to the proof of Lemma 3.4, suppose $M = (S, \pi, \rho)$ is a tableau for p_0. Let $\pi'' = \pi|\Phi_0$, $\rho'' = \rho|\Sigma_0$ and extend them to mappings

π': $\Phi_s \to \mathcal{P}(S)$ and ρ': $\Sigma_s \to \mathcal{P}(S \times S)$ so that the model constraints are satisfied. Let $M' = (S, \pi', \rho')$. Then we can show by simultaneous induction on the size of formulas and programs, using Sublemma 3.6, that for all formulas $p \in FL_s(p_0)$ and all programs $a \in \Sigma_s(p_0)$ we have:

1. $M, s \models p \to M', s \models p$, and $M, s \models \neg p \to M', s' \models \neg p$
2. $(s,t) \in \rho(a) \to (s,t) \in \rho'(a)$
3. $M, s \models \neg\langle a\rangle p \to \forall t((s,t) \in \rho'(a) \to M', t \models \neg p)$

Thus it follows that M' is an SDPDL model for p_0 whose graph is isomorphic to that of M. ∎

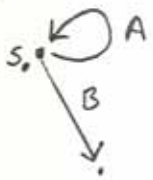
4. Tree Models

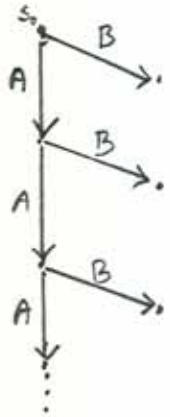
4.1 *Definition*: A *tree model* (for p) is a model (for p) where the graph corresponding to the model is a tree (with no back edges).

4.2 *Lemma*: If a formula p_0 is satisfiable then there is a tree model for p_0.

Proof: We convert a model for p_0 to a (possibly infinite) tree model for p_0 using routine arguments. We make extra copies of states so that no state is the recipient of two distinct program arrows. We omit details here. ∎

4.3 *Example*: Let p_0 be the formula $\neg\langle$**while** $\langle A\rangle\langle B\rangle$*true* **do** A **od**$\rangle$*true*, and let $M, s_0 \models p_0$, where M is the model whose graph is given on the left in the figure below. Then M can be converted to the tree model for p_0 shown on the right:





We will show that satisfiable SDPDL formulas have tree models with special properties. We are aiming for the following:

4.4 *Theorem*: Let p_0 be a satisfiable SDPDL formula such that $|p_0| = n$. Then

1. If $p_0 \in \text{SDPDL}_0$, then p_0 is satisfiable in a tree model with one branch of depth $\leq n$.

2. If $p_0 \in \text{SDPDL}_i$, $i \geq 1$, then p_0 is satisfiable in a tree model with $\leq O(k^i)$ nodes at depth k.

We need some preliminary definitions and lemmas before we can prove this theorem.

4.5 *Definition*: For $a \in \Sigma_s$ we define $\tau(a)$, the set of *a-trajectories* in $M = (S, \pi, \rho)$ by induction on the structure of a (cf. [Pr2, p.328; BHP, Section 4.2])

1. $\tau(A) = \rho(A)$
2. $\tau(a;b) = \tau(a) \circ \tau(b)$
   $= \{(s, \ldots, u, \ldots, t) |\ (s, \ldots, u) \in \tau(a) \text{ and } (u, \ldots, t) \in \tau(b)\}$
3. $\tau(\textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}) = \{(s, \ldots, t) |\ M, s \models p \text{ and } (s, \ldots, t) \in \tau(a)\} \cup \{(s, \ldots, t) |\ M, s \models p \text{ and } (s, \ldots, t) \in \tau(b)\}$
4. $\tau(\textbf{while } p \textbf{ do } a \textbf{ od}) = \bigcup_{i \geq 1} \tau((p?;a)^i; \neg p?)$
5. $\tau(p?) = \{(s) |\ M, s \models p\}$

The *length* of the trajectory $(s_0, \ldots, s_k)$ is k.

Note that $(s,t) \in \rho(a)$ iff there exists a (necessarily unique) a-trajectory $(s_0, \ldots, s_k)$ with $s = s_0$ and $t = s_k$.

4.6 *Definitions*: A *straight line path* is a (possibly infinite) straight line graph with each edge labelled by a primitive program and each node labelled by a finite (possibly empty) subset of Φ_s.

For typographical reasons we use the notation $(n_0, A_0, n_1, A_1, \ldots)$ for a straight line graph, where $n_i \subseteq \Phi_s$ and $A_j \in \Sigma_0$. Let g and h be straight line graphs, with g finite; say $g = (n_0, A_0, n_1, A_1, \ldots, A_{k-1}, n_k)$ and $h = (m_0, B_0, m_1, \ldots)$. Then gh, the concatenation of g and h is the straight line graph

$$(n_0, A_0, n_1, A_1, \ldots, A_{k-1}, (n_k \cup m_0), B_0, m_1, \ldots);$$

i.e. we place the graph of h at the end of g, fusing the first node of h and the last node of g. The *length* of g, written $|g|$, is k. (For an infinite straight line path f we say $|f| = \infty$.) For a node labelled n_j, we define $|n_j| = \Sigma_{p \in n_j} |p|$.

Let L and M be sets of straight line paths. Then in analogy to regular languages, we can define the operations of union, concatenation, Kleene star, and exponentiation on these sets, namely:

1. $L \cup M = \{g \mid g \in L \text{ or } g \in M\}$
2. $L \cdot M = \{gh \mid g \in L, h \in M, \text{ and } g \text{ is finite}\}$
3. $L^* = \epsilon \cup (\cup_{i \geq 1} L^i)$ (where ϵ denotes the empty sequence)
4. $L^\omega = \{g \mid g = g_1 g_2 g_3 \ldots g_k, g_i \in L, \text{ for all } i < k, |g_i| < \infty, \text{ and } |g_k| = \infty\}$
   $\cup \{g \mid g = g_1 g_2 g_3 \ldots, \text{ for all } i, g_i \in L \text{ and } |g_i| < \infty\}$

   (i.e. the elements of L^ω are all of infinite length, and consist of either a finite concatenation of elements of L of which the last has infinite length, or an infinite concatenation of elements of L.)

4.7 *Definition*: For every program and formula in $\Sigma_s \cup \Phi_s$, we can inductively define a corresponding set of straight line paths:

1. for $A \in \Sigma_0$, $L_A = \{(\varnothing, A, \varnothing)\}$
2. $L_{a;b} = L_a \cdot L_b$
3. $L_{p?} = \{(\{p\})\}$
4. $L_{\textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}} = L_{p?} \cdot L_a \cup L_{\neg p?} \cdot L_b$
5. $L_{\textbf{while } p \textbf{ do } a \textbf{ od}} = (L_{p?} \cdot L_a)^* \cdot L_{\neg p?}$
6. for $q = P$, $\neg P$, or $\neg\langle A\rangle \mathit{true}$, where $P \in \Phi_0$ and $A \in \Sigma_0$, $L_q = \{(\{q\})\}$
7. $L_{\neg\neg p} = L_p$
8. $L_{\langle a\rangle p} = L_a \cdot L_p$
9. $L_{\neg\langle a\rangle p} = L_a \cdot L_{\neg p} \cup L_{\neg\langle a\rangle \mathit{true}}$
10. $L_{\neg\langle a;b\rangle \mathit{true}} = L_{\neg\langle a\rangle \mathit{true}} \cup L_a \cdot L_{\neg\langle b\rangle \mathit{true}}$
11. $L_{\neg\langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle \mathit{true}} = L_{p?} \cdot L_{\neg\langle a\rangle \mathit{true}} \cup L_{\neg p?} \cdot L_{\neg\langle b\rangle \mathit{true}}$
12. $L_{\neg\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle \mathit{true}} = (L_{p?} \cdot L_a)^* \cdot L_{p?} \cdot L_{\neg\langle a\rangle \mathit{true}} \cup (L_{p?} \cdot L_a)^\omega$
13. $L_{\neg\langle p?\rangle \mathit{true}} = L_{\neg p?}$

By comparing the inductive definitions given above with those in 4.5 and the tautologies of Lemma 2.9, we can see there are close connections between straight line paths, trajectories, and satisfiable formulas. Intuitively, a formula p is satisfiable in a model M iff there is a unique straight line path in L_p which is a witness to this. The straight line paths in L_p in some sense contain the minimal syntactic information which forces p to satisfied. These notions are formalized in Lemmas 4.9 and 4.10 below.

4.8 *Definition*: The straight line path $g = (n_0, A_0, n_1, \ldots, A_{k-1}, n_k)$ is *consistent* with the trajectory $\alpha = (s_0, s_1, \ldots)$ in $M = (S, \pi, \rho)$ iff $|\alpha| = |g|$ and for all $i < |g|$, $(s_i, s_{i+1}) \in \rho(A_i)$, and for all $i \leq |g|$, if $p \in n_i$, $M, s_i \models p$.

4.9 *Lemma*: Let $M = (S, \pi, \rho)$ be an SDPDL structure.

(a) $(s_0, \ldots, s_k) \in \tau(a)$ iff there is a straight line graph consistent with $(s_0, \ldots, s_k)$ in L_a.

(b) If M is a also an SDPDL model, then $M, s_0 \models p$ iff there is a (possibly infinite) trajectory in M starting with s_0 which is consistent with some straight line path in L_p. This trajectory is unique; i.e. if $M, s_0 \models p$, then there is exactly one trajectory in M starting with s_0 which is consistent with some straight line path in L_p.

Proof: Part (a) follows by a straightforward induction on the structure of programs. We outline the steps needed to prove part (b):

(i) Using part (a), show that if the statement holds for the formula p, then it also holds for $\langle a \rangle p$.

(ii) By induction on the structure of programs, show that the statement holds for $\neg\langle a \rangle \mathit{true}$. As usual, the only case that presents any difficulty is that of $\neg\langle$**while** p **do** a **od**$\rangle \mathit{true}$. Note that $M, s \models \neg\langle a \rangle \mathit{true}$ iff there is no t with $(s,t) \in \rho(a)$ iff there is no a-trajectory starting with s. Moreover, it is easy to show there is no t with $(s,t) \in \rho($**while** p **do** a **od**$)$ iff for all k there is a state t_k with $(s,t_k) \in \rho((p?;a)^k)$ or there is a k and state t_k with $(s,t_k) \in \rho((p?;a)^k;p?)$ and no a-trajectory starting with t_k; i.e. $t_k \models \neg\langle a \rangle \mathit{true}$. Using the induction hypothesis it follows that this is true iff there is a unique trajectory starting with s which is consistent with some straight line path in $(L_{p?} \cdot L_a)^\omega \cup (L_{p?} \cdot L_a)^* \cdot L_{\neg\langle a \rangle \mathit{true}} = L_{\neg\langle}$**while** p **do** a **od**$\rangle \mathit{true}$

(iii) Now we prove (b) by induction on the size of p. It is trivial if p is primitive or the negation of a primitive predicate. If p is of the form $\langle a \rangle q$ the result follows from part (i) by the induction hypothesis. If p is of the form $\neg\neg q$, since $M, s \models \neg\neg q$ iff $M, s \models q$ and $L_{\neg\neg q} = L_q$, the result again follows from the induction hypothesis. Finally, if $p = \neg\langle a \rangle q$, we have $M, s \models \neg\langle a \rangle q$ iff ($M, s \models \langle a \rangle \neg q$ or $M, s \models \neg\langle a \rangle \mathit{true}$) iff there is an a-trajectory starting with s consistent with a straight line path in $L_{\langle a \rangle \neg q} \cup L_{\neg\langle a \rangle \mathit{true}}$ by (i), (ii) and the induction hypothesis. Then we are done since $L_{\langle a \rangle \neg q} \cup L_{\neg\langle a \rangle \mathit{true}} = L_{\neg\langle a \rangle q}$ by 4.7.9. ∎

4.10 *Lemma*: Given a formula p with $g \in L_p$ and $g = (n_0, A_0, n_1, A_1, \ldots)$. Then

(a) if $p \in SDPDL_0$, then $|g| < |p|$, and for $j < |g|$, $n_j = \varnothing$, while $n_{|g|} = \{q\}$, where q is of the form P, $\neg P$, or $\neg\langle A\rangle \mathit{true}$.

(b) if $p \in SDPDL_{i+1}$, then for $j < |g|$, $n_j \subseteq SDPDL_i \cap FL_s(p)$ while if $|g| < \infty$, $n_{|g|} \subseteq SDPDL_i \cap (FL_s(p) \cup \{\neg\langle A\rangle \mathit{true} \mid A \in \Sigma_0\})$. Moreover, if n_j is consistent in the sense that we do not have $\{q, \neg q\} \in n_j$ for some q, then $|n_j| \leq |p|$.

(Intuitively, given a path $g = (n_0, A_0, n_1, \ldots) \in L_p$, the formulas contained in n_j for $j < |g|$ are those forced to be there due to tests contained in p. Thus, if $p \in SDPDL_0$, the n_j will all be empty, while if p is of test depth i+1, all the formulas in n_j will be of depth $\leq$ i.)

Proof: We first prove analogous results for programs by induction on structure, and then prove it for formulas by induction on size. We leave the details to the reader. Note that we need the condition in (b) that n_j is consistent to deal with such programs as **while** q **do** P? **od**. For then $(\{q, \neg q, P\}) \in L_{\textbf{while q do P? od}}$, and if $|q|$ is too large we would have $(|q| + |\neg q| + |P|) > |\textbf{while q do P? od}|$. Condition (b) disallows such cases. ∎

4.11 *Definition*: Given a model $M = (S, \pi, \rho)$ and $S' \subseteq S$, by Remark 2.5 there is a unique model $M' = (S', \pi', \rho')$, such that for all $A \in \Sigma_0$, $\rho(A) \cap (S' \times S') = \rho'(A)$, and for all $P \in \Phi_0$, $\pi(P) \cap S' = \pi'(P)$. M' is said to be the submodel of M *determined* by S', and we write $M' \leq M$. If $M, s \models p$ and $M' \leq M$, then M' is said to *set p at s* if $M', s \models p$ and for all N with $M' \leq N \leq M$, $N, s \models p$. Thus, if M' sets p at s, not only is p satisfied in M', but p is also satisfied in any submodel N of M which contains M' as a submodel.

We are now ready to prove Theorem 4.4. In fact, we will prove the following stronger result, which is the key theorem for the expressiveness result of section 6.

4.12 *Theorem*: Suppose M is a tree model, $M, s_0 \models p_0$, and $|p_0| = n$. Then

(a) If $p_0 \in SDPDL_0$, there is a subtree M' of M which sets p_0 at s_0 and consists of exactly one branch and that branch has length < n.

(b) If $p_0 \in SDPDL_i$, $i \geq 1$ then there is a subtree M' of M which

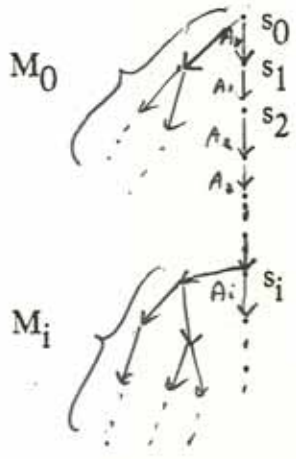
sets p_0 at s_0 and has $\leq n^{2 \cdot k^{i-1}}$ nodes at depth k. Thus p_0 is set by a submodel M′ which has only polynomially many nodes at depth k, where the degree of the polynomial depends on the test depth of p_0.

Note that by Lemma 4.2 any model for p_0 can be converted to a tree model for p_0, so Theorem 4.4 follows immediately from Theorem 4.12.

Proof: Suppose $p_0 \in \mathrm{SDPDL}_i$. By Lemma 4.9(b), there is a trajectory in M starting at s_0 consistent with some $g \in L_{p_0}$, where $g = (n_0, A_0, n_1, A_1, \ldots)$.

(a) Let i = 0. Then, by Lemma 4.10(a), $|g| < n$. Let $(s_0, \ldots, s_k)$ be the trajectory consistent with g. Note we have $k < n$. Let $S' = \{s_0, \ldots, s_k\}$, and let M′ be the subtree of M determined by S′. Then it follows from Lemma 4.10(a) that the trajectory $(s_0, \ldots, s_k)$ *in M′* is still consistent with g. Thus, by Lemma 4.9(b), $M', s_0 \models p_0$. Moreover, the preceding still holds for any N such that $M' \leq N \leq M$. Thus M′ sets p_0 at s_0.

(b) We proceed by induction on i. For the base case, assume i = 1. Let $(s_0, s_1, \ldots)$ be the (possibly infinite) trajectory consistent with g. By Lemma 4.10(b), if $g = (n_0, A_0, n_1, A_1, \ldots)$, each $n_j \subseteq \mathrm{SDPDL}_0$ and $|n_j| \leq n$. From part (a) above, for all $j \leq |g|$ and all $q \in n_j$ there is a set $S_{q,j}$, with $|S_{q,j}| < |q|$, such that $M_{q,j}$ sets q at s_j, where $M_{q,j} \leq M$ is the subtree determined by $S_{q,j}$. Let $S_j = \bigcup_{q \in n_j} S_{q,j}$ and M_j be the subtree of M determined by S_j. Note $|S_j| \leq \Sigma_{q \in n_j} |S_{q,j}| \leq \Sigma_{q \in n_j} |q| = |n_j| \leq n$ (by Lemma 4.10(b)). Then for all $q \in n_j$, $M_j, s_j \models q$ since $M_{q,j} \leq M_j$. Since each $M_{q,j}$ consists of one branch rooted at s_j of length $< |q|$, M_j is a tree with $\leq n$ nodes. Let $S' = \bigcup_{j<|g|} S_j$, and M′ be the subtree of M determined by S′. Thus we get the following picture, with the tree M_i hanging off of s_i:



The trajectory $(s_0, s_1, s_2, \ldots)$ in M′ is still consistent with g, since for every $q \in n_j$, $M', s_j \models q$ (since $M_{q,j} \leq M_j \leq M'$). Thus $M', s_0 \models p_0$. Clearly the preceding still holds for any N with $M' \leq N$, so M′ sets p_0 at s_0.

All that remains is to calculate how many nodes there are of depth k in M′. Let $N_{j,m}$ be the number of nodes in M_j at depth m, and N_m the number of nodes in M′ at depth m. Then

$$(*) \qquad N_k = \sum_{m=0}^{k} N_{k-m,m}$$

However, since each M_j has $\leq n$ nodes, it follows that $N_{j,k} = 0$ if $k \geq n$, and $N_{j,k} \leq n$ for $k < n$. Hence

$$N_k \leq \sum_{m=0}^{n-1} N_{k-m,m} \text{ (where we take } N_{i,j} = 0 \text{ if } i < 0)$$
$$\leq n^2$$

Thus, at any depth in the tree M′ there are less than n^2 nodes. (This result can be improved. It can be shown that at any depth in M′ there are less than n nodes, but we do not need this fact here.)

Assume as our inductive hypothesis that if $q \in \mathrm{SDPDL}_i$ $(i \geq 1)$ and M is a tree model with M, $s \models q$ then there is a subtree M′ of M which sets q at s and has $\leq |q|^2 \cdot k^{i-1}$ nodes at depth k. Now suppose $p \in \mathrm{SDPDL}_{i+1}$, $|p| = n$, and M, $s_0 \models p$ as in the hypothesis of the theorem. We repeat the argument given above. This time, each $n_j \subseteq \mathrm{SDPDL}_i$, so by the induction hypothesis, $M_{q,j}$ has $\leq |q|^2 \cdot k^{i-1}$ nodes at depth k. Thus the number of nodes of M_j at depth k is

$$\leq \Sigma_{q \in n_j} |q|^2 \cdot k^{i-1}$$
$$\leq (\Sigma_{q \in n_j} |q|^2) \cdot k^{i-1}$$
$$\leq (\Sigma_{q \in n_j} |q|)^2 \cdot k^{i-1}$$
$$\leq n^2 \cdot k^{i-1} \qquad \text{(using Lemma 4.10(b))}$$

i.e. $N_{j,k} \leq n^2 \cdot k^{i-1}$.

Since equation (*) above still holds, we have

$$N_k = \sum_{m=0}^{k} N_{k-m,m} \leq \sum_{m=0}^{k} n^2 \cdot m^{i-1} \leq \sum_{m=1}^{k} n^2 \cdot k^{i-1} = n^2 \cdot k^i$$

Thus M′ sets p at s_0 and has $\leq n^2 \cdot k^i$ nodes at depth k. (Again we note that the n^2 can be improved to n.) ∎

For formulas in $SDPDL_{pt}$ we can do even better: we can always find a finite tree model. We are aiming for the following:

4.13 *Theorem*: Let p_0 be a satisfiable $SDPDL_{pt}$ formula with $|p_0| = n$. Then p_0 is satisfiable in a tree model with $< n$ branches which has depth $\leq 2^n$.

We first need to prove some more detailed technical properties of straight line paths.

4.14 *Lemma*: Given a formula p with $g \in L_p$ and $g = (n_0, A_0, n_1, A_1, \ldots)$.

(a) If p is of the form $\langle a_1\rangle\ldots\langle a_m\rangle q$ and $g = f_1\ldots f_m h$, with $f_i \in L_{a_i}$ for $i \leq m$, $h \in L_q$ and $\Sigma_{i\leq m} |f_i| = N \geq 1$, then for all $i < N$ there is a formula p_i of the form $\langle A_i\rangle\langle b_1\rangle\ldots\langle b_k\rangle q \in FL_s(p)$, such that $(\varnothing, A_i, n_{i+1}, A_{i+1}, \ldots) \in L_{p_i}$ and $g \in (n_0, A_0, \ldots, A_{i-1}, n_i)\cdot L_{p_i} \subseteq L_p$. In fact we have $(n_0, A_0, \ldots, A_{i-1}, n_i)\cdot L_{A_i}\cdot L_{b_i}\cdot\ldots\cdot L_{b_k} \subseteq L_{a_i}\cdot\ldots\cdot L_{a_m}$.

(Intuitively, for any formula prefaced by $\langle\rangle$'s and any $g \in L_p$, and for any node n_i occurring on g, we can find another formula p_i in $FL_s(p)$ such that continuing from node n_i with any path in L_{p_i} leaves you with a path in L_p. Moreover, p_i begins with $\langle A_i\rangle$ so it marks the next step to take along g.)

(b) There exists a formula p′ of the form q or $\langle a\rangle q$, where q is of the form P, $\neg P$, or $\neg\langle b\rangle$*true* such that $\models p' \rightarrow p$, $L_{p'} \subseteq L_p$, and $g \in L_{p'}$. Moreover, if $|g| = \infty$, then q is of the form $\neg\langle b\rangle$*true*.

(c) If p is of the form $\neg\langle a\rangle$*true* and $|g| = \infty$, then there exists a formula p′ of the form $\neg\langle$**while** q **do** b **od**$\rangle$*true* or $\neg\langle c\rangle\langle$**while** q **do** b **od**$\rangle$*true* such that $\models p' \rightarrow p$, $L_{p'} \subseteq L_p$ and $g \in L_{p'}$.

(d) If p is of the form $\neg\langle a\rangle$*true* and $|g| \geq 1$, then for all $i < |g|$, $g' = (n_0, A_0, n_1, A_1, \ldots, A_{i-1}, n_i') \in L_p$, where $n_i' = n_i \cup \{\neg\langle A_i\rangle$*true*$\}$.

Proof: The proof of (a) proceeds by induction on i, m, and the structure of a_1. If $i = 0$ and $m = 1$, there is clearly no problem for a_1 primitive, or of the form **if** r **then** b **else** c **fi**. Nor is there a problem if a_1 is of the form r?,

since then $|f_1| = 0$. If $a_1 = b;c$ there exist $g_1 \in L_b$, $g_2 \in L_c$ with $f_1 = g_1g_2$. If $|g_1| > 0$, we are done by the induction hypothesis applied to $\langle b\rangle\langle c\rangle q \in FL_s(p)$. If $|g_1| = 0$, suppose $g_1 = (n_0')$. Then $|g_2| > 0$, $g_2h = (n_0'', A_0, n_1, \ldots) \in L_{\langle c\rangle q}$, with $(n_0' \cup n_0'') = n_0$. Now we are done by applying the induction hypothesis to $\langle c\rangle q \in FL_s(p)$.

Finally, if a is of the form **while** r **do** b **od**, then since $|f_1| > 0$, by Definition 4.7.5, $f_1 = g_1g_2g_3$, where $|g_1| = 0$, $g_2 \in L_b$, $|g_2| > 0$, and $g_3 \in L_{\textbf{while r do b od}}$. Again we can suppose $g_1 = (n_0')$, and $g_2g_3h = (n_0'', A_0, n_1, \ldots) \in L_{\langle b\rangle\langle \textbf{while r do b od}\rangle q}$. Since $|g_2| > 0$, we can apply the induction hypothesis to $\langle b\rangle\langle \textbf{while r do b od}\rangle q \in FL_s(p)$ and we are done.

There is no problem extending the result by induction to $n > 1$. For if we have shown it for i, we have a formula p_i of the form $\langle A_i\rangle\langle b_1\rangle\ldots\langle b_k\rangle q$ such that $(n_0, A_0, \ldots, A_{i-1}, n_i)\cdot L_{p_i} \subseteq L_p$, thus $(n_0, A_0, \ldots, n_i, A_i, \varnothing)\cdot L_{\langle b_i\rangle\ldots\langle b_k\rangle q} \subseteq L_p$ and $(n_{i+1}, A_{i+1}, \ldots) \in L_{\langle b_1\rangle\ldots\langle b_k\rangle q}$. Thus we can just repeat the argument given above for the case i = 0 for $p' = \langle b_1\rangle\ldots\langle b_k\rangle q \in FL_s(p)$ to get p_{i+1}.

The first half of (b) follows by a straightforward induction on the structure of programs, using 4.7.2, 4.7.7 - 4.7.9 and 2.9.1, 2.9.5, 2.9.12, and 2.9.13. The second half follows immediately from the first.

Part (c) is proved by a similar induction on the structure of programs, this time using 4.7.9 - 4.7.11 and 2.9.1, 2.9.4 - 2.9.7.

To prove part (d), we first need to show the following:

(*) if $|f| < \infty$, then $f \in L_{\neg\langle a\rangle \mathit{true}}$ iff there is an $h \in L_a$ with $|h| > |f|$ such that $h = (m_0, B_0, m_1, B_1, \ldots, B_k, m_k)$, and for some $i < k$ we have $f = (m_0, B_0, \ldots, B_{i-1}, m_i')$ where $m_i' = m_i \cup \{\neg\langle B_i\rangle \mathit{true}\}$.

Intuitively, (*) says that the finite length elements of $L_{\neg\langle a\rangle \mathit{true}}$ are essentially prefixes of elements in L_a. (Note that elements in L_a are always of finite length). As usual, (*) is proved by a straightforward induction on the structure of programs.

Returning to the proof of (d), if $|g| < \infty$ the result follows immediately from (*). If $|g| = \infty$, then by part (c), there exists a formula p' such that $g \in L_{p'} \subseteq L_p$, $\models p' \rightarrow p$, and p' is of the form $\neg(\langle c\rangle)\langle \textbf{while q do b od}\rangle \mathit{true}$.

Hence $g \in (L_c \cdot)(L_{q?} \cdot L_b)^\omega$. Given i, there exists $j > i$ and $k \geq 0$ such that $(n_0, A_0, \ldots, A_{i-1}, n_j) \in (L_c \cdot)(L_{q?} \cdot L_b)^k$. It then follows that $(n_0, A_0, n_1, \ldots, A_{j-1}, n_j'') \in L_c \cdot (L_{q?} \cdot L_b)^k \cdot L_{\neg q?} \subseteq L_a \cdot L_{\textbf{while } q \textbf{ do } b \textbf{ od}}$, where $n_j'' = n_j \cup \{\neg q\}$. Thus by (*),

$$(n_0, A_0, \ldots, A_{i-1}, n_i') \in L_{\neg(\langle c\rangle)\langle \textbf{while } q \textbf{ do } b \textbf{ od}\rangle \mathit{true}} \subseteq L_p. \quad \blacksquare$$

4.15 *Remark*: The formulas p_i of Lemma 4.14(a) correspond to the derivatives of [BHP, Definition 4.4]. It should also be noted that with a little more work, a more general form of 4.14(a) can be proved. Namely, we can show that for any formula p, if $g \in L_p$ and $g = (n_0, A_0, n_1, A_1, \ldots)$ then for all $i < |g|$ there is a formula p_i of the form $\langle A_i\rangle q$ or $\neg\langle A_i\rangle q$ such that $g \in (n_0, A_0, \ldots, A_{i-1}, n_i) \cdot L_{p_i} \subseteq L_p$ and $(\varnothing, A_i, n_{i+1}, A_{i+1}, \ldots) \in L_{p_i}$.

4.16 *Definition*: A formula of $SDPDL_{pt}$ is said to be *elementary* if it is of the form $\langle a_1\rangle \ldots \langle a_i\rangle q$, $i \geq 0$, where q is of the form P or $\neg\langle b_1\rangle \ldots \langle b_j\rangle P$, $j \geq 0$. The notion of straight line path is well-defined for elementary formulas; Definition 4.7 carries over directly. (Unfortunately, for general $SDPDL_{pt}$ formulas it does not carry over so well; conjunction causes problems. It is not clear how to define a straight line path corresponding to $\langle A\rangle P \wedge \langle B\rangle Q$). Lemmas 4.9 and 4.14 also carry over to elementary formulas with no change. In addition we have:

4.17 *Lemma*: Let $p \in SDPDL_{pt}$, with $|p| = n$.

(a) p is equivalent to a formula in "disjunctive normal form", i.e. one which is the disjunction of conjunctions of elementary formulas. Moreover, each disjunct is the conjunction of at most n elementary formulas, each of size $\leq n$.

(b) If p is elementary, and $g \in L_p$ with $g = (n_0, A_0, n_1, A_1, \ldots)$, then for all $j < |g|$, $n_j \subseteq \Phi_0$, while if $|g| < \infty$, $n_{|g|} \subseteq \Phi_0 \cup \{\neg\langle A\rangle \mathit{true} \mid A \in \Sigma_0\}$.

Note that part (b) is analogous to Lemma 4.10. Since the only tests in $SDPDL_{pt}$ formulas are primitive, the only formulas that appear in n_j for $j < |g|$ are primitive.

Proof: Part (a) follows easily by induction on the size of formulas, using 2.9.5, 2.9.10 - 2.9.13. Part (b) is similar to Lemma 4.10, and the proof is also left to the reader. ∎

Proof of Theorem 4.13: Let M, $s_0 \models p_0$. By Lemma 4.17(a), p_0 is equivalent to a formula p_1 which is the disjunction of conjunctions of elementary formulas. Thus M, $s_0 \models q_1 \wedge \ldots \wedge q_k$, where $(q_1 \wedge \ldots \wedge q_k)$ is one of the disjuncts of p_1. Note $\models (q_1 \wedge \ldots \wedge q_k) \rightarrow p_0$, and by Lemma 4.17(a), $k \leq n$ and for all $j \leq k$, $|q_j| \leq n$. By Lemma 4.9(b) (which, by the remark above, applies to elementary formulas), for each q_j there is a trajectory in M starting with s_0 consistent with some $g_j \in L_{q_j}$. We could now show, just as in Theorem 4.12, that the subtree of M determined by the union of the states in these trajectories is also a model for $q_1 \wedge \ldots \wedge q_k$, and hence for p_0. This model has $\leq n$ branches, but they are not necessarily be finite. To show that we can actually construct a finite model we use Lemmas 4.14(b) and (d). Lemma 4.14(b) says that if there is an infinite branch, it must essentially be due to a formula of the form $\neg\langle a\rangle \mathit{true}$; Lemma 4.14(d) says that we can truncate this branch and still get a model that satisfies this formula. In more detail we proceed as follows:

Note that by Lemma 4.14(b), there is a formula r_j of the form q or $\langle a\rangle q$, where q is of the form P, $\neg P$ or $\neg\langle b\rangle \mathit{true}$ such that $\models r_j \rightarrow q_j$, $L_{r_j} \subseteq L_{q_j}$, and $g_j \in L_{r_j}$. If $|g_j| = \infty$, then q is of the form $\neg\langle b\rangle \mathit{true}$. So $g_j = f_j h_j$, where $h_j \in L_{\neg\langle b\rangle \mathit{true}}$ and $f_j \in L_a$ or $f_j = (\varnothing)$; in either case $|f_j| < \infty$. We now want to truncate the infinite h_j's without affecting the finite paths. To this end, let $N = \max(\{|g_j| \mid |g_j| < \infty\} \cup \{|f_j| \mid |g_j| = \infty\})$. Thus, if $|g_j| = \infty$ and $g_j = (n_0, A_0, n_1, A_1, \ldots)$, then by 4.14(d) $g_j' = (n_0, A_0, n_1, \ldots, A_{n-1}, n_N')$ $\in L_{r_j}$ where $n_N' = n_N \cup \{\neg\langle A_N\rangle \mathit{true}\}$. Truncate the infinite trajectories consistent with those g_j such that $|g_j| = \infty$ to finite trajectories of length N. Let S' be the union of the states in the truncated trajectories, and let M' be the subtree of M determined by S'. Since, by Lemma 4.17(b), the only formulas mentioned at the nodes of g_j or $g_{j'}$ are in Φ_0 or of the form $\neg\langle A\rangle \mathit{true}$, it is easy to see that for all $j \leq k$ there is a trajectory in M' beginning with s_0 which is consistent with g_j (or $g_{j'}$ if g_j is infinite). Thus M', $s_0 \models r_1 \wedge \ldots \wedge r_k$, so we also get M', $s_0 \models p_0$.

M' is a finite tree model with $\leq n$ branches, but we are still not done since we require a model whose branches have length $\leq 2^n$. We will get this essentially by identifying states in M' which agree on all subformulas of p_0.

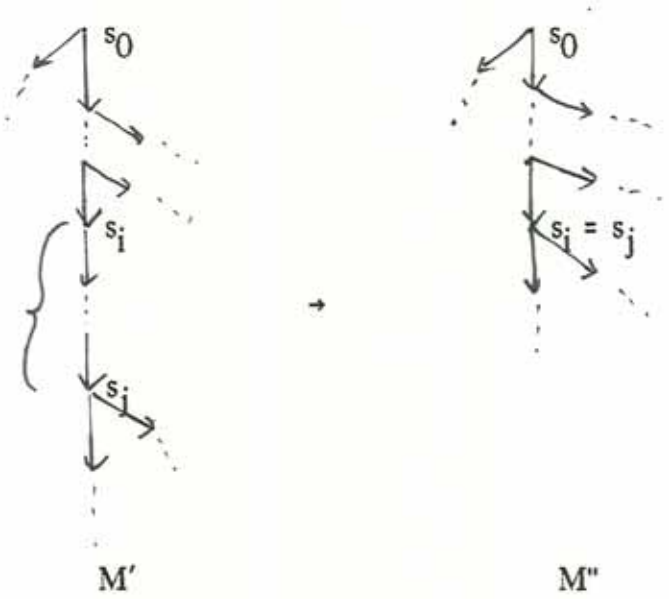
First note that by systematically replacing subformulas of p_0 of the form $(p \wedge q)$ by $\langle p?\rangle q$, we can find a formula p_2 of SDPDL such that $|p_2| = n$ and $\models p_0 \equiv p_2$. Now we define an equivalence relation $\equiv$ between states in

M′ via $s \equiv s'$ iff for all $q \in FL_s(p_2)$, M′, $s \models q$ iff M′, $s' \models q$. There are $\leq 2n$ formulas in $FL_s(p_2)$, but at most 2^n distinct equivalence classes. To see this, take the set G as in Theorem 3.2. Since $FL_s(p_2) \subseteq G \cup \neg G$, it follows that $s \equiv s'$ iff for all $q \in G$, M′, $s \models q$ iff M′, $s' \models q$. And since $|G| \leq n$, the result follows. Now suppose $(s_0, \ldots, s_m)$ is a branch of length $> 2^n$. Then there must be two states s_i, s_j on the branch with s_j a descendant of s_i and $s_i \equiv s_j$. Let M" = (S", π", ρ") be the structure whose graph is obtained from that of M′ by identifying s_i and s_j and eliminating all the descendants of s_i which are not descendants of s_j, where π" is defined so that

(*) for $q \in FL_s(p)$, M", $s \models q$ iff M′, $s \models q$.

We get the following picture:

This section is eliminated



M′ M"

We claim that M" is a tableau for p_2. All the tableau conditions except 3.4.13 follow immediately from (*). Now suppose M", $s \models \langle$**while** p **do** a **od**$\rangle$q. In M′ there is a trajectory $(t_0, \ldots, t_k) \in \tau$(**while** p **do** a **od**) such that $t_0 = s$ and M′, $t_k \models q$. If this trajectory does not pass through s_i, then it also exists in M" and we are done, since clearly $(t_0, t_k) \in \rho$"(**while** p **do** a **od**). Otherwise, suppose $g \in L_q$ is consistent with a trajectory in M′ beginning with s. This trajectory starts with $(t_0, \ldots, t_k)$, and hence by assumption it passes through s_i. Suppose s_i corresponds to node n_m in g. Then by Lemma 4.14(a) there is a formula $q_m \in FL_s(\langle$**while** p **do** a **od**$\rangle$q) $\subseteq FL_s(p_2)$ such that q_m is of the

form $\langle A_m\rangle\langle b_1\rangle\ldots\langle b_h\rangle q$, where $(n_0, A_0, \ldots, A_{m-1}, n_m)\cdot L_{A_m}\cdot L_{b_1}\cdot\ldots\cdot L_{b_h} \subseteq L_{\textbf{while } p \textbf{ do } a \textbf{ od}}$, and $(\varnothing, A_m, n_{m+1}, \ldots) \in L_{q_m}$. Again, in words, q_m has the property that continuing from $(t_0, \ldots, t_i)$ with *any* path in L_{q_m} gives us a path in L_q starting with t_0.

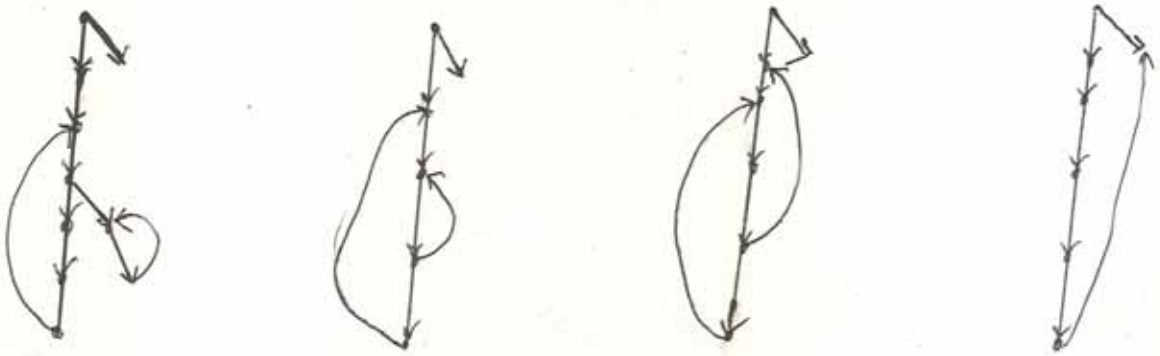
Thus M', $s_i \models q_m$, and since $s_i \equiv s_j$, we also have M', $s_j \models q_m$. Since M' and M'' are isomorphic below s_j, there is a trajectory in M' and M'' consistent with some $g' \in L_{A_m}\cdot L_{b_1}\cdot\ldots\cdot L_{b_m}$ such that if t is the last state in the trajectory, M'', $t \models q$. But $(n_0, A_0, \ldots, A_{m-1}, n_m)\cdot g' \in L_{\textbf{while } p \textbf{ do } a \textbf{ od}}$, thus there is a trajectory in M'' starting with s and ending with t consistent with some path in $L_{\textbf{while } p \textbf{ do } a \textbf{ od}}$. By Lemma 4.9(a), it follows that $(s,t) \in \rho''(\textbf{while } p \textbf{ do } a \textbf{ od})$, giving us our result.

By Lemma 3.5 we can convert M'' to a model with a graph isomorphic to that of M''. (In fact, using the stronger form of Lemma 4.14(a) mentioned in Remark 4.15, it can be shown that M'' itself is actually a model). And by repeating the procedure described above a finite number of times, we can find a model for p_2, and hence p_0, with n branches all of length $\leq 2^n$. ∎

We cannot in general hope to find finite tree models for all satisfiable formulas of SDPDL. Consider, for example, the $SDPDL_1$ formula $\neg\langle\textbf{while } \langle A\rangle \textit{true} \textbf{ do } A \textbf{ od}\rangle \textit{true}$. It is easily seen to be satisfiable in the following tree model, where $s_i \models \langle A\rangle \textit{true}$ for all i:

It is also easy to see that it is not satisfiable in any finite tree model exactly because each $s_i \models \langle A\rangle \textit{true}$. However, we can get a finite representation of the infinite tree by allowing backedges. This motivates the following

4.18 *Definition*: A *treelike model* (for p) is a model (for p) whose graph is a tree with backedges only to ancestors and no nesting or crossing of backedges; i.e. each backedge induces a unique cycle. More precisely, if i, j, k, and l are nodes on the graph of a treelike model and there is a backedge from j to i and from k to l, then (i) i < j (where "<" denotes ancestor of) and (ii) if (i,j) ≠ (k,l) then we do *not* have i, k < j, l. Thus the graph on the left is the graph of a treelike model, while the other three are not.



4.19 *Theorem*: If p_0 is a satisfiable formula of $SDPDL_i$, then p_0 is satisfiable in a treelike model of depth $\leq in2^{n+1} + n$.

Before we can prove Theorem 4.19 we need one more definition, which will also be crucial to the algorithm presented in section 5.

4.20 *Definition*: Let $F = \{q_1, ..., q_k\}$ be a finite set of formulas. Then the *weight* of F, written $\|F\|$, is a pair (i,j), where i is the depth of the formula in F of greatest depth, and j is the number of formulas in F of depth i. We put an ordering on weights via

$$(i,j) < (i',j') \quad \text{iff} \quad i < i' \text{ or } (i = i' \text{ and } j < j')$$

Proof of Theorem 4.19: Let $p_0 \in SDPDL_i$. It is easy to see from Definition 3.1 that $FL_s(p_0) \subseteq SDPDL_i$. We show by induction on the weight of $F \subseteq FL_s(p_0)$ that if $F = \{q_1, ..., q_m\}$ and $\|F\| = (j,k)$, then if $q_1 \wedge ... \wedge q_m$ is satisfiable, it is satisfiable at the root of a treelike model of depth $\leq (jn2^{n+1} + (k-1)2^n + n)$. Since p_0 is satisfiable and $\|\{p_0\}\| = (i,1)$, this will give us the desired result.

In the case $\|F\| = (0,k)$, we have $F \subseteq FL_s(p_0) \cap SDPDL_0$. It is easy to check that every element in $FL_s(p_0) \cap SDPDL_0$ has size $\leq n$, so it follows immediately from Theorem 4.12(a) that there is a tree model of depth $\leq n$ satisfying the conjunction of the formulas in F.

In the general case, suppose $\|F^0\| = (j,k)$, where $F^0 = \{q_1, ..., q_m\}$, and

M is a tree model such that M, $s_0 \models q_1 \wedge \ldots \wedge q_m$. Using M as an oracle, we will construct a treelike tableau M' of the required depth such that all the formulas in s_0 are true at the root. The first step is to "massage" all the formulas of F^0 to the "right" form, either $\langle A\rangle r$ or $\neg\langle A\rangle r$. We do this in stages: F^0_0, F^0_1, F^0_2, ... Let $F^0_0 = F^0$ and let

$$F^0_{x+1} = \{q_\alpha, q_\beta \mid q \in F^0_x, q_\alpha, q_\beta \notin F^0_y, \text{for } y \leq x\} \cup$$
$$\{q \mid q \in F^0_x \text{ and } q \text{ is of the form } \langle A\rangle r \text{ or } \neg\langle A\rangle r\},$$

where q_α, q_β are defined by the following rules. (Note that M is needed in rules 4, 5, 8, 9, and 10 below. Roughly speaking, q_α corresponds to the tests in a program, while q_β corresponds to the consequence of the test.)

1. If q is of the form $\langle A\rangle r$, $\neg\langle A\rangle r$, P, or $\neg P$, $q_\alpha = q_\beta = q$.
2. If q is of the form $\neg\neg r$, $q_\alpha = q_\beta = r$.
3. If q if of the form $\langle a;b\rangle r$, $q_\alpha = q_\beta = \langle a\rangle\langle b\rangle r$.
4. If q is of the form $\langle$**if** p **then** a **else** b **fi**$\rangle r$ and M, $s \models p$, then $q_\alpha = p$, $q_\beta = \langle a\rangle r$. If M, $s \models \neg p$, then $q_\alpha = \neg p$, $q_\beta = \langle b\rangle r$.
5. If q is of the form $\langle$**while** p **do** a **od**$\rangle r$, and M, $s \models p$, then $q_\alpha = p$, $q_\beta = \langle a\rangle\langle$**while** p **do** a **od**$\rangle r$. If M, $s \models \neg p$, then $q_\alpha = \neg p$, $q_\beta = r$.
6. If q is of the form $\langle p?\rangle r$, then $q_\alpha = p$, $q_\beta = r$.
7. If q is of the form $\neg\langle a;b\rangle r$, then $q_\alpha = q_\beta = \neg\langle a\rangle\langle b\rangle r$.
8. If q is of the form $\neg\langle$**if** p **then** a **else** b **fi**$\rangle r$ and M, $s \models p$, then $q_\alpha = p$ and $q_\beta = \neg\langle a\rangle r$. If M, $s \models \neg p$, then $q_\alpha = \neg p$, $q_\beta = \neg\langle b\rangle r$.
9. If q is of the form $\neg\langle$**while** p **do** a **od**$\rangle r$ and M, $s \models p$, then $q_\alpha = p$, $q_\beta = \neg\langle a\rangle\langle$**while** p **do** a **od**$\rangle r$. If M, $s \models \neg p$, then $q_\alpha = \neg p$, $q_\beta = \neg r$.
10. If $q = \neg\langle p?\rangle r$ and M, $s \models p$, then $q_\alpha = q_\beta = \neg r$. If M, $s \models \neg p$, then $q_\alpha = q_\beta = \neg p$.

We require a technical sublemma to verify that we do indeed massage all the formulas to the intended form. Part 7 of the sublemma will also be useful at later stages of the construction.

4.21 *Sublemma:*

1. For all x, $F^0_x \subseteq FL_s(p_0)$ (i.e. the massaging process keeps us in $FL_s(p_0)$).
2. For all x and $q \in F^0_x$, M, $s_0 \models q$.
3. $\|F^0_{x+1}\| \leq \|F^0_x\|$.

4. For some $x_0 \leq n$, all the formulas in $F^0_{x_0}$ are of the intended form $\langle A\rangle r$ or $\neg\langle A\rangle r$.

5. If $q_\beta = q_\alpha$, then $L_{q_\beta} \subseteq L_q$; if $q_\beta \neq q_\alpha$, $\{(q_\alpha)\}\cdot L_{q_\beta} \subseteq L_q$. (Roughly, if there is no test, then $\models q_\beta \rightarrow q_\alpha$ so $L_{q_\beta} \subseteq L_{q_\alpha}$. If there is a test, then any path in L_{q_β} where the test holds at the first node is a path in L_{q_α}.)

6. Define $\Rightarrow$ ("leads to") to be the least reflexive and transitive relation on $\cup_x F^0_x$ such that $q \Rightarrow q_\beta$ for all $q \in \cup_x F^0_x$. Then if $q \Rightarrow r$, there is a $G \subseteq \cup_x F^0_x$ such that $\{(G)\}\cdot L_r \subseteq L_q$. (G is a collection of tests. If they all hold, and we append to them a path in L_r, we get a path in L_q).)

7. If $p \in \cup_x F^0_x$ of the form $\langle a\rangle r$, then either $p \Rightarrow r$ or there exists r' of the form $\langle A\rangle\langle b_1\rangle\ldots\langle b_k\rangle r$ such that $p \Rightarrow r'$.

Proof: Parts 1, 2, and 5 are immediate from rules 1-10 above. For 3, note that q_β has depth less than or equal to that of q, and if $q_\alpha \neq q_\beta$, then q_α has depth less than that of q. The result follows immediately. To prove 4, note that if some formula of F^0_x is not of the form $\langle A\rangle r$ or $\neg\langle A\rangle r$, then it must be the case that the sets F^0_0 and $(F^0_y - F^0_{y-1})$, $0 < y \leq x$, are non-empty and pairwise disjoint. Since $\cup_x F^0_x$ is a consistent subset of $FL_s(p_0)$ (i.e. does not contain a formula and its negation), by the comments made in the proof of Theorem 4.13 we have $|\cup_x F^0_x| \leq n$. Thus x_0 must be less than n, and in particular all the formulas in $F^0_{x_0}$ have the desired form. Part 6 follows from the observation that $\{(q,r) \mid q, r \in \cup_x F^0_x$ and for some $G \subseteq \cup_x F^0_x$, $\{(G)\}\cdot L_r \subseteq L_q\}$ is a reflexive transitive relation, which, from part 5, contains all pairs (q,q_β).

There are two possibilities in part 7. If the unique a-trajectory starting at s_0 in M has length 0, we show by a straightforward induction on the structure of a that $p \Rightarrow r$. (Note we know there is an a-trajectory starting at s_0 since $s_0 \models \langle a\rangle r$.) And if the a-trajectory has length > 0, we show that an appropriate r' exists, again by induction on the structure of A. It should be noted that the formula r' corresponds exactly to the formula p_0 of Lemma 4.14(a). ∎

We assume for ease of exposition that the formula p_0 mentions only two primitive programs, say A_1 and A_2. It will be clear that the procedure is not affected by the presence of more primitive programs. After we are done

with the procedure above, from Lemma 4.21.4 it follows that we have "massaged" all the formulas into the form $\langle A_i\rangle r$ or $\neg\langle A_i\rangle r$, i = 1, 2. Once we have the formulas in this form, we know which formulas must be true at the states which are successors of s_0. Thus for i = 1, 2, let

$$G_i = \{q \in F^0_{x_0} \mid q \text{ is of the form } \langle A_i\rangle r \text{ or } \neg\langle A_i\rangle r\} \text{ and}$$
$$G_i' = \{r \mid \langle A_i\rangle r \in G_i\} \cup \{\neg r \mid \neg\langle A_i\rangle r \in G_i \text{ and for some } r' \ \langle A_i\rangle r \in G_i\}.$$

We call G_i' the *A_i-successor* of F^0. Clearly G_i' is made up of formulas which must be true at the A_i-successor of s in M, and thus the conjunction of the formulas in G_i' is satisfiable. Note that if all the formulas in G_i are of the form $\neg\langle A_i\rangle r$, then $G_i' = \varnothing$. We extend the relation $\Rightarrow$ of 4.21.6 so that for $\langle A_i\rangle r$ in $F^0_{x_0}$, $\langle A_i\rangle r \Rightarrow r$ in G_i'; similarly, if $G_i' \neq \varnothing$, $\neg\langle A_i\rangle r$ in $F^0_{x_0} \Rightarrow \neg r$ in G_i'.

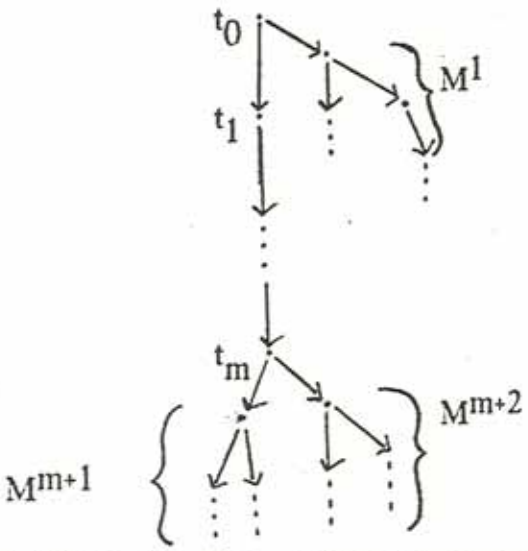
Using Lemma 4.21.3 to obtain the second inequality, we can easily check

$$\|G_1' \cup G_2'\| \leq \|G_1 \cup G_2\| = \|F^0_{x_0}\| \leq \|F^0_0\| = \|F^0\|.$$

Thus we have either $\|G_1'\|, \|G_2'\| < \|F^0\|$, or, without loss of generality, $\|G_1'\| = \|F^0\|$ and $\|G_2'\| < \|F^0\|$ (since it is easy to see that we cannot have $\|G_1'\| = \|G_2'\| = \|F^0\|$). If $\|G_1'\| = \|F^0\|$, let $F^1 = G_1'$ and $G^1 = G_2'$. Repeat the above procedure with F^1 in place of F^0. Eventually one of two things will happen. Either we have a finite sequence of subsets of $FL_s(p_0)$, say $F^0, \ldots, F^m, G^1, \ldots, G^m$, with $\|F^0\| = \|F^1\| = \ldots = \|F^m\|$, and $\|G^i\| < \|F^{i-1}\|$, such that *both* successors of F^m have weight less than $\|F^m\| = \|F^0\|$ (note that $\|G_1'\|, \|G_2'\| < \|F^0\|$ is a special case of this) or we get infinite sequences $F^0, F^1, \ldots$, and $G^1, G^2, \ldots$ with $\|F^0\| = \|F^1\| = \ldots$ and $\|G^i\| < \|F^{i-1}\|$. In either case there corresponds a trajectory $(s_0, s_1, \ldots)$ in M such that for all $q \in F^i$, $M, s_i \models q$. And if $G^i \neq \varnothing$, there is a state u_i which is a successor of s_i such that for all $q \in G^i$, $M, u_i \models q$.

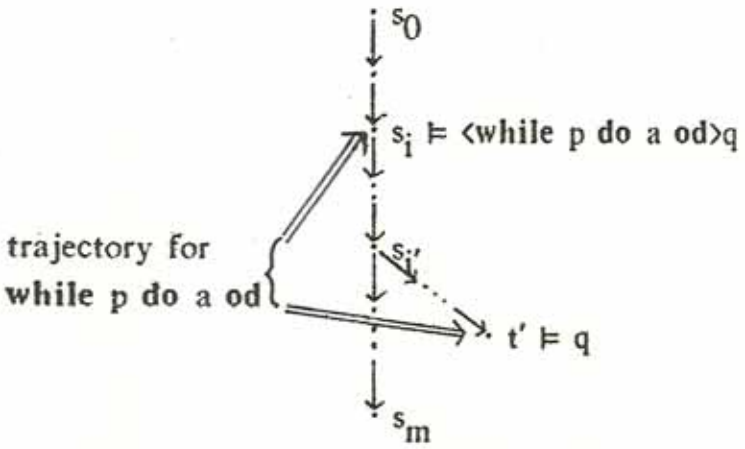
In the first case we construct a tableau M′ as follows. The "backbone" of M′ are the states $t_0, \ldots, t_m$ which correspond to $F^0, \ldots, F^m$; i.e. the formulas true at t_i are exactly those in $\cup_x F^i_x$. Let G^{m+1} and G^{m+2} be the successors of F^m. Each of $G^1 \ldots, G^{m-2}$ has lower weight than F^0, so, by the induction hypothesis, if $G^y \neq \varnothing$ there is a treelike model M^y of the

appropriate depth such that all the formulas of G^y are true at its root. By attaching these models at the appropriate places we get this picture for M′:



Of course, the edges not contained in $M^1, \ldots, M^{m+2}$ are labelled by either A_1 or A_2, depending on whether F^{i+1} or G^{i+1} is an A_1- or A_2-successor of F^i.

To show that M′ is indeed a tableau, the only condition that requires checking is 3.4.13. In fact, we only need show it in the case M′, $t_i \models$ ⟨**while** p **do** a **od**⟩q for $i \leq m$ (the other possibilities are covered by the fact that $M^1, \ldots, M^{m+2}$ are models). Let $(s_0, \ldots, s_m)$ be the trajectory in M corresponding to $(t_0, \ldots, t_m)$. Then if M′, $t_i \models$ ⟨**while** p **do** a **od**⟩q, we must also have M, $s_i \models$ ⟨**while** p **do** a **od**⟩q. Thus there is a **while** p **do** a **od**-trajectory in M starting with s_i and ending with a state t′ such that M, $t' \models q$. Either this trajectory is completely contained within $(s_0, \ldots, s_m)$, or there is a state $s_{i'}$ where it branches away as shown in the figure below:



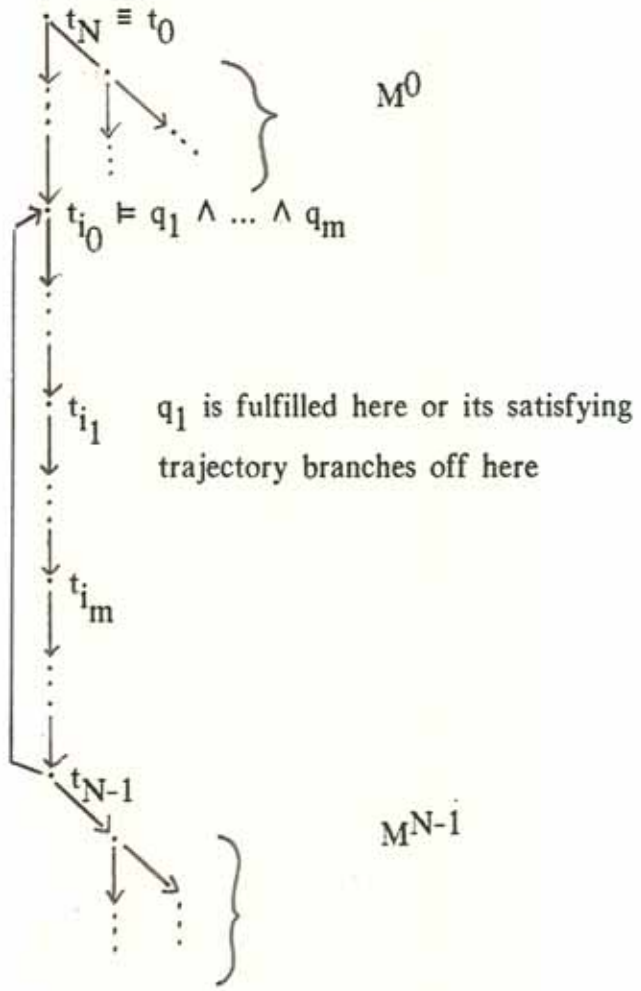
In the former case we can easily check the corresponding trajectory in M′ is a **while** p **do** a **od** trajectory whose final state satisfies q. In the latter, combining the ideas of Lemmas 4.14(a) and 4.21.7, we can show there is a formula r in $F^{i'}$ of the form $\langle A\rangle\langle a_1\rangle...\langle a_h\rangle q$ ($h \geq 0$) such that $\langle$**while** p **do** a$\rangle$q in $F^i \Rightarrow$ r in $F^{i'}$ and $(n_i, B_i, ..., B_{i'-1}, n_{i'})\cdot L_r \subseteq L_{\langle \textbf{while p do a od}\rangle q}$ (where $n_j \subseteq F^j$ for $i \leq j \leq i'$ and $B_j = A_1$ or A_2 is the appropriate program joining F^j and F^{j+1}). Then $\langle a_1\rangle...\langle a_h\rangle q \in G^{i'}$, and since the root of $M^{i'}$ models all the formulas in G, we are done. Essentially what has happened is that there is a formula r′ (= $\langle a_1\rangle...\langle a_h\rangle q$) true at a node which in not on the backbone, but is the *child* of a node on the backbone, such that if r′ is satisfied at this node, $\langle$**while** p **do** a **od**$\rangle$q will be satisfied at the root. (In the terminology of [BHP], if $\langle$**while** p **do** a **od**$\rangle$q is not fulfilled along $(t_0, ..., t_m)$, then there is a derivative of this formula in some $G^{i'}$, which will be fulfilled in $M^{i'}$.)

By Lemma 3.5, M′ can be converted to a treelike model. By the induction hypothesis each M^i has depth $\leq jn2^{n+1} + (k-2)2^n + n$, so the treelike model derived from M has depth $\leq m + 1 + jn2^{n+1} + (k-2)2^n + n$. If $m < 2^n$, the depth is $\leq jn2^{n+1} + (k-1)2^n + n$ as required. But the proof of Theorem 4.17 shows that we can assume without loss of generality that m is indeed $< 2^n$. Otherwise we can use the "cut and paste" argument of Theorem 4.13: if $m \geq 2^n$ we will have $i < i' \leq m$ such that $t_i \equiv t_{i'}$, and by identifying t_i and $t_{i'}$, and eliminating all the descendants of t_i which are not descendants of $t_{i'}$ we still get a model for which all the formulas of s_0 are true at the root. We can repeat this procedure until we get a model of the required depth.

Now we must deal with the second case. Here we have an infinite sequence $F^0, F^1, F^2, ...$, all with the same weight. Note that it follows that no formula in G^i can be of depth = j (since $\|G^i \cup F^i\| \leq \|F^0\|$). Thus for all i, $\|G^i\| \leq (j-1, n-1)$. Just as above, we can construct a treelike tableau, whose backbone will be the infinite sequence $t_0, t_1, ...$, where the formulas true at t_i will be exactly those in $\cup_x F^i_x$. Then if $G^{i+1} \neq \varnothing$, we append to t_i the model M^{i+1}, at whose root all the formulas in G^i are true. As in Theorem 4.17, we can define an equivalence relation among the t_i so that $t_i \equiv t_{i'}$ exactly if $\cup_x F^i_x = \cup_x F^{i'}_x$. Since there are only finitely many equivalence classes, there must be at least one class with infinitely many representatives. Choose such an equivalence class and let t_{i_0} be its first representative. The basic idea

is that we will convert the infinite tableau to a finite one by identifying equivalent states, but we must be a little careful.

Let $q_1, \ldots, q_m$ be all the formulas true at t_{i_0} of the form $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$, where q is not of the form $\langle b\rangle r$. Turning our attention to q_1 for the moment, we argue just as for $\langle$**while** p **do** a **od**$\rangle$q above that there is a state t_{i_1} such that $(t_{i_0}, \ldots, t_{i_1}) \in \tau(A;a_1;\ldots,a_h)$ and $M', t_{i_1}, \models q$ or the $\tau(A;a_1;\ldots;a_h)$ trajectory in M starting at s_{i_0} branches away from $(s_0, s_1, s_2, \ldots)$ at s_{i_1}. Similarly for $q_2, \ldots, q_m$ we can find states $t_{i_2}, \ldots, t_{i_m}$. We can assume without loss of generality that $i_1 \leq i_2 \leq \ldots \leq i_m$. Choose $N > i_m$ such that $t_{i_0} \equiv t_N$. We get a new treelike tableau M" by identifying t_{i_0} and t_N, and eliminating all the states in M′ below t_N:



To prove that M" is a tableau, just as above we must only check 3.4.13 in the case M", $t_i \models$ ⟨**while** p **do** a **od**⟩q. The only problem arises if the **while** p **do** a **od**-trajectory in M′ starting at t_i goes through t_N. But in this case, just as above there will be a formula r in t_N of the form $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$ such that ⟨**while** p **do** a **od**⟩q in $F^i \Rightarrow$ r in F^N and $(n_i, B_i, \ldots, B_{n-1}, n_N)\cdot L_r \subseteq L_{\langle \textbf{while p do a od}\rangle q}$. But $t_N \equiv t_{i_0}$, so that M′, $t_{i_0} \models \langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$. Our construction guarantees there will be an $A;a_1;\ldots;a_h$-trajectory starting at t_{i_0} which ends in a state satisfying q, so we are done.

Finally we must calculate the depth of M". Since $\|G^i\| \leq (j\text{-}1, n\text{-}1)$ by the induction assumption the depth of M" is $\leq N + (j\text{-}1)n2^{n+1} + (n\text{-}2)2^n + n$. But note that by applying the techniques of Theorem 4.17 to identify equivalent states we can assume without loss of generality that $i_0 \leq 2^n$, that for $1 \leq j' \leq m$ we have $i_{j'} - i_{j'-1} \leq 2^n$, and that $N - i_m \leq 2^n$. Thus we can take $N \leq (m+2)2^n$. But there are at most n formulas in $FL_s(p)$ of the form $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$, so $m \leq n$. Hence $N \leq (n+2)2^n$, and the depth of M" is $\leq jn2^{n+1} + n$. ∎

5. SDPDL is Polynomial Space Complete

5.1 *Theorem*: There is a deterministic algorithm for deciding if an SDPDL formula is satisfiable which runs in polynomial space.

Proof: Let p be an SDPDL formula, $|p| = n$. For reasons of clarity we will initially present an algorithm which is not optimal with respect to space, but nevertheless runs in polynomial space. We will make some remarks on improving the algorithm at the end of the proof.

The algorithm essentially tries to construct the treelike model of Theorem 4.19 without the benefit of the tree model as an oracle. Thus it needs to nondeterministically guess what the oracle would have said. Note that we will need Theorem 4.19, which says that such a model exists iff p is satisfiable, to establish the correctness of the algorithm. Finally, we use the result of Savitch (cf. [Sav]) which allows us to eliminate nondeterminism and still have a polynomial space algorithm.

At all times the algorithm will be working on a subset of formulas of $FL_s(p)$ analogous to the F_0^0 of Theorem 4.19. There will also be polynomially many other such subsets on a pushdown stack, waiting their turn to be worked on. As in Theorem 4.19, these subsets represent states of the tableau. Since we

can work on only one branch of the tableau at a given time, the sets on the stack represent formulas for which the tableau conditions will be satisfied along a branch of the tableau other than the one on which we are currently working. With each subset is associated a counter (a number $\leq 2^{2n+1}$) and (possibly) one other subset which we call the *backpointer*. This second subset is one which must have the same weight (in the sense of Definition 4.20) as the first subset. It corresponds to the state t_{i_0} in the second half of the proof of Theorem 4.19; i.e. the state to which a backedge (if there is one) will go. (Remember that we are constructing a *treelike* tableau). Initially we work on {p}, with the counter set to 0; the stack is empty.

Suppose at a certain time the algorithm is working on the set $\{q_1, \ldots, q_k\} \subseteq FL_s(p)$ with the counter set to $N \leq 2^{2n+1}$. Call this set F^0_0 and apply rules 1-10 of Theorem 4.21 to get F^0_1, F^0_2, ... Of course rules 4, 5, 8, 9, and 10 required the oracle M, so we replace them by 4′, 5′, 8′, 9′, and 10′ which guess what M would have said. For example, 4′ is:

> if q is of the form ⟨**if** p **then** a **else** b **fi**⟩r then *nondeterministically* set $q_\alpha = p$, $q_\beta = \langle a\rangle r$ or $q_\alpha = \neg p$, $q_\beta = \langle b\rangle r$.

It is easy to check that parts 1, 3, 4, 5, and 6 of Sublemma 4.21 still hold, so that in particular we have that for some $x_0 \leq n$, all the formulas in $F^0_{x_0}$ are of the form ⟨A⟩r or ¬⟨A⟩r. Continue applying the rules until this is the case. Then

1. If for some formula r, $\{r, \neg r\} \subseteq \cup_x F^0_x$, terminate with "p is unsatisfiable" (the particular sequences of choices made by the algorithm was bad).

2. For ⇒ defined as in 4.21.6, check that 4.21.7 holds. That is, for $q \in \cup_x F^0_x$ of the form ⟨a⟩r, check that either $q \Rightarrow r$ or there exists r′ of the form $\langle A\rangle\langle b_1\rangle\ldots\langle b_m\rangle r$ such that $q \Rightarrow r'$. If not, terminate with "unsatisfiable".

Let us again assume for ease of exposition that the only primitive programs mentioned in p are A_1 and A_2. We then form the sets G_1' and G_2' just as in Theorem 4.19. If there is no backpointer associated with the original set F^0_0, we proceed as follows.

3. If both G_1' and G_2' are empty, then we work on the top element in the stack. If the stack is empty, we have succeeded in finding a treelike tableau for p; terminate and return "satisfiable".

4. If only one of G_1' and G_2' is empty, we work on the non-empty one next with the counter set to N + 1. If they are both nonempty, we work on the one with lower weight with the counter set to N + 1, while the one with higher weight goes on the stack, again with its counter set to N + 1. (If G_1' and G_2' have the same weight, then we work on G_1' with counter set to N + 1.) If the set with higher weight has weight equal $\|F_0^Q\|$, we can (nondeterministically) choose to associate F_0^Q with it as a backpointer. (Note that by the inequalities of Theorem 4.19, it cannot be the case that both sets have weight equal $\|F_0^Q\|$.) In any case, if $N + 1 > 2^{2n+1}$ we terminate with "unsatisfiable".

(Note that in Theorem 4.19 we used as an inductive hypothesis that satisfiable sets of lower weight had an appropriate treelike model. Here we actually verify that the sets of lower weight have models before dealing with the sets of higher weight, which get put on the stack.)

If there is a backpointer H associated with F_0^Q, we also assume as an inductive hypothesis that some of the elements in this set have been "checked", while the ones that have not been checked which are of the form $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$ are associated with a formula of the form $\langle b_1\rangle\ldots\langle b_m\rangle q$ ($m \geq 0$) in F_0^Q. As mentioned above, H corresponds to the set t_{i_0} in Theorem 4.19. There we had to find, for each formula of the form $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$ in t_{i_0}, a state t_j where the formula is either satisfied or its satisfying trajectory involves branches off to a state of lower weight. The formulas for which such a state has been found are the ones that are checked off. Once all the formulas of the form $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$ in the associated set have been checked off, we can look for a state like t_N which is equivalent to t_{i_0}, so we can branch back. We proceed as follows:

5. We know that at most one of G_1' or G_2' has weight equal to that of F_0^Q. If neither does, terminate with "unsatisfiable". Otherwise, suppose without loss of generality that $\|G_1'\| = \|F_0^Q\|$. We then associate H with G_1', checking off all the formulas that were checked off before. We check off other formulas as follows. Suppose $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$ has not been checked off. Thus it is associated with a formula of the form $\langle b_1\rangle\ldots\langle b_m\rangle q$ in F_0^Q. By the check made in step 2 above, we know that either $\langle b_1\rangle\ldots\langle b_m\rangle q \Rightarrow r$ where r is q or of the form $\langle A_i\rangle\langle c_1\rangle\ldots\langle c_k\rangle q$, i = 1,2. If r is q or begins with $\langle A_2\rangle$, we check off the formula $\langle A\rangle\langle a_1\rangle\ldots\langle a_h\rangle q$. Otherwise we associate it $\langle c_1\rangle\ldots\langle c_k\rangle q \in G_1'$.

6. If $G_1' = H$ and all the formulas in H have been checked off, we set G_1' to the empty set, forget about H, and proceed as in steps 3 and 4. (Essentially we have branched back, so we do not have to worry about satisfying this set of formulas any more. We can continue working on the other sets in the stack.) If $G_1' \neq H$ and if $G_2' \neq \varnothing$, we work on G_2' with counter set to N + 1, putting $(G_1', N + 1, H)$ on the stack, while if $G_2' = \varnothing$, we work on $(G', N + 1, H)$. Of course, if $N + 1 > 2^{2n+1}$ we terminate with "unsatisfiable".

Since the procedure is nondeterministic, we take p to be satisfiable iff there is some sequence of choices which returns "p is satisfiable".

We must now show that the above algorithm is correct. To see that it terminates for all possible sequences of choices, suppose in the general case that there are k primitive programs mentioned in p. Then it is easy to show by induction on i that the program works on a set whose counter is set to i at most k^i times. Since the counter is always $\leq 2^{2n+1}$, for any sequence of guesses the algorithm terminates after at most $k^{2^{2n+1}}$ steps (where a "step" is the whole process outlined above).

If a sequence of choices made by the algorithm returns "p is satisfiable", the proof of Theorem 4.19 shows that we have indeed constructed a treelike tableau for p, so p really is satisfiable. Conversely, if $p \in SDPDL_i$ is satisfiable, we know by Theorem 4.19 there is a treelike tableau for p of depth $in2^{n+1} + n < 2^{2n+1}$ (since $i < n$). Thus if the algorithm makes guesses corresponding to the state of affairs on this tableau, it will output "p is satisfiable".

Finally, we must check that the algorithm works in polynomial space. To see this, let j_x be the number of formulas in $FL_s(p)$ of depth x. Note $\Sigma_x\, j_x \leq 2n$. The only weights attainable by subsets of $FL_s(p)$ are those of the form (x,y) where $j_x \geq 1$ and $y \leq j_x$. Thus there are at most 2n weights attainable by subsets of $FL_s(p)$. Then it is easy to prove that if at a given stage in the algorithm we are working on a set with weight (x,y) which is the m^{th} highest attainable weight, then there are at most m(k-1) triple of (set, counter, (backpointer)) on the stack (where k is, as above, the number of primitive programs appearing in p). The proof depends crucially on the fact that we always work on the set of lowest weight possible. If any triples are added to the stack at the end of a step (and at most k-1 triples can be added at any step), then at the next step we must be working on a set of lower weight. Since $k < n$, and $m \leq 2n$, there are always $< 2n^2$ triples on the stack. Thus the algorithm only uses polynomial space.

We now take a closer look at exactly how much space is required. First of all note that instead of putting formulas on the stack, we can instead store pointers to formulas in $FL_s(p)$. Moreover, it is easy to rework the argument given in the previous paragraph to show that when we are working on a set with the m^{th} highest attainable weight, there will be at most m groups of 2n pointers on the stack. (Although k-1 triples can be added to the stack at the end of any step, the proof shows that these triples involve $\leq$ 2n formulas, and hence $\leq$ 2n pointers.) Since each pointer uses O(log(n)) space, the algorithm runs in nondeterministic space $O(n^2 \cdot (\log(n))$. By Savitch's Theorem, it runs in deterministic space $O(n^4(\log(n))^2)$. ∎

5.2 *Remarks*: 1. Essentially, the algorithm attempts to construct a treelike model for p in a depth-first manner. Every time there is a choice of paths to follow, we follow one of them and put the other sons of the node on the stack. Since we want to work in polynomial space we must be careful that the stack does not grow too large. we ensure this by following the path defined by the son of least weight. but the use of weights was not essential in the algorithm because of the following general fact: we can do a depth-first search of a tree with n nodes and outdegree k with a stack of height $\leq k \cdot \log_k(n)$. (of course, the search will not necessarily proceed down the leftmost path; we must guess the appropriate path at all times.) the proof follows by an induction on the height of the tree. since the treelike model which the algorithm tries to construct has depth $\leq 2^{2n}$ and polynomially many nodes at each depth, the result follows. however, the use of weights does eliminate the nondeterminism at this stage and gives a slightly sharper upper bound on the amount of space required.

2. A similar theorem holds for $SDPDL_{pt}$ formulas, but in this case the proof is easier since by Theorem 4.13 $SDPDL_{pt}$ formulas have finite tree models. This means we do not have to take care of the possibility that a path might branch back to an ancestor, and we do not need backpointers.

3. Polynomial space is equivalent to polynomial parallel time for a large class of parallel machine models (including parallel RAMS, vector machines, alternating machines (cf. [CKS, FW, G]). Thus our decision procedure can perhaps be implemented on such a machine to run in polynomial time, and hence might be usable for automatically verifying that real world programs written in a deterministic well-structured language are correct.

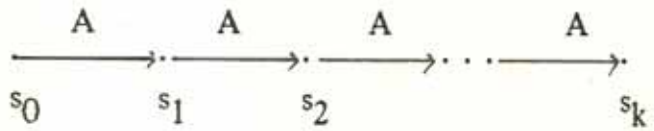
4. A *Ianov scheme* is an uninterpreted deterministic program scheme with only one program variable (cf. [Gr]). Two schemes are strongly equivalent if, given any interpretation of the symbols in the schemes, they both compute the same result or they both fail to halt. As noted in [FL], given a Ianov scheme a we can effectively construct a PDL program, say a′, such that a and b are strongly equivalent Ianov schemes iff $\langle a' \rangle Q \equiv \langle b' \rangle Q$, where Q does not appear

in either a′ or b′. But we can easily show that without loss of generality, a′ and b′ are $SDPDL_{pt}$ programs. Thus we have a polynomial space procedure for deciding strong equivalence of Ianov schemes.

5.3 *Theorem*: The decision problem for $SDPDL_{pt}$, and hence SDPDL, is polynomial space hard.

Proof: We use ideas are similar to those used in Cook's proof that SAT is NP-hard; (cf. [HU, pp. 325-327]). Given $L \in PSPACE$, we can assume without loss of generality (cf. [HU, p. p.289]) that L is accepted by a one-tape deterministic Turing Machine *M* which, for some polynomial p, runs in space $\leq p(n)$ on inputs of length n. We will construct an $SDPDL_{pt}$ formula f(x) which simulates the computation of *M* on input x, where f(x) is computable in polynomial time (and log space) from x. In particular f(x) will be satisfiable iff *M* accepts input x.

Suppose *M* has state space Q and uses tape alphabet Γ. We can describe the state of *M* at a given time by an ID (instantaneous description) of exactly p(n) symbols in $\Gamma' = (Q \times \Gamma) \cup \Gamma$. To simulate this in $SDPDL_{pt}$ we use primitive predicates P_{wi}, P'_{wi} where $1 \leq i \leq p(n)$ and w ranges over Γ'. The formula f(x) will involve only one atomic program symbol, A, so if f(x) is satisfiable the satisfying model must look like:



The intuition is that P_{wi} will be true at state s_j iff, when we run *M*, the i^{th} position on the tape at time j contains symbol w. (If $w = (q,z) \in Q \times \Gamma$ then the head of *M* is also at the i^{th} position reading symbol z and the machine is in state q). Moreover P_{wi} is true at state s_j iff P'_{wi} is true at state s_{j+1}. Thus the P'_{wi} keep track of the values of the P_{wi} at the previous time.

We can assume without loss of generality that *M* has only one accepting state, q_{acc}. Let the formula *accept* be:

$$\bigvee_{w \in (\{q_{acc}\} \times \Gamma),\ 1 \leq i \leq p(n)} P_{wi}$$

The formula f(x) will have to state the following:

1. The only P_{wi}'s true at s_0 are those that describe the initial ID.

2. In every state s_j, the P_{wi}'s that are true in s_j correspond to a string of symbols, in that for all i there is a unique $w \in \Gamma'$ such that P_{wi} is true.

3. P_{wi} is true at s_j iff P'_{wi} is true at s_{j+1}.

4. The ID true at state s_j follows from the one true at s_{j-1} (for $j \geq 1$) by the indicated move of *M*.

We take f(x) to be the conjunction of 4 the four formulas $f_1(x)$, $f_2(x)$, $f_3(x)$, and $f_4(x)$, which enforce conditions 1 through 4 respectively. Let $x = x_0 \ldots x_{n-1}$, where $x_i \in \Gamma$, let g_0 be the initial state of *M*, and let $b \in \Gamma$ denote the blank symbol.

Let *string* be the formula which says that the P_{wi}'s correspond to a string of symbols:

$$\wedge_{1 \leq i \leq p(n)} (\vee_{w \in \Gamma'} (P_{wi} \wedge (\wedge_{u \neq w} \neg P_{wi})))$$

Then $f_1(x)$ is

$$P_{\{q_0,x_0\}} \wedge P_{x_1} \wedge \ldots \wedge P_{x_{n-1},n-1} \wedge (\wedge_{n \leq i \leq p(n)} P_{bi}) \wedge \textit{string}$$

and $f_2(x)$ is the formula

$$\langle \textbf{while } \neg \textit{accept } \textbf{do } (A;\textit{string}?) \textbf{ od} \rangle \textit{true}.$$

Let $f_3(x)$ be the formula

$$\langle \textbf{while } \neg \textit{accept } \textbf{do (if } P_{wi} \textbf{ then } (A;P'_{wi}?) \textbf{ else } (A;\neg P'_{wi}?) \textbf{ fi) od} \rangle \textit{true}$$

Clearly this enforces condition 3.

To see how to write the fourth formula, note that the symbol in the i^{th} position of a given ID is completely determined by the symbols appearing in positions i-1, i, 1+1 in the previous ID. We can therefore easily (again, cf. [HU, p.327]) specify a predicate $Q(w_1,w_2,w_3,w_4)$ that is true iff symbol w_4 could appear in position i of some ID given that w_1, w_2, and w_3 appear in positions i = 1, i, i+1 of the previous, ID. (If i = 1 we take w_1 = b; if i = p(n) we take w_3 = b). Using the P_{wi}'s (which keep track of what happened at the previous ID). We can now express the formula *consistent*:

$$\wedge_{1 \leq i \leq p(n)} (\vee_{\{(u,v,y,z) \mid Q(u,v,y,z)\}} P'_{u,i-1} \wedge P'_{vi} \wedge P'_{y,i+1} \wedge P_{zi})$$

Then $f_4(x)$ is the formula

$$\langle \textbf{while } \neg \textit{accept } \textbf{do } (A;\textit{consistent}?) \textbf{ od} \rangle \textit{true}.$$

Clearly if $x \in L$, then $f(x)$ is satisfied in a model such as the one above, where there are k states corresponding to the k steps of the accepting computation by *M*. Conversely, if $M, s_0 \models f(x)$ then the graph corresponding to M looks like the one pictured above, except that it may be an infinite straight line. Let k be the least number such that $M, s_k \models \mathit{accept}$. Then it is easy to check that *M* accepts x in k steps, where $M, s_j \models P_{wi}$ iff w is in the i^{th} position of the tape at time j. ∎

From Theorems 5.1 and 5.3 and the fact that p is valid iff $\neg p$ is not satisfiable we immediately obtain

5.4 *Corollary*: The satisfiability and validity problems for SDPDL and $SDPDL_{pt}$ are polynomial space complete.

5.5 *Remark*: In [BHP] it was already shown that every satisfiable DPDL (and hence SDPDL) formula has a model of size $\leq 4^n \cdot n^2$. The question arises if we can do any better for SDPDL formulas. The answer is essentially no, since by using the techniques of Theorem 5.3 we can encode the computation of a Turing machine which counts up to 2^n and then halts into an SDPDL formula. This formula will have size O(n), and the smallest model that satisfies it will have size 2^n.

6. Expressiveness

6.1 *Definition*: For two logical languages $\mathcal{L}$ and $\mathcal{M}$, we say $\mathcal{L}$ is at least as *expressive* as $\mathcal{M}$, and write $\mathcal{L} \leq \mathcal{M}$, iff, for every formula $p \in \mathcal{L}$, there is a formula $p' \in \mathcal{M}$ such that $\models p \equiv p'$. $\mathcal{M}$ and $\mathcal{L}$ are said to be equally expressive, written $\mathcal{L} \approx \mathcal{M}$, if $\mathcal{L} \leq \mathcal{M}$ and $\mathcal{M} \leq \mathcal{L}$. $\mathcal{L}$ is less expressive than $\mathcal{M}$, written $\mathcal{L} < \mathcal{M}$, if $\mathcal{L} \leq \mathcal{M}$ and $\mathcal{L} \not\approx \mathcal{M}$.

Berman, Peterson, and Paterson show (in [B], [BP], and [P]) that for all $i \geq 0$, $PDL_i < PDL_{i+1}$. In fact, these proofs also show $(S)DPDL_i < (S)DPDL_{i+1}$. We use our structure theorem (Theorem 4.12) to rederive these results and extend them to show that $SDPDL_i < DPDL_i$ and SDPDL < DPDL.

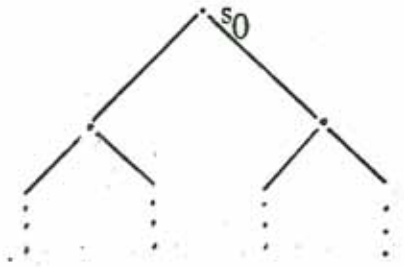
6.2 *Remark*: Meyer and Winklmann show in [MW] that $SDPDL_{pt} < DPDL_{pt}$, by showing that the $DPDL_{pt}$ formula $\langle A^* \rangle [A^*]P$ is not equivalent to any $SDPDL_{pt}$ formula. Their proof does not extend to full SDPDL. It is easy to see that for any formula p,

$$\models \langle A^* \rangle p \equiv \langle \textbf{while } \neg p \textbf{ do } A \textbf{ od} \rangle \mathit{true}, \text{ and hence}$$
$$\models [A^*]p \equiv [\textbf{while } p \textbf{ do } A \textbf{ od}] \mathit{false}.$$

From these remarks, it is easy to see that $\langle A^* \rangle [A^*]P$ *is* equivalent to an SDPDL formula. However we still get

6.3 *Theorem*: For all $i \geq 0$, $SDPDL_i < SDPDL_{i+1}$, and $SDPDL < DPDL$.

Proof: Let $\Sigma_0 = \{A,B\}$, and let M be a model whose graph is a full binary tree rooted at s_0 as illustrated below:



Let the formula p_0 be *true*, and let p_{i+1} be

$$[\textbf{while } \langle A\rangle\langle B\rangle p_i \textbf{ do } A \textbf{ od}]\mathit{false}.$$

Clearly p_i has depth i and M, $t \models p_i$ for all i and all states t. Moreover, an easy induction shows that there is a constant c_i such that if N is any subtree of M which sets p_{i+1} at s_0 (as in Theorem 4.12), then N has greater than $c_i \cdot k^i$ nodes at depth i. Thus, by Theorem 4.12, p_{i+1} is not equivalent to any $SDPDL_i$ formula.

Now consider the $DPDL_{pt}$ formula $p = [(A \cup B)^*](\langle A\rangle \mathit{true} \wedge \langle B\rangle \mathit{true})$. Again we have M, $s_0 \models p$, but there is clearly no proper submodel of M which sets p at s_0. Thus, by Theorem 4.12, p is not equivalent to any SDPDL formula. It then follows that the $DPDL_0$ formula $[(A \cup B)^*]P$ is also not equivalent to any SDPDL formula, for if it were equivalent to some SDPDL formula q, then it is easy to check that p would be equivalent to q with all the occurrences of P replaced by $(\langle A\rangle \mathit{true} \wedge \langle B\rangle \mathit{true})$. ∎

Essentially, the proof above shows that an SDPDL program cannot examine every node of a full binary tree, while $(A \cup B)^*$ can.

Combining these results with those of [B] and [P], we get the following picture, where languages not connected by a chain of <'s are incomparable in expressive power:

$$\begin{array}{ccccccc} DPDL_0 & < & DPDL_1 & < & \ldots & < & DPDL \\ \vee & & \vee & & & & \vee \\ SDPDL_0 & < & SDPDL_1 & < & \ldots & < & SDPDL. \end{array}$$

References

[B] F. Berman, Expressiveness hierarchy for PDL with rich tests, University of Washington, TR78-11-01, 1978.

[BHP] M. Ben-Ari, J. Y. Halpern, and A. Pnueli, Finite models of deterministic propositional dynamic logic, to appear *in* "Proceedings of ICALP, 1981". A revised version appears as Deterministic propositional dynamic logic: finite models, complexity, and completeness, MIT/LCS/TM-190, January, 1981.

[BP] F. Berman and M. Paterson, Test-free propositional dynamic logic is strictly weaker than PDL, University of Washington, TR77-10-02.

[Ch] B. S. Chlebus, On the computational complexity of satisfiability in propositional logics of programs, unpublished manuscript.

[CKS] A. Chandra, D. Kozen, and L. Stockmeyer, Alternation, *Journal of the Association for Computing Machinery*, 28:1, pp. 114-133, January, 1981.

[FL] M. J. Fischer and R. E. Ladner, Propositional modal logic of programs, *in* "Proceedings of the Ninth Annual ACM Symposium on Theory of Computing", 286-294, Association for Computing Machinery, New York, N. Y., 1977. A revised version appears as: Propositional dynamic logic of regular programs, *Journal of Computer and System Science* 18:2, pp. 194-211, 1979.

[FW] S. Fortune and J. Wyllie, Parallelism in random access machines, *in* "Proceedings of the 10th Annual ACM Symposium on Thoery of Computing", pp. 114-118, 1978.

[G] L. Goldschlager, Synchronous parallel computation, Ph.D. Thesis and TR-114, Department of Computer Sciences, University of Toronto, December, 1977.

[Gr] S. A. Greibach, *Theory of Program Structures: Schemes, Semantics, Verification*, Lecture Notes in Computer Science, 36, Springer-Verlag, N. Y., 1975.

[Hal] J. Y. Halpern, On the expressive power of dynamic logic, II, to appear.

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

[HU] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[KP] D. Kozen and R. Parikh, An elementary proof of the completeness of PDL, *Theoretical Computer Science* **14**:1, pp. 113-118, 1981.

[Mir] G. Mirkowska, On formalized systems of algorithmic logic, *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* **22**, pp. 421-428, 1974.

[MT] A. R. Meyer and J. Tiuryn, The equivalence of several logics of programs, to appear.

[MW] A. R. Meyer and K. Winklmann, On the expressive power of dynamic logic, MIT/LCS/TM-157, February, 1980. A preliminary report appears in "Proceedings of the 11th Annual ACM Conference on Theory of Computing", May, 1979.

[P] G. L. Peterson, The power of tests in propositional dynamic logic, University of Rochester, TR47, 1978.

[Pr1] V. R. Pratt, Semantical considerations of Floyd-Hoare logic, *in* "17th IEEE Symposium on the Foundations of Computer Science", pp. 109-121, 1976.

[Pr2] V. R. Pratt, A practical decision method for propositional dynamic logic, *in* "10th Annual ACM Symposium on the Theory of Computation", pp. 326-337, 1977. A revised version appears as: A near optimal method for reasoning about action, *Journal of Computer and Systems Science* **20**:2, pp. 231-254, 1980.

[Pr3] V. R. Pratt, Models of program logics, *in* "20th IEEE Symposium on the Foundations of Computer Science", pp. 115-122, 1979.

[Sal] A. Salwicki, Formalized algorithmic languages, *Bull. Acad. Pol. Sci., Ser. Math. Astr. Phys.* **18**:5, pp. 227-232, 1970.

[Sav] Savitch, W.J., Relationships between nondeterministic and deterministic tape complexities, *Journal of Computer and Systems Sciences*, **4**:2, pp. 147-192.