THE PROPOSITIONAL DYNAMIC LOGIC OF DETERMINISTIC, WELL-STRUCTURED PROGRAMS*

Joseph Y. HALPERN** and John H. REIF

Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138, U.S.A.

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, and give a complete axiomatization for it. The results rely on structure thorems for models of satisfiable SDPDL formulas, and the proofs give insight into the effects of nondeterminism on intractability and expressiveness in program logics.

Key words. Strict deterministic propositional dynamic logic, propositional dynamic logic, decision procedure, polynomial space complete, expressiveness, well-structured programs.

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, which compares favorably to other logics of programs, as well as other desirable properties (cf. [7, 25, 15]).

By analogy to regular languages, PDL makes use of the nondeterministic program constructors * and $\cup$. However, in programming languages used today, all constructs

* This is an expanded version of [11] given at the 22nd Symposium of the Foundations of Computer Science in Milwaukee, Wisconsin, October 1980. This research was partially supported by the National Science and Engineering Research Council of Canada, NSF Grants MCS80-10707 and MCS79-21024, and ONR Grant N00014-80C0647.

** Current address: IBM Research, San Jose, CA 95193, U.S.A.

Preprint of paper appearing in Theoretical Computer Science, Volume 27, Issues 1-2, 1983, Pages 127-165.

are *deterministic*. Indeed, historically, much of the research in logics of programs had dealt only with deterministic programs (cf. the work of Salwicki and his coworkers in Algorithmic Logic, e.g., [27, 18]). One way of excluding nondeterminism from PDL is to give the primitive programs deterministic semantics, and then restricting the use of * and $\cup$ so that they only occur in contexts which yield deterministic, well-structured programs. This gives us 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 this restriction to deterministic programs give us an easier decision procedure. (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". (The latter result answers an open question of Harel [12].) In fact we show that the problem of deciding SDPDL satisfiability is complete for polynomial space (cf. Theorems 5.1 and 6.3). This result is also shown to hold for other logics of programs, such as linear time temporal logic (cf. [8, 22, 29]).

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 and 4.21).

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. This situation has been shown to hold even in the first order case (cf. [10, 4]). Thus first order regular dynamic logic can be shown to be more expressive than its deterministic counterpart, answering another open question of Harel [12] (cf. [17], which studies the quantifier-free case). By contrast, Meyer and Tiuryn [16] have shown 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.

Results similar to our Theorems 5.1 and 5.5 on polynomial space completeness have been announced independently by Chlebus [5].

The rest of the paper is organized as follows. We review the syntax and semantics of PDL in Section 2, and introduce the modifications necessary to get SDPDL. Section 3 recalls the basic notions of the Fischer–Ladner closure and tableaux, which have proved very useful in obtaining results about PDL, and gives the modifications of these notions appropriate to SDPDL. In Section 4 we introduce *tree models* and *treelike models*, and prove a number of technical results about them which we will

need for our decision procedure and expressiveness results. 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. Finally, in Section 7 we give a complete axiomatization of SDPDL.

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. [1]). We briefly review the syntax and semantics of PDL and DPDL.

2.1. Syntax. The alphabet for PDL (as well as DPDL), $\mathscr{S}$, consists of a set Φ_0, whose elements are called atomic formulas, a set Σ_0, whose elements are called atomic programs, and the special tokens $\cup$, ;, *, ?, $\neg$, $\langle$, $\rangle$, (,) and *true*.

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

Rule 1. Any atomic program in Σ_0 is a program.

Rule 2. If a and b are programs, then so are $(a;b)$, $(a \cup b)$ and a^* (we will occasionally omit the parentheses).

Rule 3. Any atomic formula in Φ_0 is a formula; *true* is a formula.

Rule 4. If p is a formula and a is a program, then $\neg p$ and $\langle a \rangle p$ (pronounced 'diamond a p') are formulas.

Rule 5. If p is a formula, then $p?$ is a program.

We also use the following abbreviations:

$$p \wedge q \quad \text{for } \langle p? \rangle q, \qquad p \vee q \quad \text{for } \neg(\neg p \wedge \neg q),$$

$$p \to q \quad \text{for } \neg p \vee q, \qquad p \equiv q \quad \text{for } (p \to q) \wedge (q \to p),$$

$$\mathit{false} \quad \text{for } \neg \mathit{true}, \qquad [a]p \text{ ('box } a\ p\text{')} \quad \text{for } \neg\langle a \rangle \neg p.$$

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

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_M, π_M, ρ_M) where S_M is a set whose elements are called *states*, $\pi_M : \Phi \to \mathscr{P}(S_M)$ is an assignment of formulas to sets of states and $\rho_M : \Sigma \to \mathscr{P}(S_M \times S_M)$ is a mapping of programs into binary relations on S_M which satisfies the following constraints (we omit the subscript M here and

elsewhere in the paper if the structure is clear from the context):

(1) $\pi(\neg \mathit{true}) = \emptyset$,

(2) $\pi(p) \cap \pi(\neg p) = \emptyset$,

(3) $\rho(a;b) = \rho(a) \circ \rho(b)$ (composition of relations),

(4) $\rho(a \cup b) = \rho(a) \cup \rho(b)$ (union of relations),

(5) $\rho(a^*) = (\rho(a))^* = \bigcup_{n \geq 0} \rho(a^n)$ (reflexive and transitive closure)
(where $a^0 = \mathit{true}?$, and $a^n = a; \cdots; a$, n times),

(6) $\rho(p?) = \{(s, s) \mid s \in \pi(p)\}$.

A *DPDL structure* satisfies in addition:

(7) 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:

(8) $\pi(\neg p) = S - \pi(p)$,

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

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)–(6), (8) and (9) hold. Moreover, if ρ' satisfies condition (7), then so does ρ. Thus, for a (D)PDL model, π and ρ are completely defined by their actions on the atomic formulas and programs.

(2) We will say *t is an a-successor of s* in a stucture if $(s, t) \in \rho_M(a)$. In a DPDL structure, each $s \in S$ has at most one A-successor for all $A \in \Sigma_0$. Any (D)PDL structure $M = (S, \pi, \rho)$ can be viewed as a directed graph, where the nodes correspond to 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. If we label the node s by $\{q \mid s \in \pi(q)\}$, then the graph together with this labelling uniquely determine M. By the previous remark, if M is a model, then it suffices to label s by $\{P \in \Phi_0 \mid s \in \pi(P)\}$.

2.6. Definition. Let $M = (S, \pi, \rho)$. Then

(1) $M, s \models p$ (p is *true in* or *satisfiable at* $s \in S$) iff $s \in \pi(p)$,

(2) $M \models p$ (p is *satisfiable in* M) iff, for some $s \in S$, we have $M, s \models p$.

If M is a model and $M \models p$, then we say that M is a *model for* p.

(3) 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. SDPDL. 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 $((p?; a); \neg p?)$, which we abbreviate 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 is the same as that of DPDL.

For $p \in \Phi_s$, let $|p|_s$ be the length of p measured as a string over

$$\Phi_0 \cup \Sigma_0 \cup \{\textbf{if}, \textbf{then}, \textbf{else}, \textbf{fi}, \textbf{while}, \textbf{do}, \textbf{od}, (,), \langle, \rangle, ;\}.$$

We omit the subscript s in $|p|_s$ when it is clear from context.

The following two lemmas describe 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. Then*

(1) $\rho(\textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi})$
$= \{(s, t) \mid (s, t) \in \rho(a)$ *and* $s \in \pi(p)\} \cup$
$\{(s, t) \mid (s, t) \in \rho(b)$ *and* $s \in \pi(\neg p)\}$,

(2) $\rho(\textbf{while } p \textbf{ do } a \textbf{ od}) = \{(s, t) \mid \exists s_0 \cdots \exists s_k (s_0 = s, s_k = t,$
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 Definitions 2.3 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) $\models \langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle q \equiv ((p \wedge \langle a\rangle q) \vee (\neg p \wedge \langle b\rangle q))$.

(3) $\models \langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q \equiv ((\neg p \wedge q) \vee (p \wedge \langle a\rangle\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q))$.

(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) $\models \neg\langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle q \equiv ((p \wedge \neg\langle a\rangle q) \vee (\neg p \wedge \neg\langle b\rangle q))$.

(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. The proof immediately follows 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 (11). 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, $p \in \mathrm{DPDL}_i$. Note that $\mathrm{DPDL} = \bigcup_i \mathrm{DPDL}_i$. Let $\mathrm{SDPDL}_i = \mathrm{DPDL}_i \cap \mathrm{SDPDL}$. Thus, for example, we have

$$\langle(((q?;A)^*;\neg q?\rangle p)?;B \cup (\neg\langle(q?;A)^*;\neg q?\rangle p)?;C\rangle \mathit{true} \equiv$$
$$\equiv \langle \textbf{if } \langle \textbf{while } q \textbf{ do } A \textbf{ od}\rangle p \textbf{ then } B \textbf{ else } C \textbf{ fi}\rangle \mathit{true} \in \mathrm{SDPDL}_2.$$

Finally, let $\mathrm{DPDL}_{\mathrm{pt}}$ (*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 atomic formulas. Again $\mathrm{SDPDL}_{\mathrm{pt}}$ (*poor test* SDPDL) $= \mathrm{DPDL}_{\mathrm{pt}} \cap \mathrm{SDPDL}$.

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 [2] or [12]. For example, the formula $P \wedge Q$ is in DPDL_0 in the hierarchy à 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 $\mathrm{DPDL}_{\mathrm{pt}}$ is called $\mathrm{DPDL}_{0,S}$ in [12] and [2], 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. Definition. The FL_s-closure of an SDPDL formula p_0, $\mathrm{FL}_s(p_0)$, is a slight modification of $\mathrm{FL}(p_0)$ (cf. [7]), the Fischer–Ladner closure of a DPDL formula p_0, designed to respect the syntax of SDPDL. For technical reasons we also close it off with respect to negation. Let F be the least set such that $p_0 \in F$ and

(1) $\langle a\rangle p \in F \rightarrow p \in F$,

(2) $\langle a;b\rangle q \in F \to \langle a\rangle\langle b\rangle q \in F$,

(3) $\langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle q \in F \to \neg p, p, \langle a\rangle q, \langle b\rangle q \in F$,

(4) $\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q \in F \to \neg p, p, \langle a\rangle\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q \in F$,

(5) $\langle p?\rangle q \in F \to p \in F$,

(6) $\neg p \in F \to p \in F$.

Define $\mathrm{FL}_s(p_0) = F \cup \neg F$ (where $\neg F = \{\neg q \mid q \in F\}$).

3.2. Lemma. *If* $|p_0|_s = n$, *then* $|\mathrm{FL}_s(p_0)| \leq 2n$.

Proof. A slight modification of the proof of [7, Lemma 3.2] shows $|F| \leq n$. The result easily follows. □

3.3. Definition. A *tableau for* p_0 is a DPDL structure $M = (S, \pi, \rho)$ such that $\pi(p_0) \neq 0$ and for all $s \in S$ the following conditions hold for all formulas:

(1) $M, s \models \neg\neg p \to M, s \models p$,

(2) $M, s \models \langle a;b\rangle q \to M, s \models \langle a\rangle\langle b\rangle q$,

(3) $M, s \models \langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle q$
$\to [(M, s \models p \text{ and } M, s \models \langle a\rangle q) \text{ or } (M, s \models \neg p \text{ and } M, s \models \langle b\rangle q)]$,

(4) $M, s \models \langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q$
$\to [(M, s \models p \text{ and } M, s \models \langle a\rangle\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q)$
$\text{or } (M, s \models \neg p \text{ and } M, s \models q)]$,

(5) $M, s \models \langle p?\rangle q \to (M, s \models p \text{ and } M, s \models q)$,

(6) $M, s \models \neg\langle a;b\rangle q \to M, s \models \neg\langle a\rangle\langle b\rangle q$,

(7) $M, s \models \neg\langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle q$
$\to [(M, s \models p \text{ and } M, s \models \neg\langle a\rangle q) \text{ or } (M, s \models \neg p \text{ and } M, s \models \neg\langle b\rangle q)]$,

(8) $M, s \models \neg\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q$
$\to [(M, s \models \neg p \text{ and } M, s \models \neg q) \text{ or}$
$(M, s \models p \text{ and } M, s \models \neg\langle a\rangle\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q)]$,

(9) $M, s \models \neg\langle p?\rangle q \to (M, s \models \neg p \text{ or } M, s \models \neg q)$,

(10) $M, s \models \langle A\rangle q \to \exists t\, ((s, t) \in \rho(A) \text{ and } M, t \models q)$,

(11) $M, s \models \neg\langle A\rangle q \to \forall t\, ((s, t) \in \rho(A) \to M, t \models \neg q)$,

(12) $M, s \models \langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle q$
$\to \exists t\, ((s, t) \in \rho(\textbf{while } p \textbf{ do } a \textbf{ od}) \text{ and } M, t \models q)$.

Using Lemmas 2.8 and 2.9, we can prove the following result, which is just a modification of [24, Lemma 1] (cf. [1, Lemma 3.7]).

3.4. Lemma. *A model for* p_0 *is a tableau for* p_0. *If* M *is a tableau for* p_0, *then there is a model* M' *for* p_0 *such that* M *and* M' *have isomorphic graphs.*

Proof. The first half of the lemma immediately follows from the definitions. For the second half we need the following lemma.

3.4.1. Sublemma. *Let $M=(S, \pi, \rho)$ be a tableau for p_0, and $\langle a\rangle p \in \mathrm{FL}_s(p_0)$. Then*
(a) *if $M, s \models \langle a\rangle p$, then $\exists t\,((s,t)\in\rho(a)$ and $M, t\models p)$,*
(b) *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 Definition 3.3(8) 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 $M, s_k\models q$. □

Proof of Lemma 3.4 (*continued*). 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 $\pi': \Phi_s\rightarrow\mathcal{P}(S)$ and $\rho': \Sigma_s\rightarrow\mathcal{P}(S\times S)$ so that the model constraints are satisfied. Let $M'=(S, \pi', \rho')$. Then we can show by induction on the size of formula p and program a, using Sublemma 3.4.1, that
(1) if $\langle a\rangle q\in\mathrm{FL}_s(p_0)$, then $(s,t)\in\rho(a)\rightarrow(s,t)\in\rho'(a)$,
(2) if $\langle a\rangle q\in\mathrm{FL}_s(p_0)$ and $M, s\models\neg\langle a\rangle q$, then $(s,t)\in\rho(a)$ iff $(s,t)\in\rho'(a)$,
(3) if $\neg p\in\mathrm{FL}_s(p_0)$, then $M, s\models p\rightarrow M', s\models p$, and $M, s\models\neg p\rightarrow M', s\models\neg p$.
Thus it follows that M' is a model for p_0 whose graph is isomorphic to that of M. □

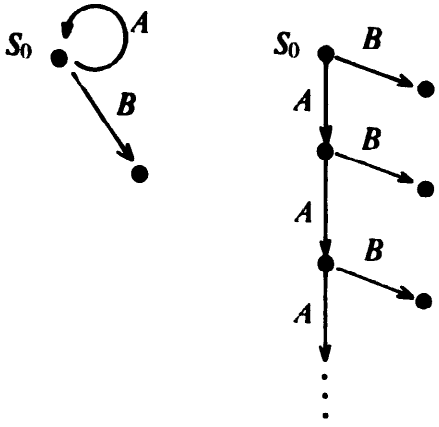
4. Tree models

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. Details are omitted 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 by the left-hand side diagram below. Then M can be converted to the tree model for p_0 shown on the right-hand side:



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

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

(1) *if $p_0 \in \mathrm{SDPDL}_0$, then p_0 is satisfiable in a tree model consisting of a single branch of depth $\leq n$,*

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

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

4.5. Definition. A *trajectory* in $M = (S, \pi, \rho)$ is a finite sequence of states. If α is the trajectory $(s_0, \ldots, s_k)$, then the *length* of α, written $|\alpha|$, is k.

Given $M = (S, \pi, \rho)$ and $a \in \Sigma_0$, we define $\tau_M(a)$, the set of *a-trajectories* in M, by induction on the structure of a (cf. [24, p. 328; 1, Section 4.2]) (we again omit the subscript M if it is clear from context):

(1) $\tau(A) = \rho(A)$,

(2) $\tau(a;b) = \tau(a) \circ \tau(b)$
$= \{(s, \ldots, u, \ldots, t) \mid (s, \ldots, u) \in \tau(a) \text{ and } (u, \ldots, t) \in \tau(b)\}$,

(3) τ (**if** p **then** a **else** b **fi**) =
$= \{(s, \ldots, t) \mid M, s \vdash p \text{ and } (s, \ldots, t) \in \tau(a)\} \cup \{(s, \ldots, t) \mid M, s \models \neg p$
and $(s, \ldots, t) \in \tau(b)\}$,

(4) τ (**while** p **do** a **od**) $= \bigcup_{i \geq 0} \tau((p?;a)^i;\neg p?)$,

(5) $\tau(p?) = \{(s) \mid M, s \models p\}$.

Note that if M is a DPDL structure, then $(s, t) \in \rho_M(a)$ iff there exists a (necessarily unique) trajectory $(s_0, \ldots, s_k) \in \tau_M(a)$ with $s = s_0$ and $t = s_k$.

We will say that a formula $\langle a \rangle q$ is *fulfilled* in a trajectory $\alpha = (s_0, \ldots, s_k)$ if $\alpha \in \tau(a)$ and $M, s_k \models q$.

4.6. Definition. 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_Σ.

For typographical reasons we use the notation $(n_0, A_0, n_1, A_1, \ldots)$ for a straight line graph, where $n_i \subseteq \Phi_\Sigma$ 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 $|n_j| = \sum_{p \in n_j} |p|$.

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, as follows:

(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) $L^* = \{(\emptyset)\} \cup (\bigcup_{i \geq 1} L^i)$,

(4) $L^\omega = \{g \mid g = g_1 g_2 g_3 \cdots g_k, g_i \in L, \text{ for all } i < k, |g_i| < \infty, \text{ and } |g_k| = \infty\}$
$\cup \{g \mid g = g_1 g_2 g_3 \cdots \text{ for all } i, g_i \in L \text{ and } |g_i| < \infty\}$

(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 $\Sigma_\Sigma \cup \Phi_\Sigma$ we can inductively define a corresponding set of straight line paths:

(1) $L_\Lambda = \{(\emptyset, \Lambda, \emptyset)\}$,

(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) $L_q = \{(\{q\})\}$ for $q = P, \neg P$, *true*, *false* or $\neg\langle A\rangle$*true*,

(7) $L_{\neg\neg p} = L_p$,

(8) $L_{\langle a;b\rangle p} = L_a \cdot L_{\langle b\rangle p}$,

(9) $L_{\langle a\cup b\rangle p} = L_{\langle a\rangle p} \cup L_{\langle b\rangle p}$,

(10) $L_{\langle a;b\rangle \text{true}} = L_{\langle a\rangle \text{true}} \cup L_a \cdot L_{\langle b\rangle \text{true}}$,

(11) $L_{\langle \textbf{if } p \textbf{ then } a \textbf{ else } b \textbf{ fi}\rangle \text{true}} = L_{p?} \cdot L_{\langle a\rangle \text{true}} \cup L_{\neg p?} \cdot L_{\langle b\rangle \text{true}}$,

(12) $L_{\langle \textbf{while } p \textbf{ do } a \textbf{ od}\rangle \text{true}} = (L_{p?} \cdot L_a)^* \cdot L_{p?} \cdot L_{\langle a\rangle \text{true}} \cup (L_{p?} \cdot L_a)^\omega$,

(13) $L_{\langle p?\rangle \text{true}} = L_{p?}$.

By comparing the inductive definitions given above with those in Definition 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 state s 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 constraints which have to be met in a model for p. These notions are made more precise in Lemmas 4.9 and 4.10 below.

4.8. Definition. The straight line path $g = (n_0, A_0, n_1, \ldots, A_{k-1}, n_k)$ and the trajectory $\alpha = (s_0, s_1, \ldots)$ are *consistent* 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 $p \in n_i$, then $M, s_i \models p$.

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

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

(b) *If M is 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 $\neg\langle \textbf{while } p \textbf{ do } a \textbf{ od} \rangle \mathit{true}$. 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 $(s, t) \in \rho(\textbf{while } p \textbf{ do } a \textbf{ od})$ 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

$$(L_p? \cdot L_a)^\omega \cup (L_p? \cdot L_a)^* \cdot L_{\neg\langle a \rangle \mathit{true}} = L_{\neg\langle \textbf{while } p \textbf{ do } a \textbf{ od} \rangle \mathit{true}}.$$

(iii) Now we prove (b) by induction on the size of p. It is trivial if p is an atomic formula or its negation. 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 Definition 4.7(9). □

4.10. Lemma. *Suppose $g = (n_0, A_0, n_1, A_1, \ldots) \in L_p$. Then*

(a) *if* $p \in \mathrm{SDPDL}_0$, *then* $|g| < |p|$, *and, for* $j < |g|$, $n_j = \emptyset$, *while* $n_{|g|} = \{q\}$, *where* q *is of the form* P, $\neg P$, *true, false, or* $\neg\langle A\rangle$*true, and* $P \in \mathrm{FL}_s(p)$,

(b) *if* $p \in \mathrm{SDPDL}_{i+1}$, *then, for* $j < |g|$, $n_j \subseteq \mathrm{SDPDL}_i \cap \mathrm{FL}_s(p)$ *while if* $|g| < \infty$,

$$n_{|g|} \subseteq \mathrm{SDPDL}_i \cap \mathrm{FL}_s(p) \cup \{\neg\langle A\rangle true \mid A \in \Sigma_0\}).$$

Moreover, if n_j *is consistent in the sense that we do not have* $\{q, \neg q\} \subseteq n_j$ *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 $p \in \mathrm{SDPDL}_0$, 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. The idea is to 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**. It is easy to see that $(\{q, \neg q, P\}) \in L_{\textbf{while } q \textbf{ do } P\textbf{? od}}$, and if $|q|$ is too large we would have $(|q| + |\neg q| + |P|) > |\textbf{while } q \textbf{ do } P\textbf{? od}|$. However, $\{q, \neg q, P\}$ is inconsistent, so this example does not contradict our result. □

4.11. Definition. Given a model $M = (S, \pi, \rho)$ and $S' \subseteq S$, by Remark 2.5(1) 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 $M' \leq N \leq M$, we have $N, s \models p$. 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* $p_0 \in \mathrm{SDPDL}_0$, *there is a subtree* M' *of* M *which sets* p_0 *at* s_0 *and consists of exactly one branch of length* $\leq n$,

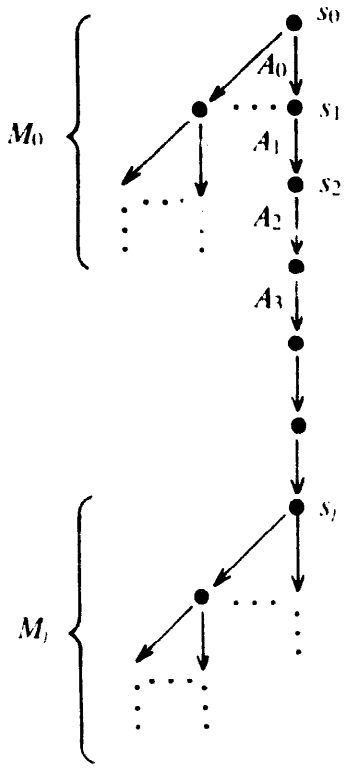
(b) *if* $p_0 \in \mathrm{SDPDL}_i$, $i \geq 1$, *then there is a subtree* M' *of* M *which sets* p_0 *at* s_0 *and has* $\leq n^2 k^{i-1}$ *nodes at depth* k. *Thus* p_0 *is set by a submodel* M' *which has only polynomially many nodes at depth* k, *where the degree of the polynomial depends on the 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 immediately follows 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 that 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 that $|S_j| \leq \sum_{q \in n_j} |S_{q,j}| \leq \sum_{q \in n_j} |q| = |n_j| \leq n$ (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 as s_j of length $< |q|$, M_j is a tree with $\leq n$ nodes and depth $< n$. 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_j hanging off of s_j:



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

$$N_k = \sum_{m=0}^{k} N_{k-m,m} \tag{1}$$

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

$$\begin{aligned} N_k &\leq \sum_{m=0}^{n-1} N_{k-m,m} \quad \text{(where we take } N_{j,m} = 0 \text{ if } j < 0) \\ &\leq n^2. \end{aligned}$$

Thus, at any depth in the tree M' there are less than n^2 nodes. (This result can be improved. It an 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 $M, s \vDash q$, then there is a subtree M' of M which sets q at s and has $\leq |q|^2 k^{i-1}$ nodes at depth k. Now suppose $p \in \mathrm{SDPDL}_{i+1}$, $|p| = n$ and $M, s_0 \vDash p$ 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 $\leq |q|^2 k^{i-1}$ nodes at depth k. Thus the number of nodes of M_j at depth k is

$$\begin{aligned} N_{j,k} &\leq \sum_{q \in n_j} |q|^2 k^{i-1} \leq \left(\sum_{q \in n_j} |q|^2 \right) k^{i-1} \leq \left(\sum_{q \in n_j} |q| \right)^2 k^{i-1} \\ &\leq n^2 k^{i-1} \quad \text{(using Lemma 4.10(b)).} \end{aligned}$$

Since (1) above still holds, we have

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

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

For formulas in $\mathrm{SDPDL}_{\mathrm{pt}}$ we can do even better: we can always find a finite tree model. Intuitively, this is because if we only have propositional tests, we can chop off all the infinite branches of the tree we generated above. We are aiming for the following.

4.13. Theorem. *Let p_0 be a satisfiable* $\mathrm{SDPDL}_{\mathrm{pt}}$ *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 $\langle a_1\rangle \cdots \langle a_m\rangle q$ and $g = f_1 \cdots f_m h$ with $f_j \in L_{a_j}$ for $j \leq m$, $h \in L_q$ and $\sum_{j \leq m} |f_j| = N \geq 1$, then, for all $i < N$, there is a formula p_i of the form $\langle A_i\rangle\langle b_1\rangle \cdots \langle b_k\rangle q \in \mathrm{FL}_s(p)$, such that $(\emptyset, A_i, n_{i+1}, A_{i+1}, \ldots) \in L_{p_i}$ and $g \in (n_0, A_0, \ldots, A_{i-1}, n_i) \cdot L_{p_i} \subseteq L_p$. In fact we have $(n_0, A_0, \ldots, A_{i-1}, n_i) \cdot L_{A_1} \cdot L_{b_1} \cdot \ldots \cdot L_{b_k} \subseteq L_{a_1} \cdot \ldots \cdot L_{a_m}$.*

(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 $\mathrm{FL}_s(p)$ 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, $\neg P$, true, false, or $\neg\langle b\rangle$true such that $\models p' \to p$, $L_{p'} \subseteq L_p$ 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 $\models p' \to p$, $L_{p'} \subseteq L_p$, and $g \in L_{p'}$.*

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

Proof. (a) Fixing $m = 1$ and $i = 0$, we prove the result by induction on the structure of a_1. Clearly there is no problem if a_1 is a primitive program, 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 $|f_1| = 0$ and the statement is vacuously true. If $a_1 = b;c$, there exist $g_1 \in L_b$, $g_2 \in L_c$ with $f_1 = g_1 g_2$. If $|g_1| > 0$, we are done by the induction hypothesis applied to $\langle b\rangle(\langle c\rangle q) \in \mathrm{FL}_s(p)$. If $|g_1| = 0$, suppose $g_1 = (n_0')$. Then $|g_2| > 0$, $g_2 h = (n_0'', A_0, n_1, \ldots) \in L_{\langle c\rangle q}$ with $(n_0' \cup n_0'') = n_0$. Now we are done by applying the induction hypothesis to $\langle c\rangle q \in \mathrm{FL}_s(p)$.

Finally, if a is of the form **while** r **do** b **od**, then by Definition 4.7(5), $f_1 = h_1 \cdots h_{2j+1}$, where for all $j' \leq j$ we have $h_{2j'-1} = (\{r\})$ and $h_{2j'} \in L_b$ while $h_{2j+1} = (\{\neg r\})$. Thus, since $|f| > 0$ (recall $m = 1$), $f_1 = g_1 g_2 g_3$, where $|g_1| = 0$, $g_2 \in L_b$, $|g_2| > 0$ and $g_3 \in L_{\text{while } r \text{ do } b \text{ od}}$. Again we can suppose $g_1 = (n_0')$, and $g_2 g_3 h = (n_0'', A_0, n_1, \ldots) \in L_{\langle b\rangle\langle \text{while } r \text{ do } b \text{ od}\rangle q}$, with $n_0' \cup n_0'' = n_0$. Since $|g_2| > 0$, we can apply the induction hypothesis to $\langle b\rangle\langle$**while** r **do** b **od**$\rangle q \in \mathrm{FL}_s(p)$ and we are done.

Keeping i fixed at 0, we now extend the result to all m by induction. If $|f_1| > 0$, exactly the same argument as given above works again, while if $|f_1| = 0$, techniques similar to ones used above allow us to reduce to considering $\langle a_2\rangle \cdots \langle a_m\rangle q$, in which case the induction hypothesis applies. Details are omitted.

Finally, we show that the result holds for all $i < N$ by induction on i. The base case was handled above. Now suppose the result holds for $i = h$ and $h + 1 < N$. By assumption, there is a formula p_h of the form $\langle A_h\rangle\langle b_1\rangle \cdots \langle b_k\rangle q$ such that $g \in (n_0, A_0, \ldots, A_{h-1}, n_h) \cdot L_{p_h} \subseteq L_p$. From this it follows that

$$(n_0, A_0, \ldots, n_h, A_h, \emptyset) \cdot L_{\langle b_1\rangle \cdots \langle b_k\rangle q} \subseteq L_p \quad \text{and} \quad (n_{h+1}, A_{h+1}, \ldots) \in L_{\langle b_1\rangle \cdots \langle b_k\rangle q}.$$

Thus, we can just repeat the argument given above for the case $i=0$ for $p' = \langle b_1\rangle \cdots \langle b_k\rangle q \in \mathrm{FL}_s(p)$ to get p_{h+1}.

The first half of part (b) follows by a straightforward induction on the structure of programs, using Definition 4.7(2), (7)–(9) and Lemma 2.9(1), (5), (12) and (13). The second half immediately follows from the first.

Part (c) is proved by a similar induction on the structure of programs, this time using Definition 4.7(2), (9)–(11) and Lemma 2.9(1), (5)–(7).

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

$$\text{if } |f|<\infty, \text{ then } f \in L_{\langle a\rangle true} \text{ iff there is an } h \in L_a \text{ with } |h|>|f| \text{ such that } h=(m_0, B_0, m_1, B_1, \ldots, B_k, m_k), \text{ and for some } i<k \text{ we have } f=(m_0, B_0, \ldots, B_{i-1}, m_i)\cdot(\{\neg\langle B_i\rangle true\}). \tag{2}$$

Intuitively, (2) says that the finite length elements of $L_{\langle a\rangle true}$ are essentially prefixes of elements in L_a. (Note that elements in L_a are always of finite length.) As usual, (2) is proved by a straightforward induction on the structure of programs. Details are omitted here.

Returning to the proof of part (d), if $|g|<\infty$ the result immediately follows from (2). If $|g|=\infty$, then, by part (c), there exists a formula p' such that $g \in L_{p'} \subseteq L_{p'}$, $\models p' \to p$ and p' is of the form $\neg\{\langle c\rangle\}\langle \textbf{while } q \textbf{ do } b \textbf{ od}\rangle true$. (We use the { } notation to indicate that the object between the curly brackets may or may not occur in the expression.) Hence $g \in \{L_c \cdot\}(L_{q?} \cdot L_b)^\omega$. Given i, there exist $j>i$ and $k \geq 0$ such that $(n_0, A_0, \ldots, A_{j-1}, n_j) \in \{L_c \cdot\}(L_{q?} \cdot L_b)^k$. It then follows that

$$(n_0, A_0, n_1, \ldots, A_{j-1}, n_j)\cdot(\{\neg q\}) \in \{L_c \cdot\}(L_{q?} \cdot L_b)^k \cdot L_{\neg q?} \subseteq L_c \cdot L_{\textbf{while } q \textbf{ do } b \textbf{ od}}.$$

Thus, by (2),

$$(n_0, A_0, \ldots, A_{i-1}, n_i)\cdot(\{\neg\langle A_i\rangle true\}) \in L_{\{c\}\langle \textbf{while } q \textbf{ do } b \textbf{ od}\rangle true} \subseteq L_{p'}. \qquad \square$$

4.15. Remark. The formulas p_i of Lemma 4.14(a) correspond to the derivatives of [1, Definition 4.4]. It should also be noted that, with a little more work, a more general form of Lemma 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

$$g \in (n_0, A_0, \ldots, A_{i-1}, n_i)\cdot L_{p_i} \subseteq L_p \quad \text{and} \quad (0, A_i, n_{i+1}, A_{i+1}, \ldots) \in L_{p_i}.$$

4.16. Definition. A formula of $\mathrm{SDPDL}_{\mathrm{pt}}$ is said to be *elementary* if it is of the form $\langle a_1\rangle \cdots \langle a_i\rangle q$, $i \geq 0$, where q is of the form P, $\neg P$, *true*, *false*, or $\neg\langle b\rangle true$.

The notion of straight line path is well-defined for elementary formulas; Definition 4.7 carries over directly. (Unfortunately, for general $\mathrm{SDPDL}_{\mathrm{pt}}$ 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 the following lemma.

4.17. Lemma. *Let $p \in \mathrm{SDPDL}_{pl}$ 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 $j < |g|$ the formulas in n_j are Boolean combinations of primitive formulas, while if $|g| < \infty$ the formulas in $n_{|g|}$ are Boolean combinations of primitive formulas or of the form true, false or $\neg\langle A\rangle true$.*

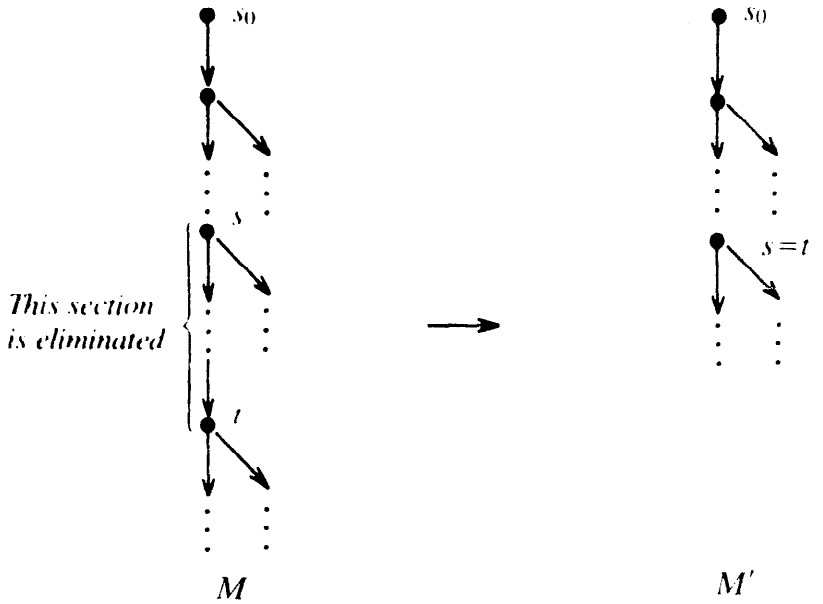
Note that part (b) is analogous to Lemma 4.10. Since the only tests in SDPDL_{pl} formulas are Boolean combinations of primitive formulas, these are the only formulas that appear in n_j for $j < |g|$.

Proof. Part (a) easily follows by induction on the size of formulas, using Lemma 2.9(5), (10)–(13). Part (b) is similar to Lemma 4.10, and the proof is also left to the reader. □

We need one more lemma before returning to the proof of Theorem 4.13. This lemma will give us the tools to truncate a tree model.

4.18. Definition. Given a structure $M = (S, \pi, \rho)$ and a set of formulas H, two states $s, t \in S$ are said to be *H-equivalent*, written $s \equiv_H t$, iff for all $q \in H$, $M, s \models q$ iff $M, t \models q$.

4.19. Lemma. *Let $M = (S, \pi, \rho)$ be a tree model with p true at the root, and suppose, for some states s, t such that t is a descendant of s on the tree, we have $s \equiv_{\mathrm{FL}(p)} t$. Let $M' = (S', \pi', \rho')$ be the structure whose graph is obtained from that of M by eliminating all descendants of s which are not also descendants of t and then identifying s and t, as shown in the diagram below:*



Define π' so that for $u \in S'$,

$$M', u \models q \quad \textit{iff} \quad q \in \mathrm{FL}_s(p) \textit{ and } M, u \models q.$$

Then M' is tableau for p.

Proof. All the tableau conditions except Definition 3.3(12) immediately follow from the construction of M'. Now suppose $M', s_0 \models \langle \textbf{while } r \textbf{ do } a \textbf{ od} \rangle q$ (the case for states other than s_0 is identical). Then by definition of π' we also have $M, s_0 \models \langle \textbf{while } r \textbf{ do } a \textbf{ od} \rangle q$. Thus there is a path $g = (n_0, A_0, \ldots, A_k, n_k) \in L_{\textbf{while } r \textbf{ do } a \textbf{ od}}$ which is consistent with some trajectory $(s_0, \ldots, s_k) \in \tau_M(\textbf{while } r \textbf{ do } a \textbf{ od})$ and $M, s_k \models q$. If this trajectory does not pass through s, then it also exists in M' and we are done. Otherwise, suppose $s = s_i$. Then by Lemma 4.14(a) there is a formula $q_i \in \mathrm{FL}_s(\langle \textbf{while } r \textbf{ do } a \textbf{ od} \rangle q) \subseteq \mathrm{FL}_s(p)$ such that q_i is of the form $\langle A_i \rangle \langle b_1 \rangle \cdots \langle b_h \rangle q$, where

$$(n_0, A_0, \ldots, A_{i-1}, n_i) \cdot L_{A_i} \cdot L_{b_1} \cdot \ldots \cdot L_{b_h} \subseteq L_{\textbf{while } r \textbf{ do } a \textbf{ od}}$$

and

$$(\emptyset, A_i, n_{i+1}, \ldots) \in L_{q_i}.$$

Again, in words, q_i has the property that continuing from $(s_0, \ldots, s_i)$ with *any* path in L_{q_i} gives us a path in L_q starting with s_0.

Since $M, s_i \models q_i$ and $s_i = s \equiv_{\mathrm{FL}_s(p)} t$, we also have $M', t \models q_i$. But M and M' are isomorphic below t, so there is a trajectory in both M and M' consistent with some $g' \in L_{A_i} \cdot L_{b_1} \cdot \ldots \cdot L_{b_h}$ such that if u is the last state in the trajectory, then $M', u \models q$. But $(n_0, A_0, \ldots, A_{i-1}, n_i) \cdot g' \in L_{\textbf{while } p \textbf{ do } a \textbf{ od}}$, thus there is a trajectory in M' starting with s_0 and ending with u consistent with some path in $L_{\textbf{while } r \textbf{ do } a \textbf{ od}}$. By Lemma 4.9(a) it follows that $(s_0, u) \in \rho'(\textbf{while } r \textbf{ do } a \textbf{ od})$, giving us our result. □

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 $M, s_0 \models q_1 \wedge \cdots \wedge q_k$, where $(q_1 \wedge \cdots \wedge q_k)$ is one of the disjuncts of p_1. Note $\models (q_1 \wedge \cdots \wedge q_k) \rightarrow p_0$, 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 $q_1 \wedge \cdots \wedge q_k$, and hence for p_0. This model has $\leq n$ branches, but they are not necessarily finite. To show that we can actually construct a finite model we use Lemma 4.14(b), (d). Lemma 4.14(b) says that if there is an infinite branch, it must essentially be due to a formula of the form $\neg\langle a \rangle true$. Lemma 4.14(d) says that we can truncate this branch and still get a model that satifies this formula. In more detail we proceed as follows:

If $|g_j| = \infty$, then, by Lemma 4.14(b), there is a formula r_j of the form $\neg\langle b \rangle a$ or $\langle a \rangle \neg\langle b \rangle q$, such that $\models r_j \rightarrow q_j$, $L_{r_j} \subseteq L_{q_j}$ and $g_j \in L_{r_j}$. So if $|g_j| = \infty$, then g_j is of the form $f_j h_j$, where $h_j \in L_{(\ldots)^{\omega}}$ and $f_j \in L_a$ or $f_j = (\emptyset)$; in either case $|f_j| < \infty$. We now

want to truncate the infinite h_i's without affecting the finite paths. To this end, let

$$N = \max(\{|g_i|: |g_i| < \infty\} \cup \{|f_i|: |g_i| = \infty\}).$$

Then if $|g_i| = \infty$ and $g_i = (r_0, A_0, n_1, A_1, \ldots)$, let

$$g_i' = (n_0, A_0, n_1, \ldots, A_{n-1}, n_N) \cdot (\{\neg\langle A_N\rangle true\}).$$

By Lemma 4.14(d), $g_i' \in L_{r_i}$. Truncate the infinite trajectories consistent with those g_i such that $|g_i| = \infty$ to finite trajectories of length N. Note the truncated trajectories are now consistent with g_i'. Let S' be the union of the states in the truncated trajectories, and let M' be the subtree of M determined by S'. Then, by Lemma 4.17(b), it follows 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 $M', s_0 \models r_1 \wedge \cdots \wedge r_k$, so we also get $M', s_0 \models p_0$.

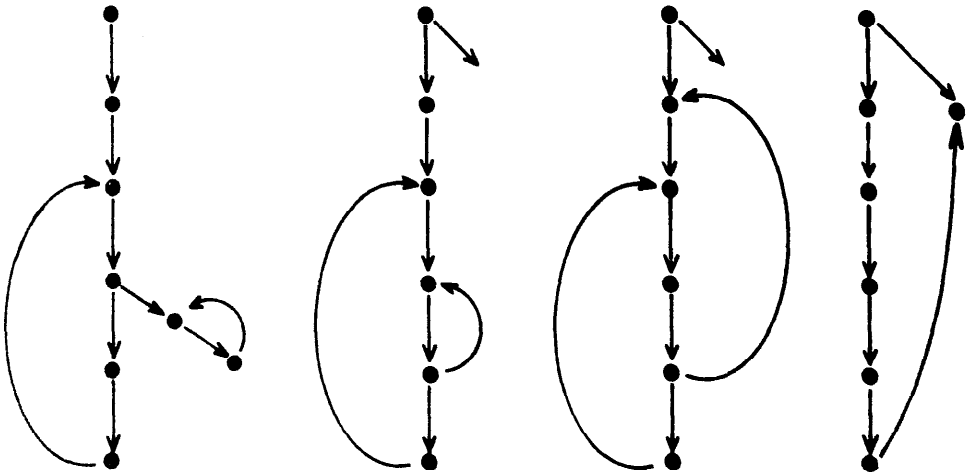
M' is a finite tree model with $\leq n$ branches, but we are still not done since we require a model whose branches have length $\leq 2^n$. Here Lemma 4.19 comes into play.

First note that by systematically replacing subformulas of p_0 of the form $\langle p?\rangle q$ by $p \wedge q$, we can find a formula p_2 of SDPDL such that $|p_2| = n$ and $\models p_0 \equiv p_2$. Now suppose $(s_0, \ldots, s_m)$ is a branch in M' 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 $s_i \equiv_{\mathrm{FL}(p_2)} s_j$, since it is easy to check that there are at most $2^n \equiv_{\mathrm{FL}(p_2)}$ equivalence classes. Let $M' = (S'', \pi'', \rho'')$ be the structure whose graph is obtained from that of M' by identifying s_i and s_j as in Lemma 4.19. Then M'' is a tableau for p_2. By Lemma 3.4 we can convert M'' to a model with a graph isomorphic to that of M''. 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 following SDPDL$_1$ formula: $\neg\langle$**while** $\langle A\rangle true$ **do** A **od**$\rangle true$. It is easily seen to be satisfiable in the following tree model, where $s_i \models \langle A\rangle true$ for all i:

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

4.20. 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 r and s are nodes on the graph of a treelike model and there is a backedge from s to r, then $r < s$ (where '$<$' denotes ancestor of). Moreover, if there is a also a backedge from node u to t, then neither u nor t lie between r and s on the graph. Thus the graph on the left of the diagram below is the graph of a treelike model, while the other three are not:



4.21. Theorem. *If p_0 is a satisfiable* SDPDL *formula, then p_0 is satisfiable in a finite treelike model.*

Before we can prove Theorem 4.21 we need a few more definitions and lemmas, which will also be crucial to the algorithm presented in Section 5.

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

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

4.23. Definitions. Let ϕ be a truth assignment to SDPDL formulas (i.e., $\Phi : \Phi_0 \to \{true, false\}$). For any formula q, we can define formulas $q_{\alpha,\phi}$ and $q_{\beta,\phi}$ by induction as follows. (Roughly speaking, $q_{\alpha,\phi}$ corresponds to the tests in a program, while $q_{\beta,\phi}$ corresponds to the consequence of the tests.)

(1) If q is of the form $\langle A\rangle r$, $\neg\langle A\rangle r$, P, $\neg P$, *true* or *false*, then $q_{\alpha,\phi} = q_{\beta,\phi} = q$.

(2) If q is of the form $\neg\neg r$, then $q_{\alpha,\phi} = q_{\beta,\phi} = r$.

(3) If q is of the form $\langle a ; b\rangle r$, then $q_{\alpha,\phi} = q_{\beta,\phi} = \langle a\rangle\langle b\rangle r$.

(4) If q is of the form $\langle$**if** p **then** a **else** b **fi**$\rangle r$ and $\phi(p) = true$, then $q_{\alpha,\phi} = p$, $q_{\beta,\phi} = \langle a\rangle r$. Otherwise $q_{\alpha,\phi} = \neg p$, $q_{\beta,\phi} = \langle b\rangle r$.

(5) If q is of the form $\langle$**while** p **do** a **od**$\rangle r$, and $\phi(p) = true$, then $q_{\alpha,\phi} = p$, $q_{\beta,\phi} = \langle a\rangle\langle$**while** p **do** a **od**$\rangle r$. Otherwise $q_{\alpha,\phi} = \neg p$, $q_{\beta,\phi} = r$.

(6) If q is of the form $\langle p?\rangle r$, then $q_{\alpha,\phi} = p$, $q_{\beta,\phi} = r$.

(7) If q is of the form $\neg\langle a\,;b\rangle r$, then $q_{\alpha,\phi} = q_{\beta,\phi} = \neg\langle a\rangle\langle b\rangle r$.

(8) If q is of the form $\neg\langle$**if** p **then** a **else** b **fi**$\rangle r$ and $\phi(p) = true$, then $q_{\alpha,\phi} = p$ and $q_{\beta,\phi} = \neg\langle a\rangle r$. Otherwise, $q_{\alpha,\phi} = \neg p$, $q_{\beta,\phi} = \neg\langle b\rangle r$.

(9) If q is of the form $\neg\langle$**while** p **do** a **od**$\rangle r$ and $\phi(p) = true$, then $q_{\alpha,\phi} = p$, $q_{\beta,\phi} = \neg\langle a\rangle\langle$**while** p **do** a **od**$\rangle r$. Otherwise $q_{\alpha,\phi} = \neg p$, $q_{\beta,\phi} = \neg r$.

(10) If $q = \neg\langle p?\rangle r$ and $\phi(p) = true$, then $q_{\alpha,\phi} = p$, $q_{\beta,\phi} = \neg r$. Otherwise $q_{\alpha,\phi} = q_{\beta,\phi} = \neg p$.

Given a set of SDPDL formulas F, let $T(F, \phi)$ be the least set of formulas containing F such that if $q \in T(F, \phi)$, then $q_{\alpha,\phi}, q_{\beta,\phi} \in T(F, \phi)$.

Given a model $M = (S, \pi, \rho)$ and a state $s \in S$, we will say ϕ *agrees with* s if $\phi(q) = true$ iff $M, s \models q$.

4.24. Lemma. *Let* $F \subseteq \mathrm{FL}_s(p_0)$, *where* $|p_0| = n$, *and let* ϕ *be any truth assignment. Then*:

(a) $T(F, \phi) \subseteq \mathrm{FL}_s(p_0)$.

(b) $T(F, \phi) = \bigcup_{i=1}^{2n} F_i$, *where* $F_0 = F$ *and*

$$F_{i+1} = \left\{ q_{\alpha,\phi}, q_{\beta,\phi} \notin \bigcup_{j=1}^{i} F_j \;\middle|\; q \in F_i \right\}$$

$$\cup \{q \in F_i \mid q \text{ is of the form } \langle A\rangle r \text{ or } \neg\langle A\rangle r\}.$$

Moreover, $\|F_{i+1}\| \leq \|F_i\|$.

(c) *If* ϕ *agrees with* s *and* $M, s \models q$ *for all* $q \in F$, *then* $M, s \models q$ *for all* $q \in T(F, \phi)$.

Proof. Parts (a) and (c) are immediate from the definition of $T(F, \phi)$. If $F \subseteq \mathrm{FL}_s(p_0)$, then $|T(F, \phi)| \leq 2n$, so it is easy to see that we must have $F_{2n+1} = F_{2n}$. The first half of part (b) follows from this observation. For the second half, we simply note that both $q_{\alpha,\phi}$ and $q_{\beta,\phi}$ have depth less than or equal to that of q, and if $q_{\alpha,\phi} \neq q_{\beta,\phi}$, then $q_{\alpha,\phi}$ has depth strictly less than that of q. □

4.25. Definition. Given a truth assignment ϕ, define the relation $\Rightarrow$ on formula-truth assignment pairs to be the least reflexive transitive relation such that

$$(q, \phi) \Rightarrow (q_{\beta,\phi}, \phi).$$

Given a tableau $M = (S, \pi, \rho)$ we can similarly define $\Rightarrow$ on formula-state pairs to be the least reflexive transitive relation such that

(a) $(q, s) \Rightarrow (q_{\beta,\phi}, s)$ if $(q, \phi) \Rightarrow (q_{\beta,\phi}, \phi)$ and ϕ agrees with s,

(b) $(\langle A\rangle q, s) \Rightarrow (q, t)$ if $M, s \models \langle A\rangle q$ and $(s, t) \in \rho(A)$.

Note that we deliberately use the same symbol $\Rightarrow$ for both relations to emphasize their similarity. It should be clear from the context which one is meant. Intuitively, the relation $\Rightarrow$ traces the progress of a formula towards getting fulfilled. This is made more precise by the next lemma.

4.26. Lemma. *Let $M = (S, \pi, \rho)$ be a tableau. Let p be a formula of the form $\langle a\rangle q$ such that $M, s \models p$, and let ϕ be a truth assignment that agrees with s.*

(a) *Either $(\langle a\rangle q, \phi) \Rightarrow (q, \phi)$ or $(\langle a\rangle q, \phi) \Rightarrow (r, \phi)$, where r is of the form $\langle A\rangle\langle b_1\rangle \cdots \langle b_k\rangle q$.*

(b) *Let $\alpha = (s_0, \ldots, s_k) \in \tau_M(a)$, $k > 0$ such that $s = s_0$ and $M, s_k \models q$. Let gh be a path with $g = (n_0, A_0, \ldots, A_{k-1}, n_k) \in L_a$, $h \in L_q$, such that g is consistent with α. For $i < k$, let p_i be the formula of the form $\langle A_i\rangle\langle b_1\rangle \cdots \langle b_k\rangle q \in \mathrm{FL}_s(\langle a\rangle q)$ which exists by Lemma 4.14(a). Then $M, s_i \models p_i$ and $(p, s) \Rightarrow (p_i, s_i)$.*

Proof. The proof is very similar to that of Lemma 4.14(a), so we omit further details here. □

Proof of Theorem 4.21. We show by induction on the weight of $F \subseteq \mathrm{FL}_s(p_0)$ that if the conjunction of the formulas in F is satisfiable, then it is satisfiable at the root of a finite treelike model. Since p_0 is satisfiable, this will give us the desired result.

In the case $\|F\| = (0, k)$, we have $F \subseteq \mathrm{FL}_s(p_0) \cap \mathrm{SDPDL}_0$. It then immediately follows from Theorem 4.12(a) that there is a finite tree model satisfying the conjunction of the formulas in F.

In the general case, suppose $\|F\| = (j, k)$, and M is a tree model whose root satisfies the conjunction of the formulas in F. Using M as an oracle, we will construct the graph of a treelike model (as described in Remark 2.5(2)) such that the formulas in F are satisfied at its root. Each node in the graph we construct will correspond to some state in M. A node t will be labelled by a set of formulas, $\mathrm{Lab}(t)$, all of which will be true at the corresponding state. For ease of notation, we identify the weight of a node with the weight of its label, writing $\|t\|$ instead of $\|\mathrm{Lab}(t)\|$. If two nodes t and t' on the graph are joined by an edge labelled A, then $(s, s') \in \rho(A)$ for the corresponding states s and s'.

The root of the graph, t_0, is labelled by $\mathrm{Lab}(t_0) = F$. Let ϕ be a truth assignment which agrees with s_0. By Lemma 4.24(c), all the formulas in $T(F, \phi)$ are true at s_0. Let $A_1, \ldots, A_k$ be the primitive programs mentioned in p_0. For $i = 1, \ldots, k$ let

$$G_i = \{r \mid \langle A_i\rangle r \in T(F, \phi)\}$$
$$\cup \{\neg r \mid \neg\langle A_i\rangle r \in T(F, \phi) \text{ and, for some } r', \langle A_i\rangle r' \in T(F, \phi)\}.$$

Note that if there are no formulas in $T(F, \phi)$ of the form $\langle A_i\rangle r$, then $G_i = \emptyset$. If $G_i \neq \emptyset$, then s_0 must have an A_i successor in M, say s', such that all the formulas of G_i are true at s'. Thus we create an A_i-successor of t_0 on the graph labelled by G_i, which corresponds to s'.

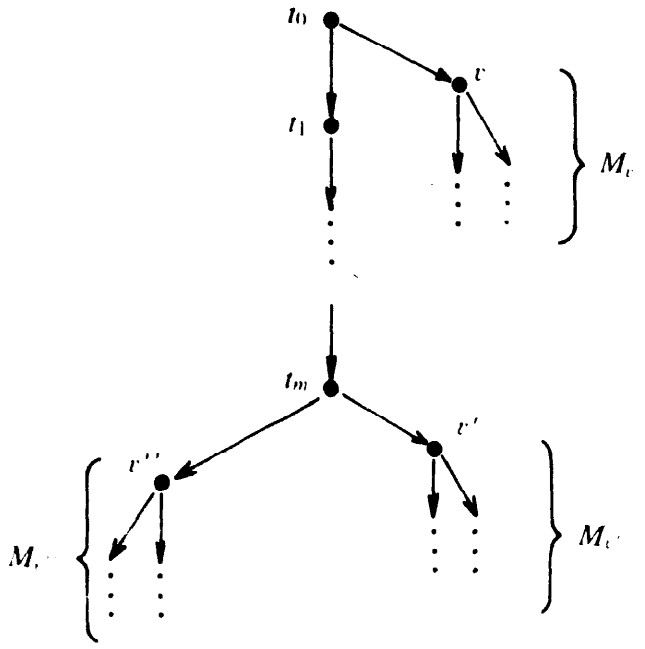
Using Lemma 4.24(b) to obtain the second inequality, we can easily check that

$$\|G_1 \cup \cdots \cup G_k\| \leq \|F_{2n}\| \leq \|F_0\| = \|F\|.$$

Since it is also easy to check that we cannot have $\|G_i\| = \|G_j\| = \|F\|$ if $i \neq j$, it follows that if t_0 has successors on the graph, their weight must be less than or equal to that of t_0, and at most one of the successors can have weight equal to t_0. If t_0 does have successors on the graph, we will call the one of greatest weight t_1. If $\|t_1\| = \|t_0\|$, we repeat the above process with t_1 in place of t_0.

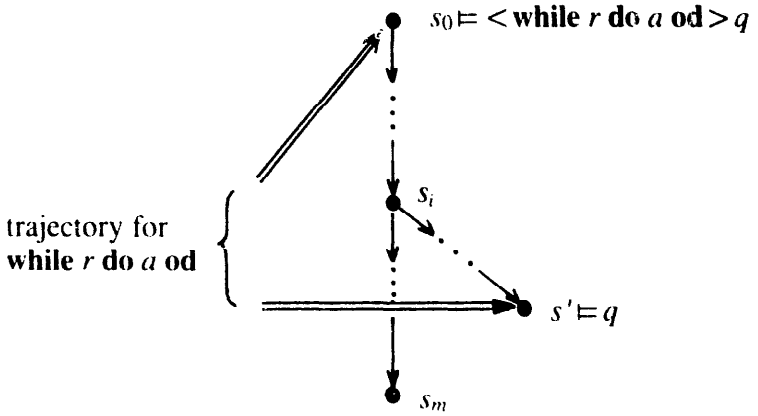
Eventually one of two things will happen. Either the process terminates and we have a finite sequence of nodes $t_0, \ldots, t_m$ such that $\|t_0\| = \cdots = \|t_m\|$, and all other nodes on the graph have lower weight, or there is an infinite sequence of nodes $t_0, t_1, \ldots$, all of which have the same weight, and again all the other nodes on the graph have lower weight. In both cases every node on the graph other than the root must be an immediate successor of one of the t_i's.

In the first case we construct a tableau M' from the graph as follows. The 'backbone' of M' consists of the nodes $t_0, \ldots, t_m$. By the construction, every node v on the graph corresponds to some state u in M, so that if $\mathrm{Lab}(v) = H$, then all the formulas in H are true at u. In particular it follows that all the formulas in H are satisfiable. Moreover, if $v \neq t_j$, $j = 0, \ldots, m$, then $\|H\| < \|F\|$, so, by the induction hypothesis, all the formulas in H are satisfiable at the root of a finite treelike model M_v. By attaching these models at the appropriate plates we get a graph M' with this picture (where we omit labelling the edges or nodes):



To show that M' is indeed a tableau, the only condition that requires checking is Definition 3.3(12). In fact, we only need to show it in the case for the states t_i, $i \leq m$ (the other possibilities are covered by the fact that the M_v's which we have

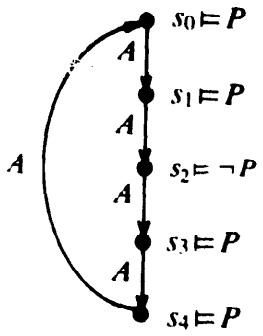
appended are all models). Let $(s_0, \ldots, s_m)$ be the trajectory in M corresponding to $(t_0, \ldots, t_m)$. Suppose $M', t_0 \models p$, where p is of the form $\langle$**while** r **do** a **od**$\rangle q$ (the case for arbitrary t_i is similar). By construction, we must also have $M, s_0 \models p$. Thus there is a **while** r **do** a **od** trajectory in M starting with s_0 and ending with a state s' such that $M, s' \models q$. 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 diagram below:



In the former case we can easily check that the corresponding trajectory in M' is a **while** r **do** a **od** trajectory whose final state satisfies q. In the latter case, suppose $g = (n_0, A_0, \ldots) \in L_{\langle \textbf{while } r \textbf{ do } a \textbf{ od}\rangle q}$ is consistent with the trajectory for $\langle$**while** r **do** r **od**$\rangle q$ starting at s_0. Then we can find a formula p_i of the form $\langle A_i\rangle\langle b_1\rangle \cdots \langle b_k\rangle q$ with all the properties stipulated in Lemmas 4.14(a) and 4.26(b). It is straightforward to check that $n_j \subseteq \mathrm{Lab}(t_j)$ for $j \leq i$, and that $p_i \in \mathrm{Lab}(t_i)$. Since the trajectory for **while** r **do** a **od** branches away from the backbone at s_i, it must be the case that $(s_i, s_{i+1}) \notin \rho(A)$, and hence t_i and t_{i+1} are not connected by an A-edge. Thus $\langle b_1\rangle \cdots \langle b_k\rangle q \in \mathrm{Lab}(v)$ where v is not on the backbone, and so is satisfied at the root of M_v by construction. Thus there is a trajectory fulfilling p_i in M' starting at t_i, and hence also a trajectory fulfilling p in M' starting at t_0. Thus we have shown that M' is a tableau, which, by Lemma 3.4, can be converted to a treelike model.

Now we must deal with the second case. Just as above, we can use the graph to construct a treelike tableau whose backbone will be the infinite sequence $t_0, t_1, \ldots$, where to each state v on the graph with weight $< \|t_0\|$, we append a finite treelike model M_v such that all the formulas in $\mathrm{Lab}(v)$ are true at its root. As in Theorem 4.21, we can define an equivalence relation among the t_i so that $t_i \equiv t_j$ exactly if $\mathrm{Lab}(t_i) = \mathrm{Lab}(t_j)$. Since there are only finitely many equivalence classes (at most $2^{|cl(p_0)|}$), 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.

The problem is best exemplified by a model M with the following treelike structure:



Suppose p_0 is the formula $\langle$**while** P **do** $A; A$ **od**$\rangle$*true*, and $M, s_2 \vDash \neg P$ while, for $i \neq 2$, $M, s_i \vDash P$. It is easy to check that $M, s_0 \vDash p_0$, and that $s_1 \equiv_{\mathrm{FL}(p_0)} s_3$. However, if we identify s_1 and s_3 as in Lemma 4.19, we no longer have a tableau for p_0 since there is no trajectory fulfilling **while** P **do** $A; A$ **od** starting from s_0. In Lemma 4.19 we considered only tree models, so we could count on there being a trajectory below s_3 which, when appended to the trajectory from s_0 to s_1, would fulfill $\langle$**while** P **do** $A; A$ **od**$\rangle q$. Here, because of the backedge this is no longer the case.

Our strategy will be to retain enough states to ensure that all the formulas in $\mathrm{Lab}(t_{i_0})$ of the form $\langle a \rangle q$ get fulfilled.

Let $q_1, \ldots, q_m$ be all the formulas in $\mathrm{Lab}(t_{i_0})$ of the form $\langle a_1 \rangle \cdots \langle a_h \rangle q$, where q is not of the form $\langle b \rangle r$. We know that $M, s_{i_0} \vDash q_1 \wedge \cdots \wedge q_m$. 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 s_{i_1} such that $(s_{i_0}, \ldots, s_{i_1}) \in \tau(a_1; \cdots; a_h)$ and $M', s_{i_1} \vDash q$ or the $\tau(a_1; \cdots; a_h)$ 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 $s_{i_2}, \ldots, s_{i_m}$. We can assume without loss of generality that $i_1 \leq i_2 \leq \cdots \leq i_m$. 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 (see the diagram at the top of p. 152).

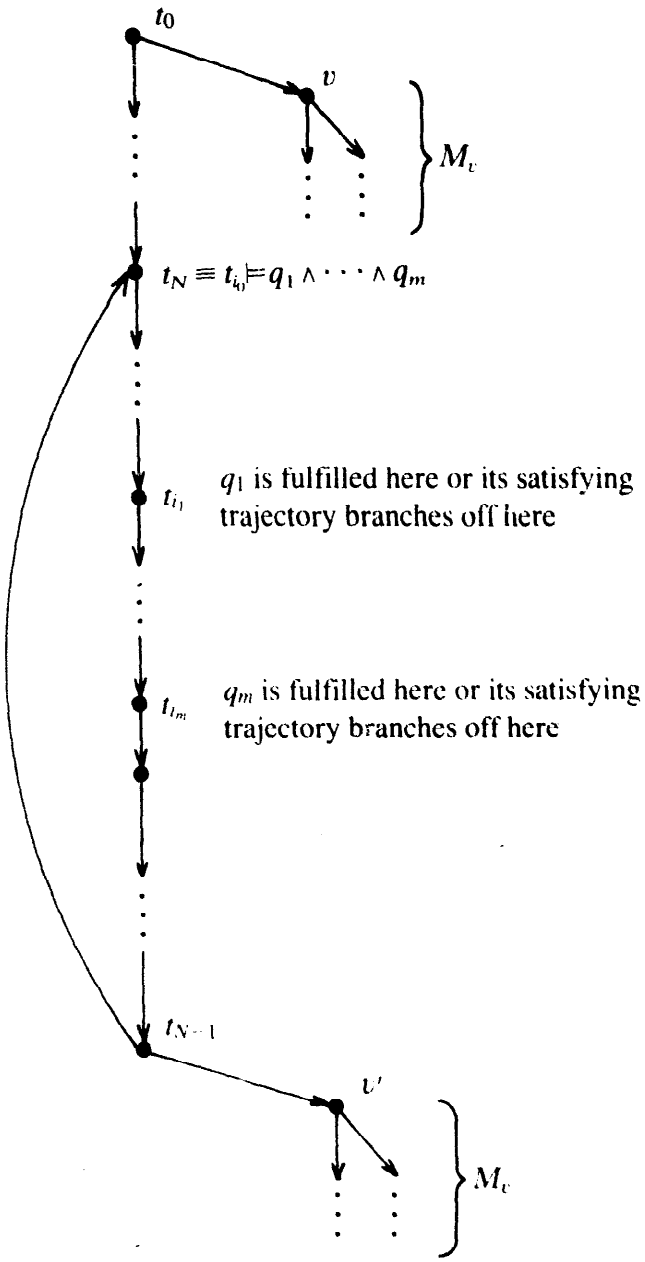
To prove that M'' is a tableau, just as above we must only check Definition 3.3(12) in the case M'', $t_i \vDash \langle$**while** p **do** a **od**$\rangle q$. The only problem arises if the (**while** p **do** a **od**) trajectory in M' starting at t_i goes through t_N. But in this case, just as above there will be a formula r in $\mathrm{Lab}(t_N)$ of the form $\langle a_1 \rangle \cdots \langle a_h \rangle q$ such that

$$(\langle \textbf{while } p \textbf{ do } a \textbf{ od} \rangle q, s_{i_0}) \Rightarrow (r, s_N)$$

and

$$(n_1, B_1, \ldots, B_{n-1}, n_N) \cdot L_r \subseteq L_{\langle \textbf{while } p \textbf{ do } a \textbf{ od} \rangle q}.$$

But $t_N \equiv t_{i_0}$, so $r \in \mathrm{Lab}(t_{i_0})$. By the same argument as we used in the first case above, we can show that there will be an $a_1; \cdots; a_h$-trajectory starting at t_{i_0} fulfilling r, so we are done. □



4.27. Remark. With a little more work we can show that if p_0 has test depth i, then the treelike model we constructed above has $\leq n2^{2m}$ nodes. We now must take the stronger inductive hypothesis that if $\|F\| = (j, k)$, then F can be satisfied in a treelike model of size $\leq n(k+1)2^{2m}$. We leave details to the interested reader.

5. SDPDL is polynomial space complete

5.1. Theorem. *There is a deterministic algorithm which runs in polynomial space to decide if an* SDPDL *formula is satisfiable.*

Proof. Let p_0 be an SDPDL formula, $|p_0| = n$. 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 for p_0 guaranteed to exist by Theorem 4.21, 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 4.21, which says that such a model exists iff p_0 is satisfiable, to establish the correctness of the algorithm. Finally, we use the result of Savitch [28] 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 $\mathrm{FL}_s(p_0)$ analogous to the F of Theorem 4.21. There will also be polynomially many other such subsets on a pushdown stack, waiting their turn to be worked on. As in Theorem 4.21, these subsets correspond to (the labels of) 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 (possibly) associated one other subset which we call the *backpointer*. If there is an associated backpointer, then some formulas in the backpointer will be associated with formulas in the first subset. Intuitively, the backpointer corresponds to the state t_w in the second half of the proof of Theorem 4.21, i.e., the state to which a backedge (if there is one) will go. (Remember that we are constructing a *treelike* tableau.) Thus the backpointer must have the same weight (in the sense of Definition 4.22) as the first subset. If the first subset corresponds to state s on the tableau while the backpointer corresponds to state t, then formula q in the backpointer will be associated with formula q' in the first subset precisely if $(q, t) \Rightarrow (q', s)$. Thus we will be able to keep track of the progress of formulas in the backpointer towards getting fulfilled. We will be able to branch back in our construction exactly when the backpointer is identical to the first set and there are no further formulas waiting to be fulfilled.

Initially we work on $\{p_0\}$; the stack is empty. Suppose at a certain time the algorithm is working on the set $F = \{q_1, \ldots, q_k\} \subseteq \mathrm{FL}_s(p_0)$. We nondeterministically guess a truth assignment ϕ and compute $T(F, \phi)$ as in Lemma 4.24(a). (Note we only need to guess the value of ϕ on the formulas of $\mathrm{FL}_s(p_0)$.) Then:

Step 1. If $\{r, \neg r\} \subseteq T(F, \phi)$ for some formula r, we terminate with "unsatisfiable" (the particular sequences of choices made by the algorithm was bad).

Step 2. For $\Rightarrow$ given as in Definition 4.25, we check that Lemma 4.26(b) holds. That is, for $q \in T(F, \phi)$ of the form $\langle a \rangle r$, we check that either $(q, \phi) \Rightarrow (r, \phi)$ or there exists r' of the form $\langle A \rangle \langle b_1 \rangle \cdots \langle b_m \rangle r$ such that $(q, \phi) \Rightarrow (r', \phi)$. If not, we terminate with "unsatisfiable".

Suppose that the primitive programs mentioned in p_0 are $A_1, \ldots, A_k$. We then form the sets $G_1, \ldots, G_k$ just as in Theorem 4.21. If there is no backpointer

associated with the original set F, we can nondeterministically choose to associate F with a copy of itself as a backpointer, in which case we associate every formula in the copy of the form $\langle a_1\rangle \cdots \langle a_k\rangle q$ with the same formula in F.

Step 3. If F has no associated backpointer, we immediately go to Step 4. Otherwise, we must check that one of the G_j's has weight equal to $\|F\|$. (By the equalities of Theorem 4.21, we know there can be at most one.) If not, we terminate with "unsatisfiable". Otherwise we associate the backpointer H with G_j, where $\|G_j\| = \|F\|$ $(= \|H\|)$. If the formula $p_1 \in H$ is associated with $p_2 \in F$, and $(p_2, \phi) \Rightarrow (\langle A_j\rangle r, \phi)$ (for the ϕ chosen above), we associate p_1 with $r \in G_j$. If $G_j = H$ and none of the formulas in H are associated with a formula in G_j, we set G_j to the empty set and forget about H. (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.)

Step 4. If all of $G_1, \ldots, G_k$ 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; we terminate and return "satisfiable".

Step 5. If some of the G_j's are nonempty, we work next on the one of lowest weight, putting all the rest on the stack. If there is more than one G_j of lowest weight, we choose one of these arbitrarily to work on next.

(Note that in Theorem 4.21 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 a sequence of choices made by the algorithm returns "satisfiable", the proof of Theorem 4.21 shows that we have indeed constructed a treelike tableau for p_0, so p_0 really is satisfiable. Conversely, if $p_0 \in \mathrm{SDPDL}_t$ is satisfiable, we know by Theorem 4.21 that there is a finite treelike tableau for p_0. If the algorithm makes guesses corresponding to the state of affairs on this tableau, it will output "satisfiable".

Finally, we must check that the algorithm works in polynomial space. To see this, let j_x be the number of formulas in $\mathrm{FL}_x(p_0)$ of depth x. Note that $\sum_x j_x \leq 2n$. The only weights attainable by subsets of $\mathrm{FL}_x(p_0)$ 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 $\mathrm{FL}_x(p_0)$. 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 mth highest attainable weight, then there are at most $m(k-1)$ pairs of (set, (backpointer)) on the stack (where k is, as above, the number of primitive programs appearing in p_0). The proof depends crucially on the fact that we always work on the set of lowest weight possible. If any pairs are added to the stack at the end of a step (and at most $k-1$ pairs can be added at any step), then at the next step we must be working on a set of lower weight. Since $k \leq n$ and $m \leq 2n$, there are always $\leq 2n^2$ pairs on the stack. Thus the algorithm only uses polynomial space.

We now sketch a method to improve the space complexity. We first note that instead of putting $G_1, \ldots, G_k$ on the stack (as in Step 3), we can instead put on those formulas in $T(F, \phi) \subseteq \mathrm{FL}_s(p_0)$ of the form $\langle A_i \rangle q$ or $\neg\langle A_i \rangle q$; these in turn can be represented by a bit string of length $|\mathrm{FL}_s(p_0)| \leq 2n$. We can then modify the algorithm so that it deals with these bitstrings, and by arguments similar to those used in the previous paragraph, we can show that when we are working on a set with the mth highest attainable weight, there are at most m such bitstrings on the stack. This modified algorithm (which must also represent $\mathrm{FL}_s(p_0)$ in a space efficient manner, something which we claim without proof can be done) can be shown to run in nondeterministic space $O(n^2)$. By Savitch's Theorem, it runs in deterministic space $O(n^4)$. □

5.2. Remarks. (1) Essentially, the algorithm attempts to construct a treelike model for p_0 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 $\leq k \log_k(n)$. (Of course, the search will not necessarily proceed down the leftmost path; we must guess the appropriate path to follow at all times.) The proof follows by an induction on the height of the tree. By Remark 4.27, if p_0 has a model, it has one of size $< n2^{2m}$, so we can nondeterministically construct the model in polynomial space. 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 $\mathrm{SDPDL}_{\mathrm{pt}}$ formulas, but in this case the proof is easier since, by Theorem 4.13, $\mathrm{SDPDL}_{\mathrm{pt}}$ formulas have finite tree models. This means we do not have to take care of the possibility that a path might branch back to a ancestor, and we do not need backpointers.

(3) A *Ianov scheme* is an uninterpreted deterministic program scheme with only one program variable (cf. [9]). 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 [7], given Ianov schemes a and b we can effectively construct PDL programs, say a' and b', such that a and b are strongly equivalent Ianov schemes iff $\langle a' \rangle Q = \langle b' \rangle Q$, where Q does not appear in either a' or b'. But we can easily show that, without loss of generality, a' and b' are $\mathrm{SDPDL}_{\mathrm{pt}}$ programs. Thus we have a polynomial space procedure for deciding strong equivalence of Ianov schemes.

Although we have assumed that our programs are well-structured (i.e., formed from primitive programs by means of the constructs **while** ... **do** ... **od** and

if ... then ... else), our results would clearly also hold if we viewed our programs as deterministic flowcharts whose instructions consisted of primitive programs and tests. We could then get a deterministic propositional flowchart logic by allowing the programs to appear in the $\langle\ \rangle$ construct (cf. [26]), and prove a polynomial space decision procedure for this logic as well, using much the same techniques as presented here. Indeed, these techniques can also be used to prove polynomial space decision procedures for several other logics of programs including linear time temporal logic (TL) and extended linear time temporal logic. (See [8] and [30] for the syntax and semantics of these languages, as well as further details. Polynomial space results for TL are also presented in [29].) We formalize these observations in the following theorem.

5.3. Theorem. *There is a polynomial space algorithm for deciding satisfiability of formulas in each of the following logics*:

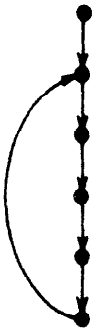
(a) *deterministic propositional flowchart logic*,

(b) DPDL *with only one primitive program*,

(c) *linear time temporal logic*,

(d) *extended linear time temporal logic*.

Proof. As we remarked above, the proof of part (a) is essentially identical to that of SDPDL. For the remaining parts, we note that the techniques of Theorem 4.21 can be used to show that a satisfiable formula of DPDL with only one primitive program (resp. TL, ETL) is satisfiable in a treelike model of the following form:



(Indeed the argument is much simpler than that needed for full SDPDL, since in all these cases there is essentially only one primitive program.) Once we know that for any satisfiable formula such a model exists, we can apply a variant of the algorithm presented in Theorem 5.1 to guess it. Of course, we do not need to use a stack or the notion of weight since here is no branching, and thus the algorithm can be shown to run in nondeterministic space $O(n)$. We omit the details. □

5.4. Remarks. (1) Neither well-structuredness nor determinism is necessary for a polynomial space decision procedure, as we can see by parts (a) and (b) of Theorem 5.3 above. That they are not sufficient follows from results in [13], where it is shown that if we augment the regular programs of SDPDL by adding a small family of

deterministic, well-structured context-free programs (which simulate recursive calls), the satisfiability problem for the resulting logic is undecidable (and, in fact, Π_1^1 complete).

The basic requirements for a logic to have a polynomial space decision procedure seem to be that there be an analogue to the Fischer–Ladner closure of size polynomial in the length of the formula, and that an analogue of Theorem 4.12 holds, i.e., that every satisfiable formula be satisfiable in a tree model with only polynomially many nodes at every depth. As we will show in the next section, the analogue to Theorem 4.12 does *not* hold for DPDL with two primitive programs. Results of Parikh [19] combined with those of Fischer and Ladner [7] also show that the decision procedure for DPDL with two primitive programs is complete in exponential time.

(2) We note that a formula p of TL can be translated to a formula p^t of DPDL with only one primitive program such that p is TL-satisfiable iff p^t is DPDL-satisfiable by using the following unductively defined procedure:

$$P^t = P \quad \text{for an atomic formula } P,$$

$$(\neg p)^t = \neg(p^t), \qquad (p \wedge q)^t = p^t \wedge q^t,$$

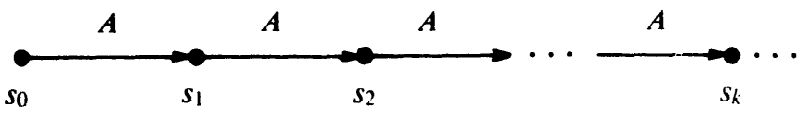
$$(Gp)^t = [A^*]p^t, \quad (pUq)^t = \langle((p^t)?; A)^*\rangle q^t.$$

We leave to the reader the task of checking that the translation has the required property. In Section 6 we will show that each translated formula is actually equivalent to an SDPDL formula of essentially the same length. Note that this also gives us another proof of Theorem 5.3(c). Similar results can also be shown for ETL. We omit details here.

5.5. Theorem. *The decision problem for* SDPDL$_{\text{pt}}$, *and hence* SDPDL, *is polynomial space hard.*

Proof. We use ideas that are similar to those used in Cook's proof that SAT is NP-hard; (cf. [14, pp. 325–327]). Given $L \in$ PSPACE, we can assume without loss of generality (cf. [14, p. 289]) that L is accepted by a one-tape deterministic Turing Machine $\mathcal{M}$ which, for some polynomial p, runs in space $\leq p(n)$ on inputs of length n. We will construct an SDPDL$_{\text{pt}}$ formula $f(x)$ which simulates the computation of $\mathcal{M}$ on input x, where $f(x)$ is computable in polynomial time (and log space) from x. In particular, $f(x)$ will be satisfiable iff $\mathcal{M}$ accepts input x.

Suppose $\mathcal{M}$ has state space K and uses tape alphabet Γ. We can describe the state of $\mathcal{M}$ at a given time by an ID (instantaneous description) of exactly $p(n)$ symbols in $\Gamma'' = (K \times \Gamma) \cup \Gamma$. To simulate this in SDPDL$_{\text{pt}}$ we use primitive predicates P_{wi}, Q_{wi} where $1 \leq i \leq p(n)$ and w ranges over Γ''. 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 $\mathcal{M}$, the ith position on the tape at time j contains symbol w. (If $w=(q, z)\in K\times\Gamma$, then the head of $\mathcal{M}$ is also at the ith position reading symbol z and the machine is in state q.) Moreover, P_{wi} is true at state s_j iff Q_{wi} is true at state s_{j+1}. Thus the Q_{wi} keep track of the values of the P_{wi} at the previous time.

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 up to the time x is accepted, 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 Q_{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 $\mathcal{M}$.

We take $f(x)$ to be the conjunction of the four formulas $f_1(x)$, $f_2(x)$, $f_3(x)$ and $f_4(x)$, which enforce conditions (1) through (4) respectively. Let $x=x_0\cdots x_{n-1}$, where $x_i\in\Gamma$, let q_0 be the initial state of $\mathcal{M}$, 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:

$$\bigwedge_{1\leq i\leq p(n)}\left(\bigvee_{w\in\Gamma}\left(P_{wi}\wedge\left(\bigwedge_{u\neq w}\neg P_{ui}\right)\right)\right).$$

Then $f_1(x)$ is

$$P_{(q_0,x_0),0}\wedge P_{x_1,1}\wedge\cdots\wedge P_{x_{n-1},n-1}\wedge\left(\bigwedge_{n\leq i\leq p(n)}P_{bi}\right)\wedge \textit{string}.$$

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

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

Then $f_3(x)$ 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

$$\langle\textbf{while } \neg\textit{accept } \textbf{do } (\textbf{if } P_{w_1} \textbf{ then } (A\,;Q_{w_1}?) \textbf{ else } (A\,;\neg Q_{w_1}?)\ \textbf{fi})\ \textbf{od}\rangle\textit{true}.$$

Clearly this enforces condition (3).

To see how to write the fourth formula, note that the symbol in the ith position of a given ID is completely determined by the symbols appearing in positions $i-1$, i, $i+1$ in the previous ID. We can therefore easily (again, cf. [14, p. 327]) specify a predicate $R(w_1, w_2, w_3, w_4)$ 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, respectively, 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 Q_{wi}'s (which keep track of what happened at the previous ID), we can now express

the formula *consistent*:

$$\bigwedge_{1 \le i \le p(n)} \left(\bigvee_{\{(u,v,y,z) \mid R(u,v,y,z)\}} Q_{u,i-1} \wedge Q_{vi} \wedge Q_{y,i+1} \wedge P_{zi} \right).$$

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 $\mathcal{M}$. Conversely, if $M, s_0 \models f(x)$, then the graph corresponding to $\mathcal{M}$ looks like the one pictured above, except that it may be an infinite straight line. Let k be the least number such that $M, s_k \models \textit{accept}$. Then it is easy to check that $\mathcal{M}$ accepts x in k steps, where $M, s_j \models P_{wi}$ iff w is in the ith position of the tape at time j. □

From Theorems 5.1 and 5.5 and the fact that p_0 is valid iff $\neg p$ is not satisfiable we immediately obtain the following.

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

5.7. Remarks. (1) In [1] it was already shown that every satisfiable DPDL (and hence SDPDL) formula has a model of size $\leq 4^n n^2$. The question arises if we can do any better for SDPDL formulas. The answer is essentially "no", since by using the techniques of Theorem 5.5 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.

(2) Similar proofs can be used to show that all the logics of Theorem 5.3 have decision problems complete in polynomial space.

6. Expressiveness

6.1. Definition. For two logical languages $\mathcal{L}$ and $\mathcal{M}$, we say $\mathcal{M}$ is at least as *expressive* as $\mathcal{L}$, 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 $\mathcal{L} \not\approx \mathcal{M}$.

Peterson [21], Berman [2] and Berman and Paterson [3] show that, for all $i \geq 0$, $\text{PDL}_i < \text{PDL}_{i+1}$. In fact, these proofs also show $\text{(S)DPDL}_i < \text{(S)DPDL}_{i+1}$. We use our structure theorem (Theorem 4.12) to rederive these results and extend them to show that $\text{SDPDL}_i < \text{DPDL}_i$ and $\text{SDPDL} < \text{DPDL}$.

6.2. Remark. Meyer and Winklmann [17] show that $\text{SDPDL}_{\text{pt}} < \text{DPDL}_{\text{pt}}$ 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,

$$\models \langle A^* \rangle p \equiv \langle \textbf{while } \neg p \textbf{ do } A \textbf{ od} \rangle \mathit{true}$$

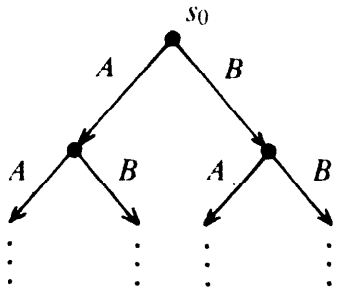
and hence

$$\models [A^*]p \equiv [\textbf{while } p \textbf{ do } A \textbf{ od}] \mathit{false}.$$

From this remark it is easy to see that $\langle A^* \rangle [A^*]P$ *is* equivalent to an SDPDL formula. (Moreover, this observation, combined with Remark 5.4(2), also shows how to translate TL into SDPDL.) However we still get the following theorem.

6.3. Theorem. *For all* $i \geq 0$, *we have* $\text{SDPDL}_i < \text{SDPDL}_{i+1}$ *and* $\text{SDPDL} < \text{DPDL}$.

Proof. Let $\Sigma_0 = \{A, B\}$, and let M be a model whose graph is a full binary tree rooted as 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 test 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 $c_i k^i$ nodes at depth k. Thus, by Theorem 4.12, p_{i+1} is not equivalent to any SDPDL_i formula.

Now consider the DPDL_0 formula $p_0 = [(A \cup B)^*](\langle A \rangle \mathit{true} \wedge \langle B \rangle \mathit{true})$. Again we have $M, s_0 \models p$, but there is clearly no proper submodel of M which sets p_0 at s_0. Thus, by Theorem 4.12, p_0 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_0 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 above proof 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 Peterson [21] and Berman [2], we get the following picture, where languages not connected by a chain of $<$'s are incomparable in expressive power:

$$\begin{array}{ccccccc} \mathrm{DPDL}_0 & < & \mathrm{DPDL}_1 & < \cdots < & \mathrm{DPDL} \\ \vee & & \vee & & \vee \\ \mathrm{SDPDL}_0 & < & \mathrm{SDPDL}_1 & < \cdots < & \mathrm{SDPDL}. \end{array}$$

7. A complete axiomatization for SDPDL

In [1] it was shown that by adding the axiom schema $\langle A\rangle p \to [A]p$ to the Segerberg axioms for PDL we obtain a complete axiomatization for DPDL. Since SDPDL is a restriction of DPDL, this automatically gives us a complete axiomatization of SDPDL. However, we would still have to have an axiomatization that respects the syntax of SDPDL. We present such a system below. Essentially all we do is replace axioms about $\cup$ and * by their appropriate analogues involving **if** ... **then** ... **else** ... **fi** and **while** ... **do** ... **od**. The axioms presented are based on those suggested by Passy (cf. [20]) who proved that this system without axiom (8) was complete for a variant of SDPDL in which the primitive programs were not assumed to be deterministic (note the analogy between PDL and DPDL here.) A similar axiomatization was suggested by Chlebus [6].

7.1. Axiom Schemes and Inference Rules. Consider the following deductive system for SDPDL (where we assume that $\wedge$ is now a symbol in the language):

Axiom Schemes

(1) All (substitution instances of) tautologies of propositional calculus.

(2) $\langle a\rangle(p \vee q) \equiv \langle a\rangle p \vee \langle a\rangle q$.

(3) $\langle a ; b\rangle p \equiv \langle a\rangle\langle b\rangle p$.

(4) $\langle$**if** p **then** a **else** b **fi**$\rangle q \equiv ((p \wedge \langle a\rangle q) \vee (\neg p \wedge \langle b\rangle q))$.

(5) $\langle$**while** p **do** a **od**$\rangle q \equiv ((\neg p \wedge q) \vee (p \wedge \langle a\rangle\langle$**while** p **do** a **od**$\rangle(\neg p \wedge q)))$.

(6) $[a](p \to q) \to (\langle a\rangle p \to \langle a\rangle q)$.

(7) $\langle p?\rangle q \equiv p \wedge q$.

(8) $\langle A\rangle p \to [A]p$ for $A \in \Sigma_0$.

Inference Rules

(9) $$\frac{(p \wedge q) \to [a]q}{q \to [\textbf{while } p \textbf{ do } a \textbf{ od}]q}.$$

(10) $$\frac{p,\ p \to q}{q}$$ (modus ponens).

(11) $$\frac{p}{[a]p}$$ (generalization).

7.2. Theorem. *Axiom schemes and rules* (1)–(11) *above give a complete axiomatization for* SDPDL.

Proof. We essentially follow the lines of [15] and [1]. We say that a formula p is *provable*, and write $\vdash p$, if there exists a finite sequence of formulas, the last one being p, such that each formula is an instance of an axiom scheme or follows from previous formulas by one of the inference rules. A formula p is *consistent* if not $\vdash \neg p$, i.e., if $\neg p$ is not provable in this system. We want to show that any valid SDPDL formula is provable. It suffices to show that if p_0 is consistent, then p_0 is SDPDL satisfiable.

Call a subset of $\mathrm{FL}_s(p_0)$ *maximal* iff for every formula $\neg q \in \mathrm{FL}_s(p_0)$, either $q \in s$ or $\neg q \in s$. If s is a subset of $\mathrm{FL}(p_0)$, let p_s, the *atom associated with* s, be the formula $\bigwedge_{q \in s} q$.

Now suppose p_0 is consistent. Consider the structure $M = (S, \pi, \rho)$, where $S = \{s \subseteq \mathrm{FL}(p_0) \mid s \text{ is maximal, } p_s \text{ is consistent}\}$, π is defined via

$$s \in \pi(p) \quad \text{iff} \quad p \in s \text{ (or equivalently, iff } p \in \mathrm{FL}_s(p_0) \text{ and } \vdash p_s \to p),$$

and ρ is defined on Σ_0 via

$$(s, t) \in \rho(A) \quad \text{iff} \quad p_s \wedge \langle A \rangle p_t \text{ is consistent}$$

and extended in the usual way to all programs.

We will show that although M is not a DPDL structure (there is no reason for $\rho(A)$ to be deterministic), it does satisfy all the tableau conditions in Definition 3.3 for p_0 and

$$\text{if } M, s \models \langle A \rangle p \text{ and } (s, t) \in \rho(A),$$
$$\text{then } M, t \models p \text{ (i.e., } M, s \models \langle A \rangle p \to [A] p). \tag{3}$$

In the language of [1] this makes M *a partial D model for* p_0. Moreover, in [1, Theorem 4.1] it is shown that any partial D model for p_0 can be *unwound* to a model for p_0. Thus in order to show that p_0 is SDPDL satisfiable, it suffices to check the tableau conditions and (3), and show that $p_0 \in s$ for some s.

First note that, for any $q \in \mathrm{FL}_s(p_0)$, using propositional reasoning we can show

$$\vdash q \equiv \bigvee_{t \in S, q \in t} p_t \quad \text{(with } q \equiv \mathit{false} \text{ if the disjunction is empty)}. \tag{4}$$

In particular, (4) holds for p_0. Since we are assuming p_0 is consistent, it follows that $\vdash p_s \to p_0$, and hence $p_0 \in s$, for some $s \in S$.

Tableau conditions (1)–(9) immediately follow from the axioms, since, for each $s \in S$, p_s is consistent and s is maximal.

For tableau condition (10), note that if $M, s \models \langle A \rangle q$, then $\vdash p_s \to \langle A \rangle q$, so using Axiom Schemes (2), (6) and (11), and observation (4) above we get

$$\vdash p_s \equiv p_s \wedge \langle A \rangle q \equiv \bigvee_{t \in S, q \in t} (p_s \wedge \langle A \rangle p_t).$$

Since p_s is consistent, so is $p_s \wedge \langle A\rangle p_t$ for some $t \in S$ with $q \in t$. For this t we have $(s, t) \in \rho(A)$ and $M, t \models q$.

For tableau condition (11), suppose by way of contradiction that $M, s \models \neg\langle A\rangle q$, but for some t with $(s, t) \in \rho(A)$ we have $M, t \models q$. Thus it follows that

$$\vdash p_s \rightarrow \neg\langle A\rangle q, \quad \vdash p_t \rightarrow q \quad \text{and} \quad p_s \wedge \langle A\rangle p_t \text{ is consistent.}$$

Using Axiom Schemes (6), (10) and (11) and propositional reasoning, it follows that $\neg\langle A\rangle q \wedge \langle A\rangle q$ is consistent, and this is a contradiction.

To check (3), suppose that $M, s \models \langle A\rangle q$, but for some t with $(s, t) \in \rho(A)$ we have $M, t \models \neg q$. As above we get

$$\vdash p_s \rightarrow \langle A\rangle q, \quad \vdash p_t \rightarrow \neg q \quad \text{and} \quad p_s \wedge \langle A\rangle p_t \text{ is consistent.}$$

Using Axiom Schemes (6), (8), (10) and (11) and propositional reasoning, it follows that $[A]q \wedge \langle A\rangle \neg q$ is consistent, again a contradiction.

Finally, we can check condition (12) just as condition (10) once we have the following lemma (cf. [15, Lemma 1]).

7.2.1. Lemma. *If there is a formula of the form* $\langle a\rangle q \in \mathrm{FL}_s(p_0)$ *and* $p_s \wedge \langle a\rangle p_t$ *is consistent for* $s, t \in S$, *then* $(s, t) \in \rho(a)$.

Proof. We proceed by induction on the structure of programs. The base case follows by definition, the cases $p?$ and **if** q **then** a **else** b **fi** are straightforward and left to the reader, while the proof for the case $b; c$ is identical to that given in [15]. We thus consider only the case where a is of the form **while** q **do** b **od**.

Suppose $p_s \wedge \langle$**while** q **do** b **od**$\rangle p_t$ is consistent. We want to show that $(s, t) \in \rho($**while** q **do** b **od**$)$. Our first observation is that $\vdash p_t \rightarrow \neg q$. Otherwise, by the maximality of t we have $\vdash p_t \rightarrow q$. But then, by Axiom Schemes (5), (6), (10) and (11) we get

$$\begin{aligned}
&\vdash \langle\textbf{while } q \textbf{ do } b \textbf{ od}\rangle p_t \\
&\quad \rightarrow (p_t \wedge \neg q) \vee \langle b\rangle\langle\textbf{while } q \textbf{ do } b \textbf{ od}\rangle(p_t \wedge \neg q) \\
&\quad \equiv \mathit{false} \vee \langle b\rangle\langle\textbf{while } q \textbf{ do } b \textbf{ od}\rangle \mathit{false} \quad (\text{since} \vdash p_t \rightarrow q) \\
&\quad \equiv \mathit{false}
\end{aligned}$$

which contradicts the consistency of $p_s \wedge \langle$**while** q **do** b **od**$\rangle p_t$.

Let Y be the least subset of S containing t such that if $q \in u$, $(u, v) \in \rho(b)$ and $v \in Y$, then $u \in Y$. It is easy to see that, for all $u \in Y$, $(u, t) \in \rho($**while** q **do** b **od**$)$. Thus it suffices to show that $s \in Y$.

Let r be the formula $\bigvee_{u \notin Y} p_u$. Note that $\vdash \neg r \equiv \bigvee_{v \in Y} p_v$. We claim that $\vdash r \wedge q \rightarrow [b]r$. Suppose not. Then $q \wedge p_u \wedge \langle b\rangle p_v$ is consistent for some $u \notin Y$, $v \in Y$. Thus $q \in u$, and by the induction hypothesis we have $(u, v) \in \rho(b)$, contradicting the fact that $u \notin Y$. Thus, by Axiom Scheme (9), $\vdash r \rightarrow [$**while** q **do** b **od**$]r$. But if $s \notin Y$, then $\vdash p_s \rightarrow r$ and hence $\vdash p_s \rightarrow [$**while** q **do** b **od**$]r$, contradicting the consistency of $p_s \wedge \langle$**while** q **do** b **od**$\rangle p_t$. So $s \in Y$ as desired.

This completes the proof of the lemma, and from it the proof of the theorem. □

Acknowledgment

We would like to acknowledge the anonymous referees for their comments and suggestions on clarifying the presentation of these results.

References

[1] M. Ben-Ari, J.Y. Halpern and A. Pnueli, Finite models of deterministic propositional dynamic logic, in: *Proc. 8th Internat. Conf. on Automata, Languages, and Programming* (1981) pp. 249–263; revised version: Deterministic propositional dynamic logic: Finite models, complexity, and completeness, *J. Comput. System Sci.* **25** (3) (1982) 402–417.

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

[3] F. Berman and M. Paterson, Test-free propositional dynamic logic is strictly weaker than PDL, TR77-10-02, University of Washington, 1978.

[4] P. Berman, J.Y. Halpern and J. Tiuryn, On the expressive power of nondeterminism in dynamic logic, in: *Proc. 9th Internat. Conf. on Automata, Languages, and Programming* (1982) pp. 48–60.

[5] B.S. Chlebus, On the computational complexity of satisfiability in propositional logics of programs, unpublished manuscript, 1981.

[6] B.S. Chlebus, Private communication, 1982.

[7] M.J. Fischer and R.E. Ladner, Propositional modal logic of programs, in: *Proc. 9th Ann. ACM Symp. on Theory of Computing* (Association for Computing Machinery, New York, NY, 1977) pp. 286–294; revised version: Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18** (2) (1979) 194–211.

[8] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, On the temporal analysis of fairness, in: *Proc. 7th Ann. Symp. on Principles of Programming Languages* (1980) pp. 163–173.

[9] S.A. Greibach, *Theory of Program Structures: Schemes, Semantics, Verification*, Lecture Notes in Computer Science **36** (Springer, New York, 1975).

[10] J.Y. Halpern, On the expressive power of dynamic logic, Part II, MIT/LCS/TM-204, 1981.

[11] J.Y. Halpern and J. Reif, The propositional dynamic logic of deterministic, well-structured programs, in: *Proc. 22nd Ann. Symp. on the Foundations of Computer Science* (1981) pp. 322–334.

[12] D. Harel, Logics of programs: Axiomatics and descriptive power, MIT/LCS/TR-200, 1978.

[13] D. Harel, A. Pnueli and J. Stavi, Propositional dynamic logic of context-free programs, in: *22nd Symp. of the Foundations of Computer Science* (1981) pp. 310–321.

[14] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[15] D. Kozen and R. Parikh, An elementary proof of the completeness of PDL, *Theoret. Comput. Sci.* **14** (1) (1981) 113–118.

[16] A.R. Meyer and J. Tiuryn, A note on equivalences among logics of programs, in: D. Kozen, ed., *Proc. IBM Conf. on Logics of Programs*, Lecture Notes in Computer Science **131** (Springer, Berlin, 1982).

[17] A. R. Meyer and K. Winklmann, Expressing program looping in regular dynamic logic, *Theoret. Comput. Science* **18** (3) (1982) 301–324.

[18] G. Mirkowska, On formalized systems of algorithmic logic, *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.* **22** (1974) 421–428.

[19] R. Parikh, Propositional logics of programs: Systems, models, and complexity, in: *7th Ann. ACM Symp. on Principles of Programming Languages* (1980) pp. 186–192.

[20] S. Passy, Filtration lemma for deterministic programming algebras, unpublished manuscript, 1982.

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

[22] A. Pnueli, The temporal logic of programs, in: *18th IEEE Symp. on the Foundations of Computer Science* (1977) pp. 46–57.

[23] V.R. Pratt, Semantical considerations of Floyd–Hoare logic, in: *17th IEEE Symp. on the Foundations of Computer Science* (1976) pp. 109–121.

[24] V.R. Pratt, A practical decision method for propositional dynamic logic, in: *10th Ann. ACM Symp. on the Theory of Computation* (1978) pp. 326–337; revised version: A near optimal method for reasoning about action, *J. Comput. Systems Sci.* **20** (2) (1980) 231–254.

[25] V.R. Pratt, Models of program logics, in: *20th IEEE Symp. on the Foundations of Computer Science* (1979) pp. 115–122.

[26] V.R. Pratt, Using graphs to understand PDL, in: D. Kozen, ed., *Proc. IBM Conf. on Logics of Programs*, Lecture Notes in Computer Science **131** (Springer, Berlin, 1982) pp. 357–396.

[27] A. Salwicki, Formalized algorithmic languages, *Bull. Acad. Pol. Sci., Ser. Math. Astr. Phys.* **18** (5) (1970) 227–232.

[28] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. Systems Sci.* **4** (2) (1970) pp. 147–192.

[29] A. P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logics, in: *14th Ann. ACM Symp. in the Theory of Computation* (1982) pp. 159–169.

[30] P. Wolper, Temporal logic can be more expressive, in: *22nd IEEE Symp. of the Foundations of Computer Science* (1981) pp. 340–348.