MIT/LCS/TM-113

A NEAR-OPTIMAL METHOD FOR REASONING ABOUT ACTION

Vaughan R. Pratt

September 1978

A Near-optimal Method for Reasoning about Action

Vaughan R. Pratt

August, 1978

This research was supported by the National Science Foundation under NSF grant nos. MCS76-18461 and MCS78-04338.

This paper is a revision of "A Practical Decision Method for Propositional Dynamic Logic," presented at the 10th Annual ACM Symposium on Theory of Computing, San Diego, May 1978. The revision incorporates a substantially more efficient method. The material on process logic and axiom systems in the symposium paper has been omitted and will appear elsewhere.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

A Near-optimal Method for Reasoning about Action

Vaughan R. Pratt

Abstract

We give an algorithm for "before-after" reasoning about action. The algorithm decides satisfiability and validity of formulae of propositional dynamic logic, a recently developed logic of change of state that subsumes the zero-order component of most other action-oriented logics. The algorithm requires time at most proportional to an exponentially growing function of the length (number of occurrences of variables and connectives) of the input. Fischer and Ladner have shown that that every algorithm for this problem must take exponential time, making this algorithm optimal to within a polynomial. No decision method for any other logic is known to be optimal to within less than an exponential. The typical time for our algorithm makes it a heuristically efficient algorithm of considerable practical interest. Application areas include program verification, program synthesis, and discourse analysis. The algorithm is based on the method of semantic tableaux, appropriately generalized to dynamic logic. A novel treatment of Hintikka sets via theory algebras supplies the theoretical basis for our treatment of tableaux.

Key words

Dynamic logic, program verification, decision method, exponential time, alternating Turing machine, tableau method, truth set, Hintikka set.

A Near-optimal Method for Reasoning about Action

Vaughan R. Pratt

1. INTRODUCTION

Dynamic Logic

Almost all existing logics of imperative programs contain implicitly or explicitly the construct "*after*(a,p)" which asserts that after program a halts, p holds. They also almost all cater for programming constructs to do with assignment, testing, sequencing, choice and iteration, and logical constructs to do with truth functions and quantification. Dynamic logic [36] consists of (i) a language in which such constructs appear explicitly, and (ii) a formal semantics for that language.

Propositional dynamic logic (PDL) was defined by Fischer and Ladner [12] as the natural restriction of first-order dynamic logic to the term-free case (therefore no assignments, quantifiers or non-zeroary predicates). This restriction is of interest in that it gives a convenient way of studying the term-independent part of reasoning with formulae and programs. Propositional variables (as with any variables) can be viewed as expressions whose internal structure is of no concern in the reasoning at hand. Thus the techniques we develop here apply to more general reasoning about action in the same way that propositional calculus techniques apply to more general static reasoning about specific domains.

The language of PDL is a set of expressions divided into formulae and programs. Letting a,b,c,... range over programs and p,q,r,... over formulae, we may enumerate the expressions of PDL as follows.

The set Φ of formulae:

Atomic formulae:	P, Q, R, ...
Composite formulae:	~p, ⟨a⟩p

The set Σ of programs:

Atomic programs:	A, B, C, ...
Composite programs:	a∪b, a;b, a*, p?

P,Q,R,... are the usual propositional or formula variables, ranging over truth values, and ~ is logical negation. (We obtain all other logical connectives as abbreviations, starting with p∧q for ⟨p?⟩q.) ⟨a⟩p is our notation for "a can ensure p." To be more precise, ⟨a⟩p is true of state u just when p is true of one of the states a can terminate in when started in u. (The need to deal with more than one state in such a definition is what makes this a dynamic logic, as opposed to say the more static propositional calculus, where ~p is true of u just when p is not true of *the same state* u.)

A,B,C,... are program variables, analogous to formula variables. They may range over nondeterministic actions in general. a∪b is the *choice* of a or b. a;b is the *sequence* a followed by b. a^* is the *iteration* of a an indefinite but finite number of times. p? is the *test* of p, a program whose execution is permitted just when p holds. These concepts are made more precise by the semantics given in Section 2.

Typical assertions possible in dynamic logic are ⟨a∪b⟩p (one of a or b can ensure p, equivalent to ⟨a⟩p∨⟨b⟩p), ⟨a*⟩p (a can eventually ensure p), ⟨a⟩⟨b⟩p (a can ensure that b can ensure that p, equivalent to ⟨a;b⟩p), and ~⟨a⟩p (p is guaranteed to be false after executing a, i.e. a cannot attain any state satisfying p).

Definability. Dynamic logic subsumes a number of other logics by offering definitions for their constructs, and we shall take advantage of this throughout the paper. Using this ability we treat p∧q as an abbreviation for ⟨p?⟩q, F as an abbreviation for P∧~P, T as ~F, p⊃q as ~(p∧~q), and similarly for ∨ and ≡. Definable programming concepts include *if* p *then* a *else* b, as (p?;a)∪(~p?;b), and *while* p *do* a, as (p?;a)*;~p?. We define [a]p as ~⟨a⟩~p; [a] is the dual of ⟨a⟩ in the same sense that ∀x is the dual of ∃x. Definable program correctness constructs include Hoare's [20] partial correctness construct p{a}q, definable as p⊃[a]q, and Basu and Yeh's [2] total correctness construct p[a]q for deterministic programs, as p⊃⟨a⟩q.

Dijkstra's total correctness construct *wp*(a,p) for nondeterministic programs is definable as [a]p∧⟨a⟩T∧~∞a where ∞a asserts that a has a diverging computation. For deterministic programs ∞a is definable in DL as [a]F but for the more general case of nondeterministic programs ∞a is not definable in propositional DL. However Meyer and Winklmann [29,44] have shown that it *is* definable in first-order DL.

Theorems. Some formulae are always true, or *valid.* They include p∨~p; ⟨a∪b⟩p≡⟨a⟩p∨⟨b⟩p as we saw above along with ⟨a;b⟩p≡⟨a⟩⟨b⟩p; [a](p⊃q)∧[a]p ⊃ [a]q (a sort of "delayed Modus Ponens"); ⟨a⟩(p∨q) ≡ ⟨a⟩p∨⟨a⟩q; ⟨a⟩p∧⟨a⟩q ⊃ ⟨a⟩(p∧q) (but not the converse); ⟨a*⟩p ≡ p∨⟨a⟩⟨a*⟩p (decomposition of a^* into zero and non-zero number of iterations); ⟨(a;b)*;a⟩p ≡ ⟨a;(b;a)*⟩p; and p∧[a*](p⊃[a]p) ⊃ [a*]p (analogous to mathematical induction). One would expect such obviously valid formulae to be among the theorems of any practical axiom system for DL, and to be efficiently identifiable as valid by any practical decision method for validity.

Rules. We may also observe that if p and p⊃q are valid then so is q (corresponding to the rule of Modus Ponens), and if p is valid then so is [a]p for any a (the rule of Necessitation from modal logic). Any rule whose conclusion is valid when its premises are valid is called *sound.* Other rules include: from p⊃p' and p'{a}q infer p{a}q; from p{a}q' and q'⊃q infer p{a}q; from p{a}q and q{b}r infer p{a;b}r; from p∧r{b}q and p∧~r{b}q infer p{*if* r *then* a *else* b}q; and from p∧q{a}p infer p{*while* q *do* a}p∧~q (Hoare's rules [20]). One would expect such obviously sound rules to be derivable in any practical axiom system for DL.

Example. The following gives a simple example of the sort of problem PDL is useful for. Consider the two programs "while P do (A;A)" and "while P do A". (We assume that testing P has no side-effects, that is, does not cause a change of state.) It is the case that if the first program can reach a final state when started in a given state, so can the second. This is true even if A is nondeterministic. (When A is deterministic, "can reach a final state" means "is guaranteed to halt," or "terminates.") For if not, then P must hold after every execution of A, whence it holds after every execution of A;A.

This valid statement about the relationship between the termination of the respective programs can be easily stated in PDL, as $\langle \textit{while}\ P\ \textit{do}\ (A;A)\rangle T \supset \langle \textit{while}\ P\ \textit{do}\ A\rangle T$, or $\sim\langle\langle(P?;A;A)^*;\sim P?\rangle\sim\langle P?\rangle\sim P?\rangle\sim\langle(P?;A)^*;\sim P?\rangle\sim\langle P?\rangle\sim P$ if we were to expand out *all* our abbreviations (which we obviously wouldn't want to have to do in actual applications).

Outline

The main contribution of this paper is a deterministic exponential time algorithm for deciding satisfiability in PDL.

Fischer and Ladner [12,13] showed how a nondeterministic Turing machine could accept satisfiable PDL formulae of length n within a number of steps proportional to c^n for some c (a *nondeterministic exponential upper bound*), and proved that there did not exist a deterministic Turing machine that could always decide whether an arbitrary PDL formula of length n was satisfiable in fewer than d^n steps for some $d>1$ (a *deterministic exponential lower bound*). The upper bound was obtained by using the equivalent for PDL of the method of truth tables, which enumerates all possible models of a given size and evaluates the formula in each. With our present knowledge about nondeterministic computation the best deterministic upper bound derivable from their result is 2^{c^n} for some c. The lower bound was obtained by reducing the acceptance problem for linear-space-bounded alternating Turing machines to the decision problem for PDL satisfiability.

Using the equivalent for PDL of the method of tableaux we give a *deterministic* exponential upper bound, meeting the lower bound to within a polynomial and giving a method not much worse than any of the thus-far-analyzed methods for satisfiability in ordinary propositional calculus. Our method could be viewed as showing how to reduce the satisfiability problem for PDL to the acceptance problem for linear-space-bounded alternating Turing machines. We shall not however so view it, since the algorithm is just as easy to describe without using alternating machines. The reader familiar with alternation will see the connection without any difficulty.

The significance of our result will be felt first in the area of automatic program verification, where the primary objective is to minimize the amount of detail the programmer must supply in a proof of correctness of a program that is to be certified mechanically. In fact our motivation for studying this problem was to find and apply such an algorithm to the program verification system we have been constructing at MIT during the past two years. Our experience was that the amount of detail required from the user in the area characterizable as the propositional dynamic logic component of the system was the greatest bottleneck in user productivity. Observing the work being done with other verification systems, particularly those of Wegbreit and of Oppen, convinced us that our problems were common to most and probably all program verification systems. The role of dynamic logic in this was to simplify the problem domain, making it easier for us to formulate and solve this particular problem.

We hope that at some time the need for logics of action will be felt by the computational lignuistics community, at which time our algorithm will find application there also.

The result is of theoretical interest because of the scarcity of naturally occurring problems with such tight non-linear bounds. In fact the only other such known problem is that of testing for circularity in Knuth-type attribute grammars, which was shown by Jazayeri, Ogden, and Rounds [22] to have a deterministic time lower bound of $c^{n/\log n}$ for some c>1 and a deterministic time upper bound of d^n for some d for grammars of length n bits. Even this gap, small as it is, is larger than any polynomial.

Our result is of further theoretical interest in that, like the Jazayeri et al result, the tight bounds can be proved easily via a correspondence with a family of automata that characterize deterministic exponential time. There are at present two such families: Cook's family of linear-space auxiliary push-down automata [7], and the Chandra-Kozen-Stockmeyer family of linear-space alternating Turing machines [5,24]. The Jazayeri et al result uses a correspondence with the former, our PDL result with the latter. Actually it is the lower bound that needs to make the connection explicitly, as is done in [12].

The result will be of interest to those concerned with automatic theorem proving because it offers an example of a logic in which the tableau method is provably superior to the method of truth tables, by an exponential. In contrast the tableau method for the propositional calculus tends to perform much better than the method of truth tables in practice, but there is no analysis of the two algorithms that definitively supports this observation.

Our decision method can be viewed as symbolic execution, an approach to program verification that has attracted interest in some circles in the past few years [31,43]. The connection will become apparent when the nonstandard semantics is encountered.

Our application to PDL of the Hintikka set approach [19,41,42] may be of interest to theoretical logicians, as may be our notion of theory algebra within which we embed our treatment of Hintikka sets. The applicability of the Hintikka set approach to binary relations is not immediately obvious until it is seen.

The theory underlying our algorithm also contains most of the material for a proof of completeness of the Segerberg axioms for PDL [40]. Such a proof, using a Gentzen type axiomatization as an obvious intermediate step, was sketched in [37]. (The observant reader of [37] may have noticed two errors in our axiomatization: the two P's in the first rule should be lower case, and, as pointed out to us by M. Valiev, the first premise of our induction rule should read $\Gamma \rightarrow p, \Delta$, not $\Gamma \rightarrow p$.) This application of the theory has been removed from this paper and will appear later as a separate paper. The existence of other proofs of the completeness result, particularly [32], which appears to be the first satisfactory such proof, has lessened for us the urgency of publication of yet another proof. (There also exists a sketch for a proof by D. Gabbay [15].) Furthermore, complexity results and completeness proofs tend to appeal to different audiences; this paper accordingly has focused on complexity to the exclusion of axiomatizations.

Much of our theory is also applicable to our newly developed logic of processes, which also appeared in [37]. In the interests of factoring out our substantive novel contributions the process logic material, like the axiom system, has been transferred to a separate paper.

2. SEMANTICS

This section develops some basic facts and techniques relevant to satisfiability and validity in PDL. Our treatment has a debt to Smullyan's study [41], although our algebraic approach to Hintikka sets will differ somewhat from Smullyan's approach.

States, Facts, Transitions, Statements, Theories

States play a dual role in dynamic logic - formulae take on truth values in states, and programs travel from state to state. Given a set W of states, we take $W\times\Phi$ to be the set of all possible *facts* (u,p), written $u\models p$, and $W\times\Sigma\times W$ the set of all possible *transitions* (u,a,v), written $u\langle a\rangle v$. Although the complement ~a of program a is not in PDL, we will permit $u\langle\sim a\rangle v$ as a possible transition. The literal facts are $u\models P, u\models\sim P, u\models Q, u\models\sim Q,\ldots$, the literal transitions $u\langle A\rangle v, u\langle\sim A\rangle v, u\langle B\rangle v, u\langle\sim B\rangle v,\ldots$, for all $u,v\in W$.

A PDL *statement* is a fact or transition. Thus in this paper we draw a distinction between the statement $u\models p$ and the formula p. (The pun may be of some mnemonic value.) We let s,t,... range over statements.

A PDL *theory* is a set of statements. The theory defines what W is for that theory, namely those states that appear in some statement of the theory. (In modal logic practice, as in [12], W is given explicitly. If we were to follow this practice we would define a theory to be a pair (W,S) where S is a set of statements, or a triple (W,π,ρ) where π is a set of facts and ρ a set of transitions, the triple being in effect the structures of [12].) We let x,y,z,... range over theories.

The role of a PDL theory x is to *assign meanings to formulae and programs.* If $u\models p\in x$ we consider formula p to be *true*, or to *hold*, in state u. If $u\langle a\rangle v\in x$ we consider it possible for program a to go from state u to state v. The presence of statements s and ~s in a theory is an inconsistency. The absence of s and ~s is considered merely a lack of information about the truth values of s and ~s, either due to insufficient computation (as when both $u\langle a\rangle v$ and $v\models p$ are present but $u\models\langle a\rangle p$ is not) or to genuine ignorance (as when both $u\langle A\rangle v$ and $u\langle\sim A\rangle v$ are absent and only literal statements are present).

A PDL theory x may be envisaged as a graph each of whose vertices $u\in W$ is labelled with formulae p, namely those such that $u\models p\in x$, and each of whose edges from u to v is labelled with programs a, namely those such that $u\langle a\rangle v\in x$. Alternatively a separate labelled edge may be drawn for each transition. See [12] for some examples of theories presented in this way.

Semantics

We give the standard semantics of PDL in a form somewhat different from that in [12]. To illustrate the form we first give the semantics of the propositional calculus with connectives ~ and $\wedge$.

p,q	$\vdash$	$p \wedge q$
$\sim p$	$\vdash$	$\sim(p \wedge q)$
$\sim q$	$\vdash$	$\sim(p \wedge q)$

This table specifies a relation from theories to formulae: $\vdash$ is the least relation satisfying the above and also satisfying the conditions that if $y \vdash s$ and $y \subseteq x$ then $x \vdash s$, and if s is literal then $s \vdash s$. Strictly speaking we should write $\{p,q\} \vdash p \wedge q$ and $\{s\} \vdash s$, but we shall make a habit of dropping set braces in this context.

The intuitive meaning of $x \vdash s$ is that if all statements of x are true then s ought also to be true. In addition there ought to be some sense in which the truth of s follows "by calculation" from a subset of the elements of x. The requirement that $s \vdash s$ for literals is to reduce the extent to which we need to treat literals as a special case.

The intuitive meaning notwithstanding, we shall be quite strict about $\vdash$ being the least relation satisfying the given conditions. Thus we have $s \vdash s$ *only* when s is literal, and we never have $s \vdash (s \wedge s) \wedge (s \wedge s))$. The reason for this strictness is that we want to work with $\vdash$ in a quite mechanical and non-semantical way.

We say $x \vdash s$ *minimally* when $x \vdash s$ and if $y \vdash s$ and $y \subseteq x$ then $y=x$. Thus if $x \vdash p \wedge q$ minimally then x is p,q, while if $x \vdash p \vee q$ minimally then x is p or q. If s is literal then $s \vdash s$ minimally. In defining $\vdash$ by tabulating instances of $x \vdash s$, we tabulate just the minimal instances, omitting $s \vdash s$ for literal s.

The semantics of $\sim$ cannot be given explicitly under such a scheme. Instead we give it implicitly, albeit wastefully, by tabulating $x \vdash s$ and $x \vdash \sim s$ separately for each statement s. In addition we assume that $\sim\sim p$ and p are the same statement, or if you prefer, that $\sim\sim p$ is not an allowed statement and p and $\sim p$ are mutual negations.

We shall make a practice of using such abbreviations as $\sim p \vee \sim q$ for $\sim(p \wedge q)$, so that the above table would read $p,q \vdash p \wedge q$, $p \vdash p \vee q$, $q \vdash p \vee q$, recalling that $\sim\sim p = p$.

Standard PDL Semantics

This example and accompanying remarks should now make it possible to grasp the content of the following semantics for PDL. In this semantics W is an arbitrary set about which we will have more to say shortly. We abbreviate $u \models \sim\langle a\rangle p$ to $u \models [a]\sim p$.

$\langle a\rangle$	$u\langle a\rangle v, v \models p$	$\vdash$	$u \models \langle a\rangle p$
$\langle ?\rangle$	$u \models p$	$\vdash$	$u\langle p?\rangle u$
$\langle \cup\rangle$	$u\langle a\rangle v$	$\vdash$	$u\langle a \cup b\rangle v$
$\langle \cup\rangle$	$u\langle b\rangle v$	$\vdash$	$u\langle a \cup b\rangle v$
$\langle ;\rangle$	$u\langle a\rangle v, v\langle b\rangle w$	$\vdash$	$u\langle a;b\rangle w$
$\langle * \rangle$	$u_0\langle a\rangle u_1,\ldots,u_{k-1}\langle a\rangle u_k$	$\vdash$	$u_0\langle a^*\rangle u_k \quad k \geqslant 0$
$[a]$	$\{u\langle\sim a\rangle v \text{ or } v \models p \mid v \in W\}$	$\vdash$	$u \models [a]p$
$[?]$	$u \models \sim p$	$\vdash$	$u\langle\sim(p?)\rangle u$
$[?]$		$\vdash$	$u\langle\sim(p?)\rangle v \ (v \neq u)$

[U]	$u\langle\sim a\rangle v, u\langle\sim b\rangle v$	$\vdash$	$u\langle\sim(a\cup b)\rangle v$
[;]	$\{u\langle\sim a\rangle v$ or $v\langle\sim b\rangle w \mid v\in W\}$	$\vdash$	$u\langle\sim(a;b)\rangle v$
[*]	$\{w\langle\sim a\rangle w' \mid (w,w')\in C\}$	$\vdash$	$u\langle\sim(a^*)\rangle v$ for any cut-set C of K_W separating u,v.

(A cut-set of a directed graph separating vertices u,v is a minimal set of edges whose removal from the graph would eliminate all paths from u to v. K_W is the complete graph $W\times W$ on vertex set W.)

Duality

In addition to the binary relation $\vdash$ on theories we shall find it convenient to treat $\sim$ (*conjugate*) as a unary function on theories which replaces the statements of a theory by their logical negations, cancelling double negations. (The squarepotent nature of negation is the one semantical fact not embedded in $\vdash$ but rather in the underlying algebra.) Conjugation satisfies $\sim\sim x=x$, $\sim(x\cup y)=\sim x\cup\sim y$, $\sim(x')=(\sim x)'$, and $x=\sim x \rightarrow \exists y=x-\sim y$ (no statement is its own negation).

We call $\vdash$ *dual* when $\sim y\vdash\sim s$ iff $\forall x\vdash s(x\cap y\neq\varphi)$. The semantics for $\wedge$ is dual. To see this, consider first the literal P. Then if $x\vdash P$ and $\sim y\vdash\sim P$, P must be common to x and y. Now consider the nonliteral $s = p\wedge q$. Then x must be p,q and $\sim y$ must be p or q, so x,y are not disjoint. Moreover, if x has some element in common with p,q then x must contain one of p or q so $\sim x\vdash\sim(p\wedge q)$. The case $s = \sim(p\wedge q)$ is just the dual of this argument. Similarly it can be shown, at greater length, that the PDL semantics is dual. (We could have given just the first half of the PDL semantics together with the condition that $\vdash$ be dual, but checking that this yields the same semantics is awkward.)

Models

The following four properties (grouped as two mutually converse pairs, one pair for each of $\sim$ and $\vdash$) formalize the intended interpretations of $\sim$ and $\vdash$ in the sense that the ideal theory ought to satisfy all four of these properties.

x	*consistent*:	$\sim x \subseteq x'$
x	*complete*:	$x' \subseteq \sim x$
x	*Hintikka*:	$s\in x \rightarrow x\vdash s$
x	*closed*:	$x\vdash s \rightarrow s\in x$

A *model* (of a statement s) is a consistent complete closed Hintikka theory (containing s). We call s *satisfiable* when there exists a model of s. In the case of PDL we call p satisfiable when $u\vDash p$ is satisfiable for some u.

Testing Satisfiability

We test satisfiability by explicitly constructing a model. We shall approach such a construction cautiously, disposing of the four properties for model-hood one at a time in the order closed, complete, Hintikka, consistent. The first of these is easily disposed of.

Lemma 2.1. When $\vdash$ is dual, if x is consistent, complete and Hintikka then x is closed.

Proof. Otherwise for some s, $x \vdash s \in x' \subseteq \sim x$ (x complete), so $\sim s \subseteq x$, whence $x \vdash \sim s$ (x Hintikka), so $x \cap \sim x \neq \varphi$ ($\vdash$ dual), so x is not consistent. ∎

Completeness

The next step in constructing models is to dispose of completeness. We show how, given some consistent Hintikka theory x, we can form a complete consistent Hintikka theory by iterating the process of adding to x all s such that $x \vdash s$. If we write $x\vdash$ for $\{s | x \vdash s\}$, the construction is to form $x \cup x\vdash \cup x\vdash\vdash \cup \ldots$, as justified by the following.

Lemma 2.2. If x is Hintikka then $x \subseteq (x\vdash)$.

Proof. Immediate. ∎

Lemma 2.3. If x is Hintikka so is $x\vdash$.

Proof. $s \in x\vdash$ means $x \vdash s$, so $x\vdash \vdash s$ by Lemma 2.2. ∎

Lemma 2.4. When $\vdash$ is dual, if x is consistent so is $x\vdash$.

Proof. If $s, \sim s \in x\vdash$ then $x \vdash s$ and $x \vdash \sim s$, so by duality $x \cap \sim x \neq \varphi$, contradicting consistency. ∎

To take care of completeness we need a further constraint on $\vdash$. We call $\vdash$ ***inductive*** when there exists a function h (the ***height*** function) mapping statements to natural numbers such that

(i) when s is literal, $h(s)=h(\sim s)=0$;
(ii) when s is non-literal and $x \vdash s$ minimally, $h(s)=h(\sim s)>h(t)$ for all $t \in x$.

Taking $h(P) = h(\sim P) = 0$ for each variable P and $h(p \wedge q) = h(\sim(p \wedge q)) = 1+\max(h(p),h(q))$ in the $\wedge$ example shows that our propositional calculus semantics is inductive. In PDL we abbreviate $h(u\langle a\rangle v)$ to $h(a)$ and $h(u \vDash \langle a\rangle p)$ to $h(\langle a\rangle p)$, and take $h(A) = h(\sim A) = h(P) = h(\sim P) = 0$, $h(p?) = 1+h(p)$, $h(a \cup b) = h(a;b) = 1+\max(h(a),h(b))$, $h(a^*) = 1+h(a)$, and $h(\langle a\rangle p) = 1+\max(h(a),h(p))$. Inspection of the PDL semantics shows that it too is inductive.

When such an h exists we define x_n to be $\{s \in x | h(s) \leqslant n\}$, and say that x is ***n-complete*** when $(x')_n \subseteq (\sim x)_n$.

Lemma 2.5. Given $n \geqslant 0$ and $\vdash$ dual and inductive, if x is n-complete then $x\vdash$ is n+1-complete.

Proof. We show $((x\vdash)')_{n+1} \subseteq (\sim(x\vdash))_{n+1}$. Let $s \in ((x\vdash)')_{n+1}$. If s is literal then $s \in (x')_0 \subseteq (\sim x)_0$ whence $x \vdash \sim s$ so $s \in (\sim(x\vdash))_{n+1}$. Otherwise since $x \not\vdash s$ we have $z \cap (x')_n \neq \varphi$ for all $z \vdash s$ by inductive condition (ii), so by duality $\sim(x')_n \vdash \sim s$. But $(x')_n \subseteq (\sim x)_n \subseteq \sim x$, so $\sim(x')_n \subseteq x$, so $x \vdash \sim s$, whence $s \in (\sim(x\vdash))_{n+1}$. ∎

Define $x\vdash^0 = x$, $x\vdash^{n+1} = (x\vdash^n)\vdash$, and $x\vdash^* = x\vdash^0 \cup x\vdash^1 \cup x\vdash^2 \cup \ldots$

Lemma 2.6. When $\vdash$ is dual and inductive, if x is consistent, 0-complete and Hintikka then $x\vdash^*$ is consistent, complete and Hintikka.

Proof. Combine lemmas 2.3-2.5. ∎

Given a consistent Hintikka theory x, it may be made 0-complete without compromising its being consistent or Hintikka merely by adding one of s or ~s for each literal s not already in $x \cup \sim x$.

Non-standard PDL semantics

Our next goal is the Hintikka property. A direct assault has eluded us, and we shall approach the construction indirectly via a "semantics" for PDL that is less plausible as a definition but more convenient computationally. (In the case of propositional calculus such a step is not necessary - the standard semantics leads directly to the usual tableau method.)

With the non-standard semantics comes a third type of statement, the *link* $s \ni t$ where s and t are facts. Links are used in connection with facts of the form $u\models\langle a\rangle p$. They supply a mechanism for retaining the finite character of $*$, via the condition $\langle\ni\rangle_+$ below. A *chain* is a finite set of links $s_0 \ni s_1, s_1 \ni s_2, \ldots, s_{k-1} \ni s_k$. We write $s \ni^* t \subseteq x$ to indicate that x contains a chain starting with s and ending with t. As for any relation we write $s \ni s' \ni s''$ for $s \ni s', s' \ni s''$ and $s \ni^* s' \ni^* s'' \subseteq x$ for $s \ni^* s' \cup s' \ni^* s'' \subseteq x$.

Non-standard PDL Semantics

$\langle\ni\rangle_+$	$u\models\langle a\rangle p \ni^* v\models p$	$\vdash_+$	$u\models\langle a\rangle p$
$\langle A\rangle_+$	$u\langle A\rangle v, v\models p$	$\vdash_+$	$u\models\langle A\rangle p \ni v\models p$
$\langle ?\rangle_+$	$u\models p, u\models q$	$\vdash_+$	$u\models\langle p?\rangle q \ni u\models q$
$\langle \cup\rangle_+$	$u\models\langle a\rangle p$	$\vdash_+$	$u\models\langle a\cup b\rangle p \ni u\models\langle a\rangle p$
$\langle \cup\rangle_+$	$u\models\langle b\rangle p$	$\vdash_+$	$u\models\langle a\cup b\rangle p \ni u\models\langle b\rangle p$
$\langle ;\rangle_+$	$u\models\langle a\rangle\langle b\rangle p$	$\vdash_+$	$u\models\langle a;b\rangle p \ni u\models\langle a\rangle\langle b\rangle p$
$\langle *\rangle_+$	$u\models p$	$\vdash_+$	$u\models\langle a^*\rangle p \ni u\models p$
$\langle *\rangle_+$	$u\models\langle a\rangle\langle a^*\rangle p$	$\vdash_+$	$u\models\langle a^*\rangle p \ni u\models\langle a\rangle\langle a^*\rangle p$
$[A]_+$	$\{u\langle\sim A\rangle v \text{ or } v\models p \mid v\in W\}$	$\vdash_+$	$u\models[A]p$
$[?]_+$	$u\models q$	$\vdash_+$	$u\models[p?]q$
$[?]_+$	$u\models\sim p$	$\vdash_+$	$u\models[p?]q$
$[\cup]_+$	$u\models[a]p, u\models[b]p$	$\vdash_+$	$u\models[a\cup b]p$
$[;]_+$	$u\models[a][b]p$	$\vdash_+$	$u\models[a;b]p$
$[*]_+$	$u\models p, u\models[a][a^*]p$	$\vdash_+$	$u\models[a^*]p$

We define +-Hintikka to mean Hintikka with respect to the nonstandard semantics.

The conditions that spoil inductiveness are $[*]_+$ and $\langle * \rangle_+$. We could easily make $\vdash_+$ dual by adding further constraints, but it will turn out that we do not need duality for $\vdash_+$.

We now wish to show that, for the purposes of defining satisfiability of a PDL formula, this semantics is equivalent to the standard semantics. Our strategy will be as follows. Let h be the height function for the standard semantics (with the convention that $h(u\langle a\rangle v), h(s\ni t) \nleq n$ for any nonliteral a and any n including ω so that we can write x_ω for the non-literal-transition-free link-free part of x), and let x_n be defined with respect to this h. Taking $\hat{x}$ as $x_0 \cup (V-(x_0 \cup \sim x_0))$, we shall prove that if x is consistent and +-Hintikka, then $x_\omega \subseteq \hat{x}\vdash^*$. By observing that $\hat{x}$ is consistent, 0-complete and Hintikka we infer that if $u\models p \in x$ then $\hat{x}\vdash^*$ is a model of $u\models p$, whence p is satisfiable. We also prove the converse: if $u\models p$ has a model it occurs in some consistent +-Hintikka theory. This gives us a useful test for satisfiability: see whether $u\models p$ belongs to some consistent +-Hintikka theory.

The double induction of the proof of the main theorem has a sufficiently delicate "inner induction" that we isolate it as two lemmas. For each of these lemmas we assume (as part of the "outer" induction hypothesis) that x is +-Hintikka and that n is a positive integer such that for $0 \leq m < n$, $x_m \subseteq \hat{x}\vdash^m$, and $\hat{x}\vdash^m$ is consistent, m-complete and Hintikka.

Lemma 2.7. If $u\models[c]p \in x$ and $h(c) \leq n$, then for all $v \in W$ either $u\langle\sim c\rangle v \in \hat{x}\vdash^n$ or $v\models p \in x$.

Proof. We use induction on h(c).

$u\models[A]p \in x$. By $[A]_+$, $u\langle\sim A\rangle v$ or $v\models p \in x$ for all $v \in W$. Since $h(\sim A) = 0$, $u\langle\sim A\rangle v \in x_0 \subseteq \hat{x} \subseteq \hat{x}\vdash^n$ or $v\models p \in x$ for all $v \in W$.

$u\models[p?]q \in x$. By $[?]$, for all $v \neq u$ $u\langle\sim(p?)\rangle v \in \hat{x}\vdash \subseteq \hat{x}\vdash^n$. For $v=u$, we have by $[?]_+$ $u\models\sim p$ or $u\models q \in x$. If the latter we are done. Otherwise, since $h(\sim p) \leq n-1$, $u\models\sim p \in x_{n-1} \subseteq \hat{x}\vdash^{n-1}$, so by $\langle ?\rangle$ $u\langle\sim(p?)\rangle u \in \hat{x}\vdash^n$.

$u\models[a\cup b]p \in x$. By $[\cup]_+$ $u\models[a]p, u\models[b]p \in x$. So by induction, for all $v \in W$ $u\langle\sim a\rangle v \in \hat{x}\vdash^{n-1}$ or $v\models p \in x$, and for all $v \in W$ $u\langle\sim b\rangle v \in \hat{x}\vdash^{n-1}$ or $v\models p \in x$. Thus for all $v \in W$ $u\langle\sim a\rangle v, u\langle\sim b\rangle v \in \hat{x}\vdash^{n-1}$ or $v\models p \in x$. So by $[\cup]$, for all $v \in W$ $u\langle\sim(a\cup b)\rangle v \in \hat{x}\vdash^n$ or $v\models p$.

$u\models[a;b]p \in x$. By $[;]_+$ $u\models[a][b]p \in x$, so by induction $u\langle\sim a\rangle v \in \hat{x}\vdash^{n-1}$ or $v\models[b]p \in x$ for all v, and so again by induction $u\langle\sim a\rangle v \in \hat{x}\vdash^{n-1}$ or $v\langle\sim b\rangle w \in \hat{x}\vdash^{n-1}$ or $w\models p \in x$ for all v,w. But then $u\langle\sim(a;b)\rangle w \in \hat{x}\vdash^n$ or $w\models p \in x$ for all w.

$u\models[a^*]p \in x$: Suppose that for some $v \in W$, $u\langle\sim(a^*)\rangle v \notin \hat{x}\vdash^n$ and $v\models p \notin x$. Since $h(u\langle\sim a^*\rangle v) \leq n$ and $\hat{x}\vdash^n$ is n-complete, $u\langle a^*\rangle v \in \hat{x}\vdash^n$, i.e. $(\hat{x}\vdash^{n-1})\vdash u\langle a^*\rangle v$. Since $x\vdash^{n-1}$ is Hintikka, $u_0\langle a\rangle u_1, \ldots, u_{k-1}\langle a\rangle u_k \in \hat{x}\vdash^{n-1}$ for some $u_0, \ldots, u_k$, $k \geq 0$, where $u_0 = u$ and $u_k = v$, by $\langle * \rangle$. Now by $[*]p$, $u_0\models p, u_0\models[a][a^*]p \in x$. So by induction either $u_0\langle\sim a\rangle u_1 \in \hat{x}\vdash^{n-1}$ or $u_1\models p \in x$. The former is ruled out by the presence in $\hat{x}\vdash^{n-1}$ of $u_0\langle a\rangle u_1$ and the consistency of $\hat{x}\vdash^{n-1}$. We may continue in this way to show $u_2\models p,\ldots,u_k\models p \in x$ by induction on k, contradicting $v\models p \notin x$ since $v=u_k$. ∎

Lemma 2.8. For all formulae of the form $Lp = \langle c_1\rangle\langle c_2\rangle\ldots\langle c_g\rangle p$, $g\geqslant 0$, if $u\models\langle c\rangle Lp\Rrightarrow^* w\models p \in x$ and $h(c)\leqslant n$, there exists $v\in W$ such that $u\models\langle c\rangle Lp\Rrightarrow^* v\models Lp\Rrightarrow^* w\models p \in x$ and $u\langle c\rangle v \in \hat{x}\vdash^{n}$.

Proof. We use induction on h(c), assuming $u\models\langle c\rangle Lp\Rrightarrow^* w\models p \in x$.

c=A. Then $u\models\langle A\rangle Lp\Rrightarrow v\models Lp\Rrightarrow^* w\models p \in x$ and $u\langle A\rangle v \in x$, by ⟨A⟩. Since $h(A) = 0$, $u\langle A\rangle v \in x_0 \subseteq \hat{x}\vdash^{n}$.

c=p?. Then $u\models\langle p?\rangle Lp\Rrightarrow u\models Lp\Rrightarrow^* w\models p \in x$, and $u\models p \in x$, by ⟨?⟩. So $u\models p \in x_{n-1} \subseteq \hat{x}\vdash^{n-1}$ by induction, whence $u\langle p?\rangle u \in \hat{x}\vdash^{n}$ by ⟨?⟩.

c=a∪b. Then without loss of generality $u\models\langle a\cup b\rangle Lp\Rrightarrow u\models\langle a\rangle Lp\Rrightarrow^* v\models Lp\Rrightarrow^* w\models p \in x$ and $u\langle a\rangle v \in \hat{x}\vdash^{n-1}$, by ⟨∪⟩ and induction. So $u\langle a\cup b\rangle v \in \hat{x}\vdash^{n}$ by ⟨∪⟩.

c=a;b. Then $u\models\langle a;b\rangle Lp\Rrightarrow u\models\langle a\rangle\langle b\rangle Lp\Rrightarrow^* v\models\langle b\rangle Lp\Rrightarrow^* v'\models Lp\Rrightarrow^* w\models p \in x$ and $u\langle a\rangle v, v\langle b\rangle v' \in \hat{x}\vdash^{n-1}$, by ⟨;⟩ and induction (twice). So $u\langle a;b\rangle v' \in \hat{x}\vdash^{n}$ by ⟨;⟩.

c=a*. Then, taking $u_0=u$, $u_0\models\langle a^*\rangle Lp\Rrightarrow u_0\models\langle a\rangle\langle a^*\rangle Lp\Rrightarrow^* u_1\models\langle a^*\rangle Lp\Rrightarrow u_1\models\langle a\rangle\langle a^*\rangle Lp\Rrightarrow^* u_2\models\ldots\Rrightarrow u_m\models\langle a^*\rangle Lp\Rrightarrow u_m\models Lp\Rrightarrow^* w\models p \in x$ and $u_0\langle a\rangle u_1, u_1\langle a\rangle u_2,\ldots,u_{m-1}\langle a\rangle u_m \in \hat{x}\vdash^{n-1}$, by ⟨*⟩ and induction ($m\geqslant 0$ times). Since $u\models\langle c\rangle Lp\Rrightarrow^* w\models p$ is finite, m must be too. Hence $u_0\langle a^*\rangle u_m \in \hat{x}\vdash^{n}$ by ⟨*⟩. ∎

Theorem 2.9. Let x be consistent and +-Hintikka. Then x_ω can be extended to a PDL model.

Proof. We first show by induction that for all $n\geqslant 0$, $x_n\subseteq\hat{x}\vdash^{n}$.

Certainly $x_0\subseteq\hat{x}\vdash^{0}$, by construction of $\hat{x}$. Now suppose that the inductive claim holds for n, and that $s\in x_{n+1}$. If s is $u\models[c]p$ with $h(\sim c), h(p) \leqslant n$ then by lemma 2.7 $u\langle\sim c\rangle v \in \hat{x}\vdash^{n}$ or $v\models p \in x$ for all $v\in W$, and $v\models p \in x_n \subseteq \hat{x}\vdash^{n}$ by induction. So by [a], $u\models[c]p \in \hat{x}\vdash^{n+1}$. If s is $u\models\langle c\rangle p$ with $h(c), h(p)\leqslant n$ then by ⟨⇛⟩, for some $w\in W$ $u\models\langle a\rangle p\Rrightarrow^* w\models p$. So by lemma 2.8 (taking g = 0) there exists v such that $u\models\langle c\rangle p\Rrightarrow^* v\models p \in x$ and $u\langle c\rangle v \in \hat{x}\vdash^{n}$. So by ⟨⇛⟩, $s\Rrightarrow v\models p$ for some $s\in\Phi$, so by the appropriate $\langle\rangle_+$ condition $v\models p \in x_n \subseteq \hat{x}\vdash^{n}$. Hence $u\models\langle c\rangle p \in \hat{x}\vdash^{n+1}$. This establishes that $x_{n+1}\subseteq\hat{x}\vdash^{n+1}$.

It follows that $x_\omega \subseteq \hat{x}\vdash^{*}$. Since $\hat{x}$ is consistent (since x is), 0-complete (by construction), and Hintikka, $\hat{x}\vdash^{*}$ is a model by Lemma 2.6. ∎

Theorem 2.10. If x is a PDL model then x can be extended to a consistent +-Hintikka theory merely by adding links.

Proof. The reader may verify that the $[]_+$ conditions are all met by any PDL model x. The links to be added to x to meet the $\langle\rangle_+$ conditions are $u\models\langle A\rangle p\Rrightarrow v\models p$ whenever $u\langle A\rangle v, v\models p, u\models\langle A\rangle p \in x$, $u\models\langle p?\rangle\Rrightarrow u\models q$ whenever $u\models p, u\models q, u\models\langle p?\rangle q \in x$, and similarly for the links named in the other $\langle\rangle_+$ conditions. These additions satisfy $\langle A\rangle_+$ through $\langle *\rangle_+$ by construction. The reader may verify that the additions also satisfy $\langle\Rrightarrow\rangle_+$ automatically. ∎

It follows that $u\models p$ is satisfiable if and only if $u\models p \in x$ for some consistent +-Hintikka theory x. Henceforth we shall work only with $\models_+$ semantics, permitting us to drop the + without confusion.

3. TABLEAUX

So far we have disposed of the properties of being closed and complete, leaving the properties of being consistent and Hintikka. We now show how to use the classical method of tableaux [4,16,41,42], via the non-standard semantics, to test for satisfiability of PDL formulae. The tableau method has two parts: tableau construction, which takes care of the Hintikka property, and tableau testing, which takes care of consistency. We begin with easy-to-grasp but not very effective constructions and tests, and gradually bring them up to speed to yield the final algorithm.

A *tableau for* r is a rooted unordered tree whose vertices are labelled with theories, the root being labelled with $w\models r$ for an arbitrarily chosen state w. We use the terms "parent," "child," "sibling," for the obvious relationships among vertices, using "descendant" for the reflexive transitive closure of the "child" relation.

The tableau exhibits an exhaustive search for a consistent Hintikka theory as follows. Each path in the tableau is labelled with a sequence of theories $x_0\subseteq x_1\subseteq x_2\subseteq\ldots$ such that for each consecutive pair x_i,x_{i+1} in the sequence, there is some non-literal $s\in y$, the *chosen* non-literal of x_i, such that $x_i\not\vdash s$ and $x_{i+1} = x_i\cup z$ where $z\vdash s$ minimally. For every path, and every x_i on that path, and every statement $s\in x_i$, there exists x_j on that path with $i\leq j$ and $x_j\vdash s$, the *ω-property*.

The one non-constructive aspect of this search is the choice of W in condition $[A]_+$. We deal with $[A]_+$ constructively by taking W at each vertex to be whatever states are mentioned in theories along the path from the root to this vertex. This means that any given $u\models[A]p$ may need to be chosen infinitely often along any given path, to take into account the infinitely many states that may appear along the path. However this does not invalidate the properties we want tableaux to have.

To see that such a tableau can exist for any r we may give an explicit construction. (A different construction will be made possible by theorem 3.3, but since that theorem has very narrow applicability while these preliminary ideas are very general, this construction is worth mentioning.) At each vertex of the tree under construction, partition the theory labelling that vertex into a *queue* (a totally ordered set) and a *grave* (an unordered set). At the root, if $w\models r$ is non-literal the queue contains just $w\models r$ and the grave is empty, otherwise $w\models r$ appears only in the grave. At vertex V the chosen non-literal s of y is always the first element on the queue part of y. For each of the finitely many finite z such that $z\vdash s$ minimally, construct a child of V whose theory is $y\cup z$, whose queue is that of y less s and with z (in any order) on the end, and whose grave is that of y plus s. If s is of the form $u\models\langle a\rangle p$ then z may contain new states, in which case all statements of the form $u\models[A]p$ must be removed from the grave and put on the end of the queue following z.

By this construction the queue and grave of every vertex will be finite, which in turn guarantees that the constructed tableau has the ω-property.

Lemma 3.1. The union of theories along any path is a Hintikka theory.

Proof. By the ω-property, every possible counter-example to the Hintikka property is taken care of along a path, even for $u\models[A]p$. ∎

We have taken care of the Hintikka property, leaving only the consistency property to be tested.

A Test for Satisfiability

We call a test of satisfiability of statements *sound* when it never claims that a satisfiable statement is unsatisfiable, and *complete* when it never claims that an unsatisfiable statement is satisfiable. These terms are motivated by the application of a satisfiability tester to validity checking, where their meaning coincides with normal usage.

Our basic test is that r is satisfiable if and only if the union of the theories along some path of a tableau for r is consistent. Completeness follows from the fact that if such a consistent theory is found, it must also be Hintikka and contain r, whence r is satisfiable. Conversely suppose the root theory $u\models r = x_0$ is satisfiable. We proceed by induction. We have $x_0 \subseteq y$ for some consistent Hintikka theory y. Now suppose $x_i \subseteq y$. Let $x_{i+1} = \cap\{z \mid z\vdash s$ and $x_i \subseteq z \subseteq y\}$ where s is the chosen non-literal of x_i. Then x_{i+1} must be one of x_i's children, being minimal satisfying $x_{i+1}\vdash s$, $x_i \subseteq x_{i+1}$. It follows by induction that there is a path the union of whose theories is a subset of y. This subset must be consistent since y is. So the test is *sound.*

Inspecting all paths is less convenient than inspecting all vertices, or even all vertices standing in the descendant relation to one another. Define the predicate *bad* on vertices to be the least predicate such that bad(V) holds when:

(i) V's label is inconsistent.

(ii) All V's children are bad, unless V has no children.

The least predicate satisfying these conditions may be formed as the intersection of all predicates satisfying them; the intersection can readily be seen to satisfy the conditions.

Lemma 3.2. Given any tableau for r, r is satisfiable if and only if the root is not bad.

Proof. (If.) If the root is not bad there must be an infinite path of good vertices from the root, the union of whose theories must then be consistent.

(Only if.) If r is satisfiable then there is a consistent Hintikka theory y such that $w\models r \in y$. The tableau is constructed in such a way that if the theory labelling V is a subset of y (as is the case for V being the root) then so is some child of V. Hence there must exist an infinite path the union of whose theories is a subset of y, and hence consistent since y is. ∎

The marking procedure

We would like to identify the bad vertices by a process that takes ω stages to mark the bad vertices. In stage 0 it marks the vertices whose theories are inconsistent. In stage i+1 it marks those vertices all of whose children if any have been marked at earlier stages (not marking if there are no children) and all vertices labelled with a theory containing $u\models\langle a\rangle p$ with no descendant labelled with a theory in which $u\models\langle a\rangle p$ closes. Then the root is bad if and only if it is marked by this procedure. (We defer proof of correctness of the marking procedure until we have the tableau in its final form.) This is our basic marking procedure, almost in its final form despite the fact that a quite different kind of tableau will be used ultimately.

In the case of propositional calculus, at this point in the development there are no further obstacles to constructing a tableau in a number of steps of computation (of some reasonable machine) bounded by an exponentially growing function of the length of the formula under test. In the case of non-standard PDL semantics the effectiveness situation is complicated by condition $\langle\ni\rangle_{+}$, which on the face of things call for infinitely many children when the selected s is $u\models\langle a\rangle p$.

Dealing with chains

We modify the rule $\langle\ni\rangle$ used in constructing tableaux to:

$\langle\ni'\rangle$ $\qquad u\models\langle a\rangle p\ni t \vdash u\models\langle a\rangle p.$

That is, $u\models\langle a\rangle p$ now calls only for the first step of a derivation of a, not the whole derivation. With this modification we lose the completeness property of tableaux that every path yields a theory that is Hintikka with respect to the unmodified conditions. On the other hand we do retain the soundness property that for every Hintikka y such that $x_0\subseteq y$ some path yields a subset of y. This is because of the "self-propagating" nature of intermediate links of a chain $u\models\langle a\rangle p\ni s_1, s_1\ni s_2, \ldots, s_{k-1}\ni v\models p$. Every s_i except possibly the final $v\models p$ is of the form $u'\models\langle a'\rangle p'$, and hence there must be some path which eventually accumulates all the links of such a chain, by inspection of the semantics.

We restore completeness as follows. Define $u\models\langle a\rangle p$ to *complete* in x when $u\models\langle a\rangle p\ni^{*}v\models p \subseteq x$ for some v. Define x to be *chain-complete* when for every $u\models\langle a\rangle p \subseteq x$, $u\models\langle a\rangle p$ completes in x. Modify the path-oriented test to consider only paths the union of whose labels is chain-complete. The modified test can now be seen to be sound and complete, as the tableau overlooks no possible way of completing a chain.

The corresponding modification to our vertex-oriented test is to add a third condition in the definition of *bad.*

(iii) $u\models\langle a\rangle p$ is in the theory labelling V but every descendant of V in whose theory $u\models\langle a\rangle p$ completes is bad.

We leave it to the reader to verify that this third condition restores Lemma 3.2 when $\langle\ni'\rangle$ replaces $\langle\ni\rangle$.

Despite our simplification of $\langle\ni\rangle$ to $\langle\ni'\rangle$ there still remains the non-constructive aspect of the choices for v when $\langle\ni'\rangle$ is being applied to $u\models\langle A\rangle p$ to yield $u\models\langle A\rangle p \ni v\models p$. We deal with this by observing that renaming some state throughout a theory cannot make an inconsistent theory consistent, although it may make a consistent theory inconsistent (e.g. if v is renamed to u when the theory contains $u\models p$ and $v\models\sim p$). This then permits us to satisfy $\langle\ni'\rangle$ for $u\models\langle A\rangle p$ with exactly one descendant, in which the new theory is the old together with $u\models\langle A\rangle p \ni v\models p$ for some v not appearing higher in the tree. Clearly this smaller tableau does not compromise the completeness of the test. To see that soundness is preserved we modify the original soundness proof so that for any consistent y containing x_0 there exists a path yielding a theory z which can be made a subset z' of y by renaming states, as the reader may verify by induction. Now if z is inconsistent so is z', whence so is y, a contradiction. So if x_0 is satisfiable some path yields a consistent Hintikka theory. This disposes entirely of difficulties associated with $\langle\ni'\rangle$.

Finiteness

We have now accomplished as much as is possible without the following theorem, which we need to make our tableau construction into an algorithm that always terminates, and to make our marking procedure both effective and terminating.

Theorem 3.3. If r is of length n, $|\{p|\exists p(u\models p \text{ appears in a tableau for } r)\}| \leqslant n$.

This is essentially lemma 3.2 of [12], to which we refer the reader for a proof.

This leads us to the observation that, for any u, at most n distinct statements $u\models p$ can appear in a tableau for a formula r of length n. A second observation is that any statement $v\models p \neq w\models r$ in the tableau can be attributed to one of only three sources: $u\models\langle A\rangle p\ni v\models p$, $u\models[A]p$, or some statement naming no state other than v. We shall arrange things so that these three sources are processed in three batches in the order given. In this way all statements of the form $u\models p$ and $u\models p\ni t$ will be formed while processing these three batches. Furthermore, processing of $u\models\langle p\ni v\models p$ and $u\models[A]p$ will be left until the end of the third batch, as they belong to the first and second batches respectively of other states. We postpone a more specific account to after the next step.

Lean Tableaux

Our next objective is to reduce the theories in the tableau to the point where the only statements in a theory are of the form $u\models p$ and $u\models p\ni t$, for fixed u; we call such a theory a *theory of* u.

To achieve this requires not merely reducing the theories but also splitting a simple vertex into as many vertices as are required for each to carry a theory of a single state and still have the new tableau retain the information in the old tableau.

We make these vague notions more precise by introducing the concept of a full theory, which is one that is "as Hintikka as a theory of a state can be." More formally, a theory x of state u is *full* when $x\vdash s$ for all $s\in x$ except those of the form $u\models\langle A\rangle p\ni v\models p$ and $u\models[A]p$, and otherwise is *partial.* A full (partial) vertex is one so labelled.

We introduce a new kind of tableau, the *lean tableau*, in which $x_i \subseteq x_{i+1}$ is no longer guaranteed for consecutive theories along a path. Lean tableaux enjoy the following properties.

(i) The root as before is labelled with $u \models r$, u arbitrary.

(ii) For each vertex V labelled with a partial theory x there exists some $s \in x$ not of the form $u \models \langle A \rangle p \ni v \models q$ or $u \models [A]p$ such that the descendants of V are each labelled with some minimal theory z such that $z \vdash s$ and $x \subseteq z$, and every such minimal theory labels some descendant.

(iii) For each vertex V labelled with a full theory x each statement $u \models \langle A \rangle p \ni v \models p \in x$ corresponds to a descendant of V whose theory is $\{v \models p\} \cup \{v \models q \mid u \models [A]q \in x\}$.

Note that lean tableaux contain no transitions. Moreover, a partial vertex has either one or two children while a full vertex may have up to n children where n is the length of r.

We now describe a test for satisfiability in terms of lean tableaux. The test is just a straightforward modification of the one we used on ordinary tableaux. A *fat* path in a lean tableau is a maximal set of paths such that any two paths have a common initial segment terminating in a full vertex. Thus for any full vertex appearing in a fat path, all the children of that vertex also appear in the path, while for any partial vertex appearing, exactly one child also appears. The point of fat paths is that they correspond to ordinary paths in ordinary tableaux.

Lemma 3.4. In a lean tableau, if the union of the labels of a fat path is chain-complete it is a Hintikka theory.

Proof. The construction of lean tableaux is such that only nonstandard conditions [A], $\langle \ni \rangle$ and $\langle A \rangle$ might be violated. $\langle \ni \rangle$ is taken care of explicitly by requiring chain-completeness. Now for each $u \models \langle A \rangle p \ni v \models p$ in the union add $u\langle A \rangle v$. This suffices for condition $\langle A \rangle$ as property (iii) of lean tableaux supplies $v \models p$. Finally, for each pair of states u,v appearing in the union such that $u\langle A \rangle v$ is not in the theory add $u\langle \sim A \rangle v$. Then for every $u \models [A]p$ and for every v appearing in the theory, one of $u\langle \sim A \rangle v$ or $v \models p$ is also in the theory, by property (iii) of lean tableaux. ∎

Lemma 3.5. For every Hintikka theory y containing r, any lean tableau for r contains a fat path the union of whose labels is a chain-complete subset of y to within renaming of states.

Proof. The construction of such a path parallels the construction in the ordinary tableau case, except that (i) instead of a sequence of vertices being developed, each labelled with a subset of y to within renaming, we have an advancing frontier of vertices, and (ii) the induction hypothesis must be that the union of the labels of all vertices seen thus far is a subset of y to within renaming (since we no longer have $x_i \subseteq x_{i+1}$ all along a path, even if x_i is taken to be the union of the frontier's labels at the i-th step). ∎

We immediately infer:

Corollary 3.6. Given a lean tableau for r, r is satisfiable if and only if the union of the labels of some fat path is consistent and chain-complete.

The Lean Procedure

This path oriented test can be translated into a vertex oriented test as for ordinary tableaux, the *lean procedure.* The procedure is to mark all the inconsistent vertices at stage 0, then at stage i+1 mark all full vertices with a child marked at stage i, all partial vertices with all children marked at stage i or before, and every vertex containing some u⊨⟨a⟩p not linked by a chain of ϶ statements at vertices (anywhere in the tableau) unmarked at stage i leading to v⊨p in an unmarked vertex.

Lemma 3.7. The lean procedure leaves the root of a tableau for r unmarked if and only if r is satisfiable.

Proof. Suppose r is satisfiable. Then the union of some fat path is a consistent chain-complete theory. Now consider the first stage in the marking process at which a vertex on that path was marked. If it was marked on account of inconsistency then the union cannot be inconsistent. If it was marked as a full vertex with marked child then this cannot the first stage at which a vertex on this path was marked. If it was marked as a partial vertex with only marked children then again this cannot be the first stage. (Note that in a lean tableau every partial vertex has at least one child.) If it was marked for not being linked to v⊨p via unmarked vertices then we use the fact that in the union of the labels of a fat path all statements of the form u⊨p and u⊨p϶t label vertices of a single ordinary path within that fat path and going from a child of a full vertex to another full vertex. Hence all constituents of a chain starting u⊨⟨a⟩p϶... that are in the theory of u can be found in the full vertex at the end of a path containing the occurrence of u⊨⟨a⟩p responsible for the marking. This chain must continue in exactly one of the children of this full vertex, and so on until completion. The whole chain (to be precise, an occurrence of every link of the chain) then appears entirely within the given fat path, whence if there is a link of the chain in the fat path no occurrence of which is at an umarked vertex, again we cannot be at the first stage when this fat path was marked. Hence at no stage can any vertex of this fat path, including the root, be marked.

Conversely, suppose the root is not marked at stage i for any finite i. Construct a fat path as follows. Initialize the set S of vertices to contain just the root. Proceeding by stages, at each stage, add to S all the children of each full vertex in S, and an unmarked child of each partial vertex in S not already having a child in S. Whenever a vertex labelled with u⊨⟨a⟩p is added to S, add to S some full vertex below that vertex that contains as much of the promised chain for a⊨⟨a⟩p as appears in the theory of u (this full vertex must exist), along with all vertices between the two vertices, then follow the chain into the next state and continue adding states until the end of the chain is reached, a finite process. (As we have described it, an entire chain is added as part of a single stage.) By the marking process it should be evident that every vertex in S remains unmarked. Hence at the end the theory labelling that path must be consistent (since inconsistent formulae must belong to the theory of the same state and hence appear together in the full vertex of the part of the path going through that full state) and chain-complete (by the construction). Hence r must be satisfiable. ∎

We define two vertices to be equivalent when their theories are the same to within renaming of states, and two trees to be equivalent when their roots are equivalent and to every subtree of one root corresponds an equivalent subtree of the other. Since each theory can have at most n formulae to within renaming, there can be at most 2^n mutually inequivalent vertices. We also observe that if equivalent vertices always have the same set of labels on their children (easily arranged by having a method of choosing s that depends only on the particular partial theory being extended) then two trees with equivalent roots are equivalent.

Inspection of the marking algorithm reveals that equivalent trees will be marked identically, whence equivalent vertices will all be marked simultaneously. Hence the marking cannot proceed for more than 2^n stages without reaching some stage at which nothing new is marked, after which nothing new is ever marked.

This suggests that we *filter* the tableau (the term used by modal logicians for the process used in the proof in [12] of the finite model theorem). That is, we identify equivalent vertices to yield a directed graph, instead of a tree, having at most 2^n vertices. Such a graph can be effectively constructed by a machine.

To construct the graph using a random-access machine it suffices to construct a lean tableau, represented using bit vectors of length n to encode each theory, and to keep an eye out for repeated states. Since our algorithm is going to use exponential storage anyway, one may as well set aside an array of 2^n locations indexed by the bit vector representation of theories to represent which theories (modulo state names) have been encountered and where they are in storage. All this can be done in time proportional to 2^n times some small polynomial in n.

The marking procedure applied to this filtered tableau will at stage i mark exactly those vertices that are the images of vertices in the unfiltered tableau marked at stage i. Hence the root of the filtered graph will be marked if and only if the root of the unfiltered graph was marked by the procedure.

While the marking time is not linear in the number of vertices in the graph, it is clearly proportional to a small polynomial in the number of vertices, establishing our upper bound of c^n for some c. The non-linearity is due entirely to checking for chains. A crude method of checking would be to compute the transitive closure of the chain relation on the $n2^n$ facts appearing in the graph after each stage of checking, which would lead to c=16 if a cubic-time transitive-closure algorithm were used. This can be reduced to c=8 by taking advantage of the fact that only two links can have the same first statement, whence their are only $2n2^n$ links. Further reduction should be possible. As a practical matter, one can expect the number of vertices in the graph to be quite small typically (e.g. in a program verification context), whence the important issue is whether one can reduce the marking time to say the square or better of the number of vertices, an issue we do not go into here.

The marking procedure as described requires the marking to be done in stages, with all marks made in stage i+1 depending only on marks made at or before stage i. This is a little inconvenient to program, and the simpler procedure of having each mark depend on all marks made up till now, including those made at the current stage, will also work. This can be seen to be true by carrying out the proof of correctness of the unfiltered marking procedure modified so that at each stage only one equivalence class of vertices is marked. The procedure still halts in finitely many steps since there are only finitely many equivalence classes.

4. MISCELLANEOUS REMARKS

DL and Algorithmic Logic

We developed dynamic logic in 1974 while teaching a course on Floyd-Hoare axiomatics [34], feeling a need for a treatment of the subject in the style to which mathematicians are accustomed, and described it at an ACM symposium in 1976 [36]. (The name "dynamic" was first used in [17].) Unknown to us at the time, the Polish logician Salwicki had already given an admirable treatment of the subject in 1970 [39] for the case of deterministic programs, calling his treatment algorithmic logic. Our discovery of Salwicki's work lessened considerably our estimation of the extent of the contribution made by the development of dynamic logic. However, the dynamic logic treatment differs from the algorithmic logic treatment in three important respects, each of which we feel makes dynamic logic the better of the two treatments.

First, algorithmic logic confines itself specifically to deterministic programs, while dynamic logic deals with nondeterministic programs, thereby subsuming algorithmic logic. The syntax and semantics of algorithmic logic would take considerable restructuring to extend algorithmic logic to cope gracefully with nondeterminism. It seems unlikely to us to be possible to so extend algorithmic logic other than by a wholesale replacement of its constructs by those of dynamic logic. That the extension is necessary is evident from the recent but very strong interest in reasoning about nondeterministic action [1,8,9,14,18,21,38]. Dijkstra [9] in particular makes an excellent and delightfully illustrated case for nondeterministic programming as a valuable tool for every programmer, a case which our own programming experience bears out.

Second, algorithmic logic introduces a number of mathematically novel constructs taken from the programming milieu whereas dynamic logic has no mathematical novelties save assignment and the mathematically trivial notion of test, and even assignment is absent from PDL. Instead dynamic logic takes its constructs from modal logic and Tarski's calculus of binary relations, permitting greater utilization of existing theory and reducing the cost of entry into the program logic arena for mathematicians already expert in those areas. (The contributions made to dynamic logic by the modal logicians K. Segerberg and D. Gabbay supply concrete examples of this phenomenon.)

Third, dynamic logic is eminently suited to reasoning about action (change of state) in general, not merely the processes encountered in computer programming. Thus it is just as applicable to reasoning about change of state in natural language discourse analysis as in computer programming. For example, knowing the validity of "If your TV set won't work, then after you kick it still won't work" one can infer the validity of "If your TV set won't work then no matter how often you kick it it still won't work," using the rule, from $p\supset[a]p$ infer

$p \supset [a^*]p$. It was with such generality in mind that the term "dynamic" was chosen to describe the logic, to maintain neutrality towards particular domains of application of logics of change of state.

Despite our feeling that dynamic logic is the better of the two logics, we nevertheless feel also that algorithmic logic represents a substantial milestone in the history of logics of programs, both in supplying a mathematically excellent treatment of program logics and in starting a flourishing school of algorithmic logic in Warsarw. It is unfortunate that the work failed to catch earlier the attention of the bulk of the western world. (On the other hand, if it had we might not have felt the need for dynamic logic sufficiently strongly as to have invented it, and so would have lacked a framework within which to make the sort of discoveries reported here.)

Another contribution along the above lines was recently made by Constable [6]. Constable's logic is considerably closer to algorithmic logic than to dynamic logic. His language however departs from both propositional dynamic logic and algorithmic logic in permitting naming of and quantification over states. Constable has shown that validity is decidable in this language. More recently Parikh [33] has shown that an even richer language, quantifying over both states and state sequences (of ongoing computations), and permitting discussion of relative order of states within sequences, is decidable, using Rabin's decision method for weak SnS.

Software Engineering Considerations

DL has received a mixed reception from what one might characterize as the "hard hat" end of the program verification spectrum of researchers. A common reaction is that DL might be cute theoretically, but it is not the robust sort of product demanded by practical considerations in the real world of program verification. In fact no such objection can be sustained because dynamic logic is no more than a cleaned up and extended version of Hoare's logic, which is certainly taken these days as a *sine qua non* of verification. Thus in any proof of $p\{a\}q$, there is a corresponding proof in DL of $p \supset [a]q$. The fact that DL goes on to handle in stride other issues such as termination can hardly be seen as a criticism of DL over Hoare's logic.

There has also been the criticism that no substantial DL proofs have ever been given. This criticism is unreasonable of any system into which it is straightforward to translate substantial proofs that already exist in other logics, e.g. Hoare's. The process of translating Hoare proofs into DL proofs is a purely mechanical exercise given a table of abbreviations and rule derivations. Such tables are readily supplied for all the logics mentioned above.

More sensible however is not translating proofs at all but rather considering DL to *supply the base language for an extensible logic of programs*, where extensions are carried out by (i) defining new constructs from old, (ii) taking theorems as axioms, and (iii) taking derived rules as rules. Factoring out a base language in this way is sound engineering. The bulk of the implementation effort can be focused on the base language, leaving the implementation of the extended language as a relatively easy task. The smaller and more modular the base component can be made, the simpler its implementation becomes.

It is possible to overdo the minimization of the base language, and one must take care that the extended language remain easy to derive from the base language. It should be clear from our examples of definitions in section 1 that DL observes this constraint admirably.

This methodology is a valuable tool in metamathematics, and there is no reason why it should not be an equally effective tool in "meta-programming" (writing of program verification systems, program synthesizers, and for that matter compilers, interpreters, and any other programs that manipulate programs).

We are at present engaged in implementing mechanical proof systems to evaluate the extent to which this technique helps. One such system incorporating early ideas about DL, and implemented in LISP on a PDP-10, has been in operation since August 1977, and we are now building a second system to incorporate and further improve the developments reported in this paper.

Using the first system, the largest DL proof that has been subjected to successful mechanical verification to date is a proof of the total correctness of the Knuth-Morris-Pratt pattern-matching algorithm [23]. This proof consisted of some twenty theorems stated in DL, and was certified in 45 seconds. The proof supplies an excellent illustration of how one can take advantage of DL's "primitive" primitives in structuring the proof for better understandability to a human. This work will be reported on in the near future.

Bibliography

1. J. W. de Bakker, Semantics and Termination of Nondeterministic Recursive Programs, in "Automata, Languages and Programming," *3* (ed. Michaelson, S. and R. Milner), 435-477, Edinburgh University Press, Edinburgh, Scotland, 1976.

2. S. K. Basu and R. T. Yeh. Strong Verification of Programs. *IEEE Trans. Software Engineering, SE-1, 3* (1975), 339-345.

3. Berman, F. and M. Paterson. Test-Free Propositional Dynamic Logic is Strictly Weaker than PDL. T.R. 77-10-02, Dept. of Computer Science, Univ. of Washington, Seattle, Nov. 1977.

4. E. W. Beth, "The Foundations of Mathematics," North Holland, 1959.

5. A. E. Chandra and L. J. Stockmeyer, Alternation, *17th IEEE Symp. on Foundations of Comp. Sci.*, 98-108, Oct. 1976.

6. R. L. Constable, On the Theory of Programming Logics, *Proc. 9th Ann. ACM Symp. on Theory of Computing*, 269-285, Boulder, Col., May 1977.

7. S. A. Cook, Characterization of Pushdown Machines in Terms of Time Bounded Computers, *J. ACM, 18* (1971), 4-18. 1971.

8. E. W. Dijkstra, Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Comm. ACM, 18* (1975), 8, 453-457.

9. E. W. Dijkstra, "A Discipline of Programming," Prentice-Hall, Englewood Cliffs, N.J., 1976

10. C. C. Elgot, Structured Programming With and Without GO TO Statements, *IEEE Transactions on Software Engineering, SE-2*, (1976), 1, 41-53, March 1976.

11. E. Engeler, Algorithmic properties of structures, *Math. Sys. Thy. 1* (1967), 183-195.

12. M. J. Fischer and R. E. Ladner, Propositional Modal Logic of Programs, *Proc. 9th Ann. ACM Symp. on Theory of Computing*, 286-294, Boulder, Col., May 1977.

13. M. J. Fischer and R. E. Ladner, Propositional Dynamic Logic of Regular Programs, Manuscript, Dept. of Comp. Sc., U. of Wash., Seattle, Wash. c. Oct. 1977.

14. R. W. Floyd, Nondeterministic Algorithms, *J. ACM, 14*, 4, 636-644, Oct. 1967.

15. D. Gabbay, Axiomatizations of Logics of Programs, Manuscript, under cover dated Nov. 1977.

16. G. Gentzen, Untersuchungen ueber das Logische Schliessen, *Math. Zeitschr, 39* (1934-5), 176-210, 405-431. (English tr.: Investigations into Logical Deduction, in "The Collected Papers of G. Gentzen," 69-131, 1969.)

17. D. Harel, A. R. Meyer and V. R. Pratt, Computability and Completeness in Logics of Programs, *Proc. 9th Ann. ACM Symp. on Theory of Computing*, 261-268, Boulder, Col., May 1977.

18. D. Harel and V. R. Pratt, Nondeterminism in Logics of Programs, *Proc. 5th Ann. ACM Symp. on Principles of Programming Languages*, 203-213, Tucson, Arizona, Jan. 1978.

19. K. J. J. Hintikka, Form and content in quantification theory, *Acta Philosophica Fennica, 8*, 7-55. 1955.

20. C. A. R. Hoare, An Axiomatic Basis for Computer Programming, *Comm. ACM 12* (1969), 576-580.

21. C. A. R. Hoare, Some Properties of Predicate Transformers, *J. ACM, 25* (1978), 3, 461-480.

22. M. Jazayeri, W. F. Ogden and W. C. Rounds, The Intrinsically Exponential Complexity Problem for Attribute Grammars, *Comm. ACM, 18* (1975), 12, 697-706. 1975.

23. D. E. Knuth, J.H. Morris and V.R. Pratt, Fast Pattern Matching in Strings, *SIAM J. Comp., 6* (1977), 2, 323-350.

24. D. Kozen, On parallelism in Turing machines, *17th IEEE Symp. on Foundations of Comp. Sci.*, 89-97, Oct. 1976.

25. S. A. Kripke, Semantical considerations on Modal Logic, *Acta Philosophica Fennica*, 83-94, 1963.

26. S. A. Kripke, Semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschr. f. Math. Logik und Grundlagen d. Math.*, *9* (1963), 67-96.

27. S. D. Litvintchouk, and V.R. Pratt. A Proof-checker for Dynamic Logic. *Proc. 5th Int. Joint Conf. on AI*, 552-558, Boston, Aug. 1977.

28. E. Mendelson, "Introduction to Mathematical Logic," Van Nostrand, N.Y., N.Y., 1964.

29. A. R. Meyer, Equivalence of DL, DL+ and ADL for Regular Programs with Array Assignments, Internal report, MIT, August 1977.

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

31. J. Misra, Prospects and Limitations of Automatic Assertion Generation for Loop Programs, *SIAM J. on Comp. 6* (1977), 4, 718-729.

32. R. Parikh, A Completeness Result for PDL, *Symp. on Mathematical Foundations of Computer Science*, Zakopane, Poland, Sept. 1978.

33. R. Parikh, Second Order Process Logic, *19th IEEE Symposium on Foundations of Computer Science*, Oct. 1978.

34. V. R. Pratt, Semantics of Programming Languages, Lecture notes for 6.892, Fall 1974, M.I.T.

35. V. R. Pratt and L.J. Stockmeyer, A Characterization of the Power of Vector Machines, *JCSS, 12* (1976), 2, 198-221.

36. V. R. Pratt, Semantical Considerations on Floyd-Hoare Logic. *Proc. 17th Ann. IEEE Symp. on Foundations of Comp. Sci.*, 109-121, 1976.

37. V. R. Pratt, A Practical Decision Method for Propositional Dynamic Logic, *Proc. 10th Ann. ACM Symp. on Theory of Computing*, 326-337, San Diego, Calif., May 1977.

38. W. P. de Roever, Dijkstra's Predicate Transformer, Nondeterminism, Recursion, and Termination, I.R.I.S.A., Publication Interne No.37. 1976.

39. A. Salwicki, Formalized Algorithmic Languages, *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys., 18* (1970), 5, 227-232.

40. K. Segerberg, A Completeness Theorem in the Modal Logic of Programs. Preliminary report, *Notices of the AMS*, *24* (1977), 6, A-552.

41. R. M. Smullyan, "First-Order Logic," Springer-Verlag, Berlin, 1968.

42. R. M. Smullyan, Trees and Nest Structures, *J. Symb. Logic*, *31* (1966), 303-321.

43. B. Wegbreit, The synthesis of loop predicates. *Comm. ACM*, *17* (1974), 2, 102-112.

44. K. Winklmann, Equivalence of DL and DL$^+$ for regular programs, Internal report, Lab. for Comp. Sci., M.I.T. 1978.