JOURNAL OF COMPUTER AND SYSTEM SCIENCES **30**, 162–178 (1985)

A Probabilistic PDL

DEXTER KOZEN

IBM Research, Yorktown Heights, New York 10598

Received May 1, 1983; revised January 10, 1985

In this paper we give a probabilistic analog PPDL of Propositional Dynamic Logic. We prove a small model property and give a polynomial space decision procedure for formulas involving well-structured programs. We also give a deductive calculus and illustrate its use by calculating the expected running time of a simple random walk. © 1985 Academic Press, Inc.

1. INTRODUCTION

This paper deals with the problem of defining a fully compositional, exogenous formalism for reasoning about probabilistic programs at the propositional level. Apart from related work in first-order and endogenous logics [SPH, FH, LS, Pn], previous approaches to this problem have not met with the same level of success as has Propositional Dynamic Logic (PDL) [FL]. Ramshaw [Ra] gave a Hoare-like logic, but observed that even the if-then-else rule was incomplete. Reif [Re] gave a logic that was not expressive enough to define if-then-else; moreover, one of its proof rules was later shown unsound [FH]. Makowsky and Tiomkin [MT] gave an infinitary system and an infinitary completeness result. Parikh and Mahoney [PM] studied the equational properties of probabilistic programs. Feldman [F] gave a less expressive version of the logic of [FH], though still with quantifiers, and proved decidability by reduction to the first-order theory of the reals.

Previous approaches to this problem, with the exception of [PM], attempted to deal with probability *truth-functionally*, whereas the natural semantics is *arithmetic* [K]. For example, the Hoare-style if-then-else rule of [Ra] is incomplete because absolute propositional information about the probabilistic behavior of programs p and q, combined truth-functionally, does not yield complete propositional information about the behavior of **if** A **then** p **else** q. Under the natural semantics, however, **if** A **then** p **else** q is a *sum* $A?p + \neg A?q$, not a join [K]; this indicates that the operator $+$ is more appropriate for dealing with the if-then-else construct than $\vee$.

This example illustrates the dichotomy between two forms of possibility, which we will call the *nondeterministic form* and the *probabilistic form*, respectively. In the former, events are either possible or impossible, with no further distinction. In the latter, events occur according to a probability distribution; even if the distribution is unknown, its very existence affects the theory. One might attempt to equate "possible" in the nondeterministic form with "nonzero probability" in the probabilistic form, but this correspondence goes only so far: for example, the program

$$x := \mathit{flip};\ \textbf{while}\ x = \mathit{heads}\ \textbf{do}\ x := \mathit{flip}$$

is probabilistically total, since it halts with probability 1, but not nondeterministically total, since it has an infinite computation path.

Unfortunately, almost all of our logical apparatus belongs to the nondeterministic form. The usual logical connectives and the existential quantifier are clearly nondeterministic in nature. We must therefore be prepared to depart radically from conventional logic in order to accommodate probability in a satisfactory way.

The approach of this paper is to replace the truth-functional propositional operators with analogous arithmetic ones, which are more closely aligned with the probabilistic form. We are led to consider a formal system which we call *Probabilistic Propositional Dynamic Logic* (PPDL), although the "propositional" is really a misnomer. Each logical construct in PDL has an arithmetic counterpart in PPDL. For example, *propositions* A generalize to *measurable functions* f; they are combined linearly, as in $af+bg$, not truth-functionally. *States* s generalize to *measures* μ. *Programs* p are interpreted as real-valued functions. The program operator $\vee$ is replaced by $+$. The program operator $*$ represents not an infinite union of binary relations as in PDL, but rather an infinite sum of functions. The modal construct $\langle p \rangle$ is a measurable function transformer; $\langle p \rangle A$ is a measurable function which when applied to input state s yields the probability that program p halts in a state satisfying A. Finally, the notion of satisfiability ($s \models A$) generalizes to an integral $\int f\, d\mu$, a real-valued function giving the probability that (generalized) state μ satisfies (generalized) proposition f.

The proof rules of PDL have natural analogs in PPDL, as well. For example, the usual PDL axioms for $*$ are equivalent to the axiom and rule

$$\langle p^* \rangle A = A \vee \langle pp^* \rangle A$$

$$A \vee \langle p \rangle B \leqslant B \rightarrow \langle p^* \rangle A \leqslant B.$$

In PPDL, we have the analogous rules, for $0 \leqslant f$,

$$\langle p^* \rangle f = f + \langle pp^* \rangle f$$

$$f + \langle p \rangle g \leqslant g \rightarrow \langle p^* \rangle f \leqslant g.$$

2. Three Equivalent Semantics of Probabilistic Programs

In this section, familiarity with basic measure theory and topology is assumed. Definitions of terms such as *measure*, *measurable set*, *measurable function*, *σ-algebra*, and *integral* can be found in [H].

Let S be a set of states and Σ a σ-algebra of measurable sets on S. Let B be the space of finite measures on (S, Σ) and F the space of bounded measurable functions on (S, Σ). $\mu, \nu, \ldots,$ and $f, g, \ldots,$ denote elements of B and F, respectively. The integral $\int f\,d\mu$, which we will sometimes denote (μ, f), is a bilinear function $B \times F \to \mathbb{R}$. The integral induces a weak topology on B and F, namely the weakest topology making $\int f\,d\mu$ continuous in f and μ.

There are three equivalent ways to interpret a program over (S, Σ). The one most analogous to binary relations in PDL is the *Markov transition* or *measurable kernel* [SPH, Rev], which is a function

$$p: S \times \Sigma \to \mathbb{R}$$

satisfying (1) and (2) below. Intuitively, $p(s, A)$ is the probability that the program p, starting in state s, will halt in a state satisfying A. For finite or countable S, p can be represented by a Markov transition matrix giving the probability that s goes to t under p for each pair of states s, t.

By definition, Markov transitions must satisfy the properties:

(1) for fixed $A \in \Sigma$, the function $\lambda s \cdot p(s, A)$ is an element of F,

(2) for fixed $s \in S$, the function $\lambda A \cdot p(s, A)$ is an element of B.

These properties allow program composition by integration up the middle:

$$(p; q)(s, A) = \int_{t \in S} q(t, A)\, p(s, dt).$$

This formula reduces to ordinary matrix multiplication in the finite or countable case.

EXAMPLE. In the BASIC programming language, numeric variables range over $\mathbb{R}$. The set of states S would be $\mathbb{R}^n$, the set of valuations of n program variables $x_1, \ldots, x_n$ over $\mathbb{R}$. Σ would be the family of Lebesgue measurable subsets of S. A deterministic assignment

$$10 \ \mathbf{let}\ x_1 = x_2 + 3$$

would be modeled by a (deterministic) Markov transition $p: S \times \Sigma \to \mathbb{R}$ defined by

$$\begin{aligned} p(s, A) &= 1 \qquad \text{if } t \in A, \\ &= 0 \qquad \text{otherwise,} \end{aligned}$$

where t is the valuation obtained from s by changing the value of variable x_1 to 3 plus the value of variable x_2. A random assignment

$$10 \ \mathbf{let}\ x_1 = rnd$$

gives a random real number in the interval $[0, 1)$ with uniform probability; this would be modeled by a Markov transition $p: S \times \Sigma \to \mathbb{R}$ such that, if $s = \langle a_1, ..., a_n \rangle$ and $A = A_1 \times \cdots \times A_n$, where $a_i \in \mathbb{R}$ and the A_i are Lebesgue measurable subsets of $\mathbb{R}$, then

$$\begin{aligned} p(s, A) &= \lambda(A_1 \cap [0, 1)) && \text{if } a_i \in A_i,\ 2 \leqslant i \leqslant n, \\ &= 0 && \text{otherwise,} \end{aligned}$$

where λ is the Lebesgue measure on $\mathbb{R}$. (The value of p on elements of Σ of the form $A_1 \times \cdots \times A_n$ determines the value of p on measurable sets not of this form.)

For μ a measure, f a bounded measurable function, and p a program, the properties (1) and (2) also allow the definition of the measure $\mu\langle p \rangle$ and the measurable function $\langle p \rangle f$ by integration on either side of p:

$$(\mu\langle p \rangle)(A) = \int p(s, A)\, \mu(ds),$$

$$(\langle p \rangle f)(s) = \int f(t)\, p(s, dt).$$

The map $\mu \to \mu\langle p \rangle$ is exactly the measure-transformer semantics of [K], in which a program p maps an input measure μ *forward* into an output measure $\mu\langle p \rangle$. It is the unique linear, continuous map $B \to B$ extending p. Moreover, any such map, restricted to point masses, specializes to a Markov transition; thus these two semantics are equivalent. The map $f \to \langle p \rangle f$ can be thought of as a generalized predicate transformer, mapping a measurable function f *backward* into a measurable function $\langle p \rangle f$. For measurable set A (or rather, its characteristic function), the value of $\langle p \rangle A$ on input s intuitively gives the probability that p, started in state s, halts in a state satisfying A. Again $f \to \langle p \rangle f$ is the unique linear, continuous map $F \to F$ extending p, and any such map, restricted to characteristic functions of measurable sets, specializes to a Markov transition. Thus all three semantics are equivalent. The latter two are related by the equation

$$(\mu\langle p \rangle, f) = (\mu, \langle p \rangle f),$$

which says intuitively that the probability that the output condition f is satisfied by the output measure $\mu\langle p \rangle$ is the same as the probability that the precondition $\langle p \rangle f$ is satisfied by the input measure μ. The proof of this equation is essentially Fubini's theorem.

The equivalence of these semantics is a consequence of the functional duality between F and B, i.e., $F \cong B^*$ and $B \cong F^*$, where $B^*(F^*)$ denotes the space of linear, continuous maps $B \to \mathbb{R}(F \to \mathbb{R})$ (see [S]). The topology on B and F is also the appropriate topology for discussing the convergence of effects of approximants of a while-loop to the effects of the while-loop. With respect to this topology, the step functions are dense in F and the discrete measures are dense in B. This fact gives an easy proof of Theorem 6.1 of [K].

3. Syntax and Semantics of PPDL

Terms of PPDL are of two types: *programs* and *measurable functions*. P, Q,..., denote primitive program symbols and p, q,..., denote program terms. F, G,..., denote primitive measurable function symbols and f, g,..., denote measurable function terms. There is a distinguished primitive function symbol 1. A, B,..., denote Boolean combinations of primitive function symbols, using the operators $\vee$, $\wedge$, $\neg$ to be defined below. These will be called *propositions*. Rational numbers are denoted a, b, c, d,....

Compound program terms are formed inductively, by means of the operations

(1)	$ap + bq$, provided $0 \leqslant a, b$	(positive linear combination)
(2)	$p; q$ (or pq)	(composition)
(3)	$B?$	(test)
(4)	p^*	(iteration).

Compound function terms are formed inductively, by means of the operations

(5)	$af + bg$	(linear combination)
(6)	Bf	(pointwise multiplication)
(7)	$\langle p \rangle f$	(eventuality).

In addition, we have the following defined operations:

$A \vee B = A + B - AB$
$\neg f = 1 - f$
$0 = \neg 1$
$\textbf{skip} = 1?$
$\textbf{fail} = 0?$
$\textbf{if } B \textbf{ then } p \textbf{ else } q = B?p + \neg B?q$
$\textbf{while } B \textbf{ do } p = (B?p)^*; \neg B?$
$[p]f = \neg \langle p \rangle \neg f$

A program will be called *well-structured* if it is formed from primitive programs and tests using composition, if-then-else, and while-do.

A *formula* is an inequality $f \leqslant g$.

The semantics of the full version of PPDL requires extending the results of the previous section to unbounded and infinite functions, but we will restrict our attention below to functions involving only well-structured programs, which are always bounded.

A *model* $M = (S, \Sigma, M)$ consists of a set S of *states*, a σ-algebra Σ of *measurable sets* on S, and an interpretation M of the primitive programs P and functions F. P^M is a positive, total Markov transition on $S \times \Sigma$, i.e., one such that for any s and A, $p(s, A) \geqslant 0$ and $p(s, S) = 1$. F^M is a 0, 1-valued measurable function, i.e., the characteristic function of a measurable set. 1^M is always the constant function $\lambda s \cdot 1$. In practice, the set of states might be the set of valuations of program variables, a primitive program might be a deterministic assignment such as $x := x + y$ or random assignment $x := rnd$, and a primitive measurable function might be the characteristic function of a test such as $x \leqslant y$.

The interpretation M extends to compound program and function terms inductively:

(1) $(ap + bq)^M = ap^M + bq^M$

(2) $(p; q)^M = \lambda s, C \cdot \int_{t \in S} q^M(t, C)\, p^M(s, dt)$

(3) $B?^M = \lambda s, C \cdot C(s)\, B^M(s)$

(4) $(p^*)^M = \sum_{n=0}^{\infty} (p^n)^M$, where $p^0 = \mathbf{skip}$, $p^{n+1} = pp^n$

(5) $(af + bg)^M = af^M + bg^M$

(6) $(Bf)^M = B^M f^M$ (pointwise multiplication)

(7) $(\langle p \rangle f)^M = \langle p^M \rangle f^M$,

where in (7), $\langle p^M \rangle$ is the predicate transformer corresponding to the Markov transition p^M (see Sect. 2).

A formula $f \leqslant g$ is true in a model M if $f^M \leqslant g^M$ pointwise.

In the sequel, we often drop the superscript M and use the symbols p and f to stand for both a term and its interpretation.

4. Basic Properties

Below is a list of some valid properties. Some of these require the implicit proviso that the arguments are everywhere defined, in order to rule out trivial counterexamples involving nonconvergent * expressions:

(1) $\langle ap + bq \rangle f = a\langle p \rangle f + b\langle q \rangle f$ (linearity)

(2) $\langle p \rangle(af + bg) = a\langle p \rangle f + b\langle p \rangle g$ (linearity)

(3) $0 \leqslant f \rightarrow 0 \leqslant \langle p \rangle f$ (positivity of $\langle p \rangle$)

(4) $f \leqslant g \rightarrow \langle p \rangle f \leqslant \langle p \rangle g$ (monotonicity of $\langle p \rangle$)

(5) $\langle pq \rangle f = \langle p \rangle \langle q \rangle f$

(6) $\langle B? \rangle f = Bf$

(7) $[p] f = [p]\, 0 + \langle p \rangle f$

(8) $0 \leqslant f \rightarrow \langle p^* \rangle f = f + \langle pp^* \rangle f$

(9) $0 \leqslant f \rightarrow \langle p^* \rangle f = f + \langle p^* p \rangle f$

(10) $0 \leqslant f + \langle p \rangle g \leqslant g \rightarrow \langle p^* \rangle f \leqslant g$ (induction rule)

(11) $f \leqslant 1 \rightarrow \langle p \rangle f \leqslant 1$, for well-structured p

(12) $0 \leqslant B = BB \leqslant 1$.

A special case of the induction rule is the *while rule*

$$\neg Bf + B\langle p\rangle g \leqslant g \rightarrow \langle \textbf{while } B \textbf{ do } p\rangle f \leqslant g$$

for $f \geqslant 0$.

5. Expressiveness

Let p be well-structured. The measurable function $\langle p\rangle 1$ on input state s gives the probability that p halts, and $[p]0 = 1 - \langle p\rangle 1$ the probability that it does not halt. The formula $1 \leqslant \langle p\rangle 1$ says that p is probabilistically total, i.e., halts on all inputs with probability 1. The measurable function $[p]A$ on input state s gives the probability that p does not halt in $\neg A$. Axiom (7) of Section 4 says that this is equal to the probability that p does not halt at all ($[p]0$) plus the probability that it halts in A ($\langle p\rangle A$). From this it follows that the probabilistic counterpart of the Hoare partial correctness assertion

$$\{A\}\, p\, \{B\}$$

is the formula

$$A \leqslant [p]B$$

which says that if $A(s) = 1$, then

$$([p]0)(s) + (\langle p\rangle B)(s) = 1,$$

or in other words, if s satisfies A then with probability 1 either p does not halt or halts in B.

In any model, the partial sums of p^* are nondecreasing, since all programs are positive, but p^* may not converge to a finite value for all inputs. However, it can be shown that if p is well structured, then it is everywhere defined and takes values in $[0, 1]$. Moreover, any function term containing only well-structured programs represents a bounded function. For these reasons, it is tempting to take if-then-else and while-do as primitive, and throw * away. However, it would be a mistake to do so, because * can be used in practice to define certain useful programs and functions. For example, to calculate the expected running time of a program, one would modify the program to count each step in an integer variable c, then compute the expected value of c on output. This is usually done by integrating the unbounded function

$$\sum_{n=0}^{\infty} n\chi(c = n)$$

with respect to the output measure, where $\chi(c=n)$ is the characteristic function of the set of all states in which the value of c is n. This function can be expressed

$$\langle c := c-1^*; c \geqslant 1? \rangle\, 1$$

in PPDL:

$$\begin{aligned}\langle c := c-1^*; c \geqslant 1? \rangle\, 1 &= \sum_{n=0}^{\infty} \langle c := c-n; c \geqslant 1? \rangle\, 1\\ &= \sum_{n=0}^{\infty} \langle c \geqslant n+1? \rangle\, 1\\ &= \sum_{n=0}^{\infty} \chi(c \geqslant n+1)\\ &= \sum_{n=0}^{\infty} n\chi(c=n).\end{aligned}$$

Other moments can also be expressed; the nth moment is given by an expression of $*$-depth n. The expected running time of p on input distribution μ is the value of the integral

$$(\mu\langle p \rangle, \langle c := c-1^*; c \geqslant 1? \rangle 1) = (\mu, \langle p; c := c-1^*; c \geqslant 1? \rangle 1).$$

If in addition μ is computable by a simple probabilistic program q from any start state (e.g., a random graph on n nodes is computed by the program

$$\textbf{for } i, j \in \{1,..., n\} \textbf{ do } \tfrac{1}{2}(E(i,j) := 0 + E(i,j) := 1)),$$

then the expected running time is the value of the constant function

$$\langle q; p; c := c-1^*; c \geqslant 1? \rangle\, 1.$$

An example of a proof involving this device is given in Section 7.

6. Decision Procedure and Small Model Property

In this section we give a polynomial space decision procedure for formulas $f \leqslant g$ involving well-structured programs. Halpern and Reif [HR] give a similar complexity bound for well-structured, deterministic PDL.

Let PP and FF be fixed finite sets of primitive program and function symbols. First we build a skeleton T of a tree model of bounded out-degree, and show that every model is equivalent with respect to formulas over PP and FF to an instantiation of this skeleton.

An *atom* is a product of elements of FF or their negations, such that each $F \in FF$

appears exactly once, either as F or $\neg F$. Atoms are denoted R, S,.... For any proposition B over FF and distinct atoms R, S, either $R \leqslant B$ or $R \leqslant \neg B$, $RS = 0$, and $\Sigma R = 1$ in all models. Let the set of states of T consist of all programs of the form

$$R_0?P_1R_1?P_2R_2?\cdots P_kR_k?,$$

where R_i is an atom and $P_i \in PP$. T can be viewed as a forest with roots $R?$ and edges $(s, sPR?)$. The state $sPR?$ is call a *P-successor* of s. If $s = \cdots R?$ is a state of T, define $R(s) = R$.

An *instantiation* of this skeleton is a model whose states are the nodes of T, such that for primitive function $F \in FF$ and state s, $F(s) = 1$ iff $R(s) \leqslant F$, and for primitive programs $P \in PP$, $P(s, t) = 0$ unless t is a P-successor of s.

The following lemma was proved independently by Feldman [F].

LEMMA 1. *For any model M and state s of M, there exists an instantiation N of T and a root r of N such that $f^M(s) = f^N(r)$ for any f over FF and PP.*

Proof. Each state u of T is also a program, and $(\langle u\rangle\, 1)^M(s)$ is the probability that the program u halts in M, starting in state s. Define N as follows: for $P \in PP$, let

$$P^N(u, uPR?) = \frac{\langle uPR?\rangle\, 1^M(s)}{\langle u\rangle\, 1^M(s)},$$

or 0 if the denominator is 0. Let ι_s be the unit point mass on s in M, i.e., the measure with weight 1 on the point s and 0 elsewhere; ι_s is defined formally by

$$\begin{aligned} \iota_s(A) &= 1 \qquad \text{if } s \in A, \\ &= 0 \qquad \text{otherwise.} \end{aligned}$$

Let $\iota_{R_0?}$ be the unit point mass on $R_0?$ in N, where $R_0?$ is the unique atom such that $R_0^M(s) = 1$. Let t be a state of T. We show by induction on the length of t as a program that $\iota_{R_0?}\langle t\rangle$ is a point mass on t with weight $(\langle t\rangle\, 1)^M(s)$. Certainly,

$$\iota_{R_0?}\langle R_0?\rangle = (\langle R_0?\rangle\, 1)^M(s) = 1.$$

For any program P and atom S,

$$\iota_{R_0?}\langle tPS?\rangle = (\iota_{R_0?}\langle t\rangle)\langle PS?\rangle,$$

and by induction hypothesis, $\iota_{R_0?}\langle t\rangle$ is a point mass on t with weight $(\langle t\rangle\, 1)^M(s)$; then $(\iota_{R_0?}\langle t\rangle)\langle PS?\rangle$ is a point mass on $tPS?$, since the test $S?$ annihilates mass on other P-successors of t, and $(\iota_{R_0?}\langle t\rangle)\langle PS?\rangle$ has weight

$$(\langle t\rangle\, 1)^M(s)\, P^M(t, tPS?) = (\langle tPS?\rangle\, 1)^M(s).$$

Now, for any atom S, since

$$(\imath_{R0?}\langle t\rangle, S) = (\imath_{R0?}, \langle t\rangle S),$$
$$(\imath_s\langle t\rangle, S) = (\imath_s, \langle t\rangle S),$$

and

$$\begin{aligned}\langle t\rangle S &= \langle t\rangle 1 \qquad \text{if} \quad R(t) = S,\\ &= 0 \qquad \text{otherwise,}\end{aligned}$$

we have that

$$(\imath_{R0?}\langle t\rangle, S) = (\imath_s\langle t\rangle, S). \tag{1}$$

It follows by induction on the structure of any f over PP and FF that

$$(\imath_{R0?}\langle t\rangle, f) = (\imath_s\langle t\rangle, f).$$

The basis is given by (1). If f is of the form $ag + bh$, the result follows from the linearity of the integral (,). If f is of the form Bg then

$$\begin{aligned}(\imath_{R0?}\langle t\rangle, Bg) &= (\imath_{R0?}\langle t; B?\rangle, g)\\ &= (\imath_{R0?}\langle t\rangle, g) \qquad \text{if} \quad R(t) \leqslant B,\\ &= 0 \qquad \text{otherwise,}\end{aligned}$$

and similarly for $(\imath_s\langle t\rangle, Bg)$. If f is of the form $\langle p\rangle g$, then there are five cases, depending on the form of p. For $P \in PP$,

$$\begin{aligned}(\imath_s\langle t\rangle, \langle P\rangle g) &= (\imath_s\langle tP\rangle, g)\\ &= \sum_S (\imath_s\langle tPS?\rangle, g),\end{aligned}$$

where the sum is over all atoms S, and similarly for $(\imath_{R0?}\langle t\rangle, \langle P\rangle g)$. The result follows from the induction hypothesis. For $p = aq + br$, we rewrite $\langle p\rangle g$ as $a\langle q\rangle g + b\langle r\rangle g$ and use the linearity of (,). For $p = qr$, we rewrite $\langle qr\rangle g$ as $\langle q\rangle\langle r\rangle g$ and use the induction hypothesis with respect to q. For $p = B?$ we rewrite $\langle B?\rangle g = Bg$; this case was handled above. Finally, for $p = q^*$, we rewrite $\langle q^*\rangle g = \sum_{n=0}^{\infty}\langle q^n\rangle g$ and use the linearity and continuity of the integral to get

$$\begin{aligned}(\imath_{R0?}\langle t\rangle, \langle q^*\rangle g) &= \sum_{n=0}^{\infty} (\imath_{R0?}\langle t\rangle, \langle q^n\rangle g)\\ &= \sum_{n=0}^{\infty} (\imath_s\langle t\rangle, \langle q^n\rangle g)\\ &= (\imath_s\langle t\rangle, \langle q^*\rangle g).\end{aligned}$$

In particular,

$$f^N(R_0?) = (\iota_{R_0?}\langle R_0?\rangle, f) = (\iota_s\langle R_0?\rangle, f) = f^M(s). \quad \blacksquare$$

By Lemma 1, to check of $f-g>0$ is satisfiable, we need only look among instantiations of T.

A term is in *normal form* if it is a sum of terms of the form $\langle p_1\rangle\langle p_2\rangle\cdots\langle p_n\rangle(a1)$. Any term over well-structured programs can be put into normal form with at most quadratic increase in size. We will assume for simplicity that the input is of this form, although this is not essential to the algorithm. The *Fischer–Ladner closure* FL of a term is defined as in PDL [FL].

We now describe a procedure for labeling nodes s of T with $\Gamma(s)$, $\Delta(s)$, and $c(s)$, where $\Gamma(s)$ and $\Delta(s)$ are sums of at most n elements of FL (repetitions allowed), $n=|f-g|$, and $c(s)$ is a rational number. The labeling proceeds inductively down the tree. Each root r is labeled $\Gamma(r)=f-g$. Now suppose node s has been labeled $\Gamma(s)$ in normal form. Reduce $\Gamma(s)$ via the rules

(1) $\langle p; q\rangle f \to \langle p\rangle\langle q\rangle f$

(2) $\langle \textbf{if } B \textbf{ then } p \textbf{ else } q\rangle f \to \langle p\rangle f \quad$ *if* $R(s)\leqslant B$,
$\quad\quad \to \langle q\rangle f \quad$ if $R(s)\leqslant \neg B$

(3) $\langle \textbf{while } B \textbf{ do } p\rangle f \to f \quad$ if $R(s)\leqslant \neg B$,
$\quad\quad \to \langle p\rangle\langle \textbf{while } B \textbf{ do } p\rangle f \quad$ if $R(s)\leqslant B$

(4) $\langle B?\rangle f \to f \quad$ if $R(s)\leqslant B$,
$\quad\quad \to 0 \quad$ if $R(s)\leqslant \neg B$.

Whenever (3) is applied with $R(s)\leqslant B$, mark the occurrence of the while loop to indicate that it has been expanded. If ever (3) is about to be applied to a marked term, then the term can be set to 0. This is because under $R(s)$, if

$$\langle p\rangle\langle \textbf{while } B \textbf{ do } p\rangle f \to \cdots \to \langle \textbf{while } B \textbf{ do } p\rangle f,$$

then the while loop will execute forever without changing the value of $R(s)$. This can be proved formally using the axioms of Section 4.

Thus $\Gamma(s)$ can be reduced to a sum of terms of the form $\langle P\rangle f$ and $a1$ in FL, where P is a primitive program and a is rational. Denote this reduced form of $\Gamma(s)$ by $\Delta(s)$. Note that $\Gamma(s)$ and $\Delta(s)$ are equivalent under $R(s)$, and that the number of terms of $\Delta(s)$ is at most the number in $\Gamma(s)$, since no rule increases the number of terms in the sum.

For each primitive program P, collect all terms of $\Delta(s)$ of the form $\langle P\rangle f$ to get $\langle P\rangle f_1 + \cdots + \langle P\rangle f_k$, and rewrite this as $\langle P\rangle(f_1+\cdots+f_k)$. For each P-successor t of s, label $\Gamma(t)=f_1+\cdots+f_k$, and erase all marks from rule (3). Collect all constant terms $a1$ in $\Delta(s)$ and add them to get $c1$, and let $c(s)=c$. If no constant terms exist, let $c(s)=0$.

This process is continued in a regular fashion all the way down the tree. Each $\Delta(s)$ is a sum of at most n elements of FL, and s, t with $\Delta(s)=\Delta(t)$ have isomorphic subtrees below them, therefore there are $2^{O(n)}$ isomorphism classes of labeled subtrees of T.

By a slight abuse of notation we may write $\Delta^M(s)$ for the value of $\Delta(s)^M$ at s in some instantiation M of T. $\Delta^M(s)$ depends on the values of $\Delta^M(t)$ for immediate sucessors t of s, as follows. Let $M(s, t)$ denote the probability assigned to the edge (s, t), i.e., $M(s, t)=P^M(s, t)$, where P is the unique primitive program such that t is a P-successor of s.

LEMMA 2. $\Delta^M(s)=c(s)+\sum_{(s,t)} M(s, t)\, \Delta^M(t)$.

Define

$$\sup \Delta(s)=\sup\{\Delta^M(s) \mid M \text{ an instantiation of } T\}.$$

By Lemma 1,

$$\begin{aligned}&\sup\{(f-g)^M(s) \mid M \text{ a model}, s\in M\}\\ &\qquad=\max\{\sup \Delta^M(r) \mid M \text{ an instantiation of } T, r \text{ a root of } T\}.\end{aligned}$$

We have not claimed that this supremum is actually achieved, but this follows from the next lemma. Define an instantiation of T to be *regular* if the subtrees below any pair of states s, t with $\Delta(s)=\Delta(t)$ are isomorphic with respect to edge probabilities. Define a model to be *deterministic* if all edge probabilities are either 1 or 0.

LEMMA 3. *For all s in T, $\sup \Delta(s)$ is attained in a regular, deterministic instantiation of T.*

Proof. The proof is by induction on $n(\Delta(s))$, the number of terms in the sum $\Delta(s)$. If $n(\Delta(s))=0$, or if $\Delta(s)$ consists only of constant terms, then $\Delta(s)=c(s)$; in this case any arbitrary regular, deterministic assignment of probabilities below s suffices, since it cannot affect the value of $\Delta(s)$.

Assume that $n(\Delta(s))\geqslant 1$ and the lemma holds for $n(\Delta(t))<n(\Delta(s))$. If $\Delta(s)$ contains terms $\langle P\rangle f$ and $\langle Q\rangle g$ with $P\neq Q$, or if $\Delta(s)$ contains a constant term $a1$, then $n(\Delta(t))<n(\Delta(s))$ for every descendant t of s. Replace the subtree below t with a regular deterministic model maximizing $\Delta(t)$, by adjusting the probabilities on the edges below t. Do this in a uniform fashion so that the model is regular for nodes t with $n(\Delta(t))<n(\Delta(s))$. For any assignment M to the edges out of s,

$$\Delta^M(s)=c(s)+\sum_{(s,t)} M(s, t) \sup \Delta(t),$$

by Lemma 2. For each P such that some $\langle P\rangle f$ appears in $\Delta(s)$, pick a P-successor t of s such that $\sup \Delta(t)$ is maximal over all P-successors, and assign 1 to (s, t) and 0 to all other P-successors. If no $\langle P\rangle f$ occurs in $\Delta(s)$, assign probabilities to P-successors arbitrarily (but deterministically), as above. Again by Lemma 2, with this probability assignment M, $\Delta^M(s) = \sup \Delta(s)$, and the model is of the desired form.

The only remaining case is $\Delta(s)$ of the form $\langle P \rangle f_1 + \cdots + \langle P \rangle f_k$ for some P. Let $\Delta_1, ..., \Delta_m$ be the set of all labels with $n(\Delta_i) < n(\Delta(s))$. For each Δ_i, choose a regular deterministic model maximizing Δ_i, and replace the subtree below any t with $\Delta(t) = \Delta_i$ with that model. Let the edges in those subtrees be fixed, and consider a probability assignment M to the other edges of T. It follows from Lemma 2 that

$$\Delta^M(s) = \sum_{t \in U} M(s \to t) \sup \Delta(t),$$

where U is the set of states t such that $n(\Delta(t)) < n(\Delta(s))$ and there is a path $s \to t$ through only states u with $n(\Delta(u)) = n(\Delta(s))$, and $M(s \to t)$ is the product of the probabilities on edges along that path. This equation holds because there are no constant terms in $\Delta(s)$ or any $\Delta(u)$ along the path to contribute to $\Delta^M(s)$. Then

$$\begin{aligned}\Delta^M(s) &= \sum_{i=1}^{m} \sum_{\substack{t \in U \\ \Delta(t) = \Delta_i}} M(s \to t) \sup \Delta_i \\ &= \sum_{i=1}^{m} e_i \sup \Delta_i,\end{aligned}$$

where

$$e_i = \sum_{\substack{t \in U \\ \Delta(t) = \Delta_i}} M(s \to t).$$

But $\sum_i e_i \leqslant 1$, because for each node along the path $s \to t$ there is a unique P such that all successors of that node are P-successors. Therefore

$$\sum_{i=1}^{m} e_i \sup \Delta_i \leqslant \sup \Delta_0,$$

where $\sup \Delta_0 = \max_i \sup \Delta_i$. Thus $\sup \Delta(s)$ is attained at s by picking a descendant $t \in U$ with $\Delta(t) = \Delta_0$ and setting all edge probabilities on the path $s \to t$ to 1, and all other unfixed edges below s to 0. Moreover, such a path can be chosen with no repeated labels, by the regularity of the labeling. If no such t exists, an arbitrary regular, deterministic labeling below s suffices. This gives an instantiation below s of the desired form. ∎

Since the model is regular, it can be collapsed into an equivalent finite model with $2^{O(n)}$ states. This gives

THEOREM 1 (Finite model property). *A formula $f \leqslant g$ involving only well-struc-*

tured programs is true in all models iff it it true in all deterministic regular models with at most $2^{O(|f-g|)}$ states.

THEOREM 2. *There is a polynomial-space algorithm to decide the validity of formulas $f \leqslant g$ involving only well-structured programs.*

Proof. The procedure builds a deterministic, regular instantiation M of T nondeterministically. It first guesses a root r and labels it $\Gamma(r)=f-g$. At each node s, it constructs $\Delta(s)$ from $\Gamma(s)$, remembers $c(s)$, and for each P such that $\langle P\rangle f$ appears in $\Delta(s)$, guesses a P-successor, then repeats the process on all guessed successors. It also keeps a depth count, halting if the depth goes below a point such that some $\Delta(s)$ must have been repeated. If there is a repetition, say $\Delta(s_1)=\Delta(s_2)$ and s_1 and s_2 lie on the same path, then $c(t)=0$ at every node t on the path between s_1 and s_2. There are never more than $n(\Gamma(r))$ nodes being considered at any time, since the labeling rules never increase $n(\Gamma(s))$, nor are there ever more than $n(\Gamma(r))$ nonzero $c(s)$ that have to be remembered. After the depth count runs out, the sum of the numbers $c(s)$ is computed, and the machine accepts iff this number is positive. By Lemma 3, $f-g>0$ is satisfiable iff it is satisfied in a model obtained in this way. The algorithm can be made deterministic by Savitch's theorem. ▌

7. PROBABILISTIC PROGRAM ANALYSIS IN PPDL

In this section we illustrate how to use the formalism in probabilistic program analysis. The example is quite modest, but serves to illustrate the importance of linearity and the use of the device $\langle c := c-1^*; c \geqslant 1?\rangle\, 1$ in calculating expected running times. Of course, in real program verification, PPDL must be fleshed out with a language for the domain of computation, and properties of the domain of computation must be used in the proof. However, much of the domain-independent portion of the reasoning can be done in pure PPDL.

Consider the following simple random walk: starting at a distance $n>0$ steps from the goal, we flip a fair coin every minute, and depending on the outcome, we either take one step closer or stay where we are. What is the expected time to reach the goal? Expressed as a while program, the random walk is

```
$x := n$;
$c := 0$;    /* initialize step counter */
while $x \neq 0$ do
  begin
    $\frac{1}{2}$(skip $+ x := x-1$);
    $c := c+1$
  end
```

and we are interested in the expected final value of c. (Note the two different uses of the symbol $+$: the first is to be interpreted in the linear space of Markov transitions and the second in the natural numbers.) In terms of the primitive PPDL operators, this program is

$$p \;=\; x := n;\, c := 0;\, (x \neq 0?;\, \tfrac{1}{2}(\mathbf{skip} + x := x-1);\, c := c+1)^*;\, x = 0?$$

and to get the expected final value of c, we wish to calculate the value of the constant function

$$\langle p;\, c := c-1^*;\, c \geqslant 1? \rangle\, 1. \tag{2}$$

Let

$$q = (x \neq 0?;\, \tfrac{1}{2}(\mathbf{skip} + x := x-1);\, c := c+1)^*$$

and observe by rule (9) and linearity, for any g,

$$\begin{aligned}
&\langle x := n;\, q;\, x = 0? \rangle\, g \\
&\quad = \langle x := n;\, q;\, x \neq 0?;\, \tfrac{1}{2}(\mathbf{skip} + x := x-1);\, c := c+1;\, x = 0? \rangle\, g \\
&\quad\quad + \langle x := n;\, \mathbf{skip};\, x = 0? \rangle\, g
\end{aligned} \tag{3}$$

and

$$\langle x := n;\, \mathbf{skip};\, x = 0? \rangle\, g = 0,$$

using the assignment axiom $\langle x := n \rangle (x = n)$ of Dynamic Logic and the assumption $n \neq 0$. By linearity and simple facts about the integers,

$$\begin{aligned}
&\langle x \neq 0?;\, \tfrac{1}{2}(\mathbf{skip} + x := x-1);\, c := c+1;\, x = 0? \rangle\, g \\
&\quad = \tfrac{1}{2}\langle x \neq 0?;\, \mathbf{skip};\, c := c+1;\, x = 0? \rangle\, g \\
&\quad\quad + \tfrac{1}{2}\langle x \neq 0?;\, x := x-1;\, c := c+1;\, x = 0? \rangle\, g \\
&\quad = 0 + \tfrac{1}{2}\langle x = 1?;\, c := c+1 \rangle\, g,
\end{aligned}$$

therefore (3) is equal to

$$\tfrac{1}{2}\langle x := n;\, q;\, x = 1?;\, c := c+1 \rangle\, g.$$

Using the * rules and facts about the integers, one can prove that for any h,

$$\begin{aligned}
&\langle x := n;\, q;\, x = 1? \rangle\, h \\
&\quad = \langle x := n;\, (x \neq 0?;\, \tfrac{1}{2}(\mathbf{skip} + x := x-1);\, c := c+1)^*;\, x = 1? \rangle\, h \\
&\quad = \langle x := n;\, (\tfrac{1}{2}(\mathbf{skip} + x := x-1);\, c := c+1)^*;\, x = 1? \rangle\, h
\end{aligned}$$

which says that our original formula (1) is equivalent to

$$\tfrac{1}{2}\langle c := 0; x := n; (\tfrac{1}{2}(\textbf{skip} + x := x - 1); c := c + 1)^*;$$
$$x = 1?; c := c + 1; c := c - 1^*; c \geqslant 1?\rangle\, 1.$$

Using the definition $p^* = \sum_{n=0}^{\infty} p^n$, and linearity and continuity to take the summation out front, this is equivalent to

$$\tfrac{1}{2}\sum_i \sum_j \langle c := 0; x := n; (\tfrac{1}{2}(\textbf{skip} + x := x - 1); c := c + 1)^i;$$
$$x = 1?; c := c + 1; c := c - 1^j; c \geqslant 1?\rangle\, 1$$

and by commutativity of statements involving disjoint sets of variables, this becomes

$$\tfrac{1}{2}\sum_i \sum_j \langle x := n; (\tfrac{1}{2}(\textbf{skip} + x := x - 1))^i; x = 1?; c := 0;$$
$$c := c + 1^i; c := c + 1; c := c - 1^j; c \geqslant 1?\rangle\, 1$$
$$= \sum_i \sum_j 2^{-(i+1)} \langle x := n; (\textbf{skip} + x := x - 1)^i; x = 1?\rangle$$
$$\langle \mathrm{c} := i + 1 - j; c \geqslant 1?\rangle\, 1$$
$$= \sum_i \sum_j \sum_{k=0}^{\infty} 2^{-(i+1)} \binom{i}{k} \langle n = k + 1?; j \leqslant i?\rangle\, 1$$
$$= \sum_i \sum_j 2^{-(i+1)} \binom{i}{n-1} \langle j \leqslant i?\rangle\, 1$$
$$= \sum_i (i+1)\, 2^{-(i+1)} \binom{i}{n-1}\, 1$$

using the facts

$$\langle c := i + 1 - j; c \geqslant 1?\rangle\, 1 = \langle j \leqslant i?\rangle\, 1,$$
$$(\textbf{skip} + x := x - 1)^i = \sum_{k=0}^{\infty} \binom{i}{k} (x := x - 1)^k$$
$$\langle x := n; x := x - 1^k; x = 1?\rangle\, 1 = \langle n = k + 1?\rangle\, 1.$$

The final value of the expression is

$$\sum_{i=0}^{\infty} i 2^{-i} \binom{i-1}{n-1}\, 1,$$

which reduces to $2n$ by elementary combinatorics [L, p. 32]. In general, one would first translate a while program into PPDL, then apply elementary valid program transformations until all programs are removed. The resulting expression can be evaluated using elementary combinatorics as in the example above, or estimated using Stirling's formula.

Acknowledgments

This work has benefited greatly from the ideas of many people. In particular I would like to thank Ernst Doberkat, Joe Halpern, David Harel, Rohit Parikh, and John Reif. Special thanks go to Gordon Plotkin for the many stimulating discussions and ideas which ultimately led to the insights of Section 2.

References

[F] Y. A. Feldman, A decidable propositional probabilistic dynamic logic, *in* "Proc. 15th ACM Sympos. on Theory of Computing," 1983, pp. 298–309.

[FH] Y. A. Feldman and D. Harel, A probabilistic dynamic logic, *in* "Proc. 14th ACM Sympos. on Theory of Computing," 1982, p. 181.

[FL] M. J. Fischer and R. E. Ladner, Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18** No. 2 (1979).

[H] P. R. Halmos, "Measure Theory," Van Nostrand, Princeton, N.J., 1950.

[HR] J. Halpern and J. Reif, Propositional dynamic logic of deterministic, well-structured programs, *in* "Proc. 22nd IEEE Sympos. on Foundations of Computer Science," 1981, p. 322.

[K] D. Kozen, Semantics of probabilistic programs, *J. Comput. System. Sci.* **22** (1981), 328.

[LS] D. Lehmann and S. Shelah, Reasoning with time and chance, manuscript, Oct. 1982.

[L] C. L. Liu, "Introduction to Combinatorial Mathematics," McGraw–Hill, New York, 1968.

[MT] J. A. Makowsky and M. L. Tiomkin, Probabilistic propositional dynamic logic, manuscript.

[PM] R. Parikh and A. Mahoney, A theory of probabilistic programs, manuscript.

[Pn] A. Pnueli, On the extremely fair treatment of probabilistic algrithms, manuscript, Nov. 1982.

[Ra] L. Ramshaw, "Formalizing the Analysis of Algorithms," Ph.D. thesis, Stanford University, 1980.

[Re] J. Reif, Logics for probabilistic programming, *in* "Proc. 12th ACM Sympos. on Theory of Computing," 1980.

[Rev] D. Revuz, "Markov Chains," North-Holland, Amsterdam, 1975.

[S] H. H. Schaefer, "Topological Vector Spaces," Graduate Texts in Mathematics Vol. 3, Springer-Verlag, New York/Berlin, 1971.

[SPH] M. Sharir, A. Pnueli, and S. Hart, Probabilistic verification of programs. I. Correctness analysis of sequential programs, manuscript, Dec. 1981.