DECIDABILITY QUESTIONS FOR FINITE PROBABILISTIC PROPOSITIONAL DYNAMIC LOGIC

by

M.L. Tiomkin and J.A. Makowsky

Technical Report #394
January 1986
Decidability Questions for Finite
Probabilistic Propositional Dynamic Logic

(Revised version, July 1985)

M. L. Tionkin

IBM Scientific Center, Haifa, Israel

J. A. Makowsky

Department of Computer Science, Technion, Haifa, Israel

Abstract: This paper deals with various versions of finite propositional-probabilistic dynamic logic. Besides the logics previously introduced in the literature [Fe, Ko79, Ko83] we present some natural variations and extensions of these logics. We also present another kind of probabilistic propositional dynamic logic with simple probabilistic estimations and "almost regular" program language. We investigate the (un)decidability of these logics and present a complete picture of decidability and undecidability. Some of these logics have the finite model property, and therefore, if they are undecidable, they are exactly \( \Pi_2 \). We show that allowing nesting and probabilistic choice leads to undecidability.
1. Introduction.

This paper deals with various versions of finite propositional probabilistic dynamic logic (PPDL). The basic dynamic constructs here are either stating that a program \( \alpha \) terminates in a state satisfying \( p \) with probability \( \lambda \), or a program \( \alpha \) terminates in a state satisfying \( p \) with probability greater than some constant \( \lambda \).

The exact way we want to speak about the probabilities \( \lambda \) may vary. We study three basic versions of PPDL:

1. Deterministic PPDL (DPPDL) introduced in [Fe], where only deterministic regular expressions \( \alpha \) appear as program constructs and boolean combinations of propositional variables \( p \) are allowed. However, to speak about the probabilities we can use first order formulas of real closed fields.

2. Rational PPDL (PPDL\( >r \)), where we can say about the probabilities only that they are greater than a rational constant \( r \).

3. Positive PPDL (PPDL\( >0 \)) is like rational PPDL, where the only probability we talk about is 0.

Clearly, (3) is a special case of (2) and deterministic (2), with "poor test", is a special case of (1). In [Fe] it is proved that DPPDL is decidable, and therefore also PPDL\( >r \) and PPDL\( >0 \). The question we want to ask in this paper, is, whether DPPDL remains decidable, when we allow nesting ("rich test") or add additional constructs of global assignment and probabilistic choice.

In the case of DPPDL the addition of global assignment does not make the resulting logic undecidable [Fe], but our result here is that probabilistic choice or nesting do make it undecidable. In the case of PPDL\( >r \) we get exactly the same decidability /undecidability results as in DPPDL, whereas in the case of PPDL\( >0 \) all the extensions remain decidable (because of its equivalence to the usual PDL).
In spirit our investigation is similar to the investigation of decidability questions in [HS]. There it was shown that adding a simple context free program makes propositional dynamic logic undecidable. Here, adding either nesting or probabilistic choice makes already PPDL\textgreater r undecidable, though it was obtained from DPPDL by removing the quantifers and arithmetic operations from the logic. This shows that, as in the case of PDL, the decidability of rather simple probabilistic dynamic logics is very sensitive to the choice of permissable constructs. The undecidability results in this paper are nontrivial and were obtained in [T], cf. also [TM], whereas the decidability results are either old or trivially obtained from the literature, especially [Fe,PM,Ko83].

The paper is structured as follows. In section 2 we give the basic formal definitions and state our results formally. In section 3 we show that PPDL\textgreater 0 is reducible to PDL. In section 4 we collect some model theoretic results for PPDL\textgreater r. In section 5 we present our undecidability results for the extensions of DPPDL. This is done by a straight coding of arithmetic. In section 6 we present the undecidability results for PPDL\textgreater r. Here we use an undecidability result for probabilistic automata, due to A.Paz [Paz]. In an appendix we show the undecidability of the logic introduced in [Ko83], when we allow more than one inequality.

2. Definitions and main results.

2.1 Language definition.

A signature \( \sigma \) is a triple \( \sigma = \{ p_i \}_{i \in \omega}, \{ \alpha_j \}_{j \in \omega}, \{ q_k \}_{k \in \omega} \), where \( \{ p_i \}_{i \in \omega} \) is a set of (local) propositional variables, \( \{ \alpha_j \}_{j \in \omega} \) is a set of program variables and \( \{ q_k \}_{k \in \omega} \) is a set of global propositional variables. For a countable signature \( \sigma \) and a tuple \( xyz ( x,y,z = 0,1) \) we define a language PPDL_{x,y,z}. We shall define simultaneously the sets of formulas \( (Fm_{x,y,z}) \) and program terms \( (Pr_{x,y,z}) \) of PPDL_{x,y,z} \((x=1\) means that the language allows global assignments, \( y=1\) - probabilistic choice...
and \( z = 1 \) - nesting of probabilistic estimations). We shall sometimes omit \( \sigma \).

a) if \( p_i \) is a propositional variable, then \( p_i \in \text{Pr}_{\text{sys}} \); if \( \alpha_j \) is a program variable, then \( \alpha_j \in \text{Pr}_{\text{sys}} \); if \( q_k \) is a global propositional variable, then \( q_k \in \text{Pr}_{\text{sys}} \) and \( q_k := \text{true}, q_k := \text{false} \in \text{PPDL}_{\text{sys}} \);

b) \( \text{true}, \text{false} \in \text{Pr}_{\text{sys}} \); if \( \phi, \psi \in \text{Pr}_{\text{sys}} \), then \( \phi \land \psi, \phi \lor \psi, \phi \Rightarrow \psi \in \text{Pr}_{\text{sys}} \);

c) if \( \alpha_1, \ldots, \alpha_n \in \text{Pr}_{\text{sys}}, \tau_1, \ldots, \tau_n \) are nonnegative real numbers with \( \sum_{i=1}^{\infty} \tau_i \leq 1 \), then \( (\tau_1 \alpha_1 \cup \ldots \cup \tau_n \alpha_n) \in \text{Pr}_{\text{sys}} \);

d) if \( \phi \in \text{Pr}_{\text{sys}}, \) and \( \alpha, \beta \in \text{Pr}_{\text{sys}} \), then \( (\alpha \land \beta), (\text{if } \phi \text{ then } \alpha) \), \( (\phi?) \in \text{Pr}_{\text{sys}} \) (we shall write \( (\text{if } \phi \text{ then } \alpha) \) instead of \( (\text{if } \phi \text{ then } \text{true}) \));

e) if \( \phi \in \text{Pr}_{\text{sys}}, \lambda \) is a nonnegative real number \( \leq 1 \) and \( \alpha \in \text{Pr}_{\text{sys}} \), then \( \langle \alpha \rangle \lambda \phi \in \text{Pr}_{\text{sys}} \); if \( \phi \) and \( \lambda \) don't contain a "diamond" \( (\langle \rangle) \), then \( \langle \alpha \rangle \lambda \phi \in \text{Pr}_{\text{sys}} \).

2.2 Semantics for PPDL.

Here we consider programs as transition probabilities on the space of 'events' (a state with values of global variables), as it is described in [Ne]. It is assumed that a reader is familiar with measure theory and Lebesgue integral on arbitrary measure (definitions and fundamental properties). Our semantics here is different from [Fe,Ko83 and PM]. They allow only discrete probabilities. The reason why we allow arbitrary probabilities lies in its greater generality. This implies that we shall conceivably have fewer valid formulas, but the discrete and the finite model property will be stronger statements.

Let \( D \) be some nonempty domain and \( S \) a \( \sigma \)-algebra on \( 2^D \). We define a set \( \text{Meas}(S) \) as a set of all \( \sigma \)-additive measures on \( S \) with a measure of \( D \leq 1 \). A function \( f:D \rightarrow \text{Meas}(S) \) is a \textbf{program image} on \( S \), if for each real \( \lambda \geq 0 \) and \( \forall \in S \{ \exists D : f(I)(U) < \lambda \} \in S \).
Composition of program images is defined componentwise by the integral:
\[ f \circ g (I)(U) = \int_D g(J)(U)d\mu_J, \]
and multiplication by nonnegative number \( \lambda \) and the (countable) sum of program images are also defined componentwise provided that the corresponding measures on \( D \) will not be \( >1 \).

Using these properties we can assign program images as meanings of program variables, measurable sets as meanings of propositional variables, and give an inductive definition of meaning of a program term and/or a formula of PPDL in a model \( M = \langle D, S, \{ P_i \}, \{ f_{ai} \} \rangle \), where \( D \) is a nonempty domain, \( S \subset 2^D \) is a \( \sigma \)-algebra on \( D \), \( P_i \in S \) are the meanings of propositional variables and \( f_{ai} \) are program images on \( S \) which serve as the meanings of program variables. After that we shall say \( I \models_M \varphi \) if \( I \) belongs to the meaning of the \( \varphi \), \( M \models \varphi \) if the meaning of \( \varphi \) is \( D \) and \( \models_{pr} \varphi \) if for every \( M \models \varphi \).

### 2.3 Primitive formulas.

In this subsection we define a normal form used to facilitate structural induction in the proof of proposition 4.4.

**Definition 2.3.1.** A formula \( \varphi \) of PPDL is **finitary primitive** (or \( FPF \)) if it is built from program and propositional variables using only finitary logical connectives and probabilistic diamonds with rational probabilities. We call a set of these formulas \( FPF \).

**Theorem 2.3.2** Every formula \( \varphi \) of PPDL is equivalent (semantically) to some (may be infinitary) boolean combination \( \psi \) of finitary primitive formulas. The translation is recursive.

This is proved by induction and a proof uses a set of valid (not always finite) PPDL formulas which serve us also as the examples for valid formulas (the
PPDL

numbers $r$ and $s$ run over rational numbers): 

(i) $<a>\lambda \psi = \bigvee_{\lambda < 1} <a> \psi$,  

(ii) $<\psi \triangleright \lambda \varphi = \psi \& \varphi$, for $\lambda < 1$,  

(iii) $<\text{if } \psi \text{ a else } \beta > \varphi = (\psi \triangleright <a> \varphi) \& (\neg \psi \triangleright \beta > \varphi)$,  

(iv) $<a; \beta > \varphi = \bigvee_{0<r_1<\ldots r_n<1} \bigwedge_{i=1}^n <a> r_i (<\beta > s_i \varphi \& \neg <\beta > s_i+1 \varphi), r_1 s_1 + \ldots + r_n s_n > r$  

(v) $<\text{while } \psi > r \varphi = \bigvee_{n r < r_1 + \ldots + r_n < 1} \bigvee_{i=1}^n <(\psi ? : a)_i \triangleright \neg \psi ? > r_i \varphi$.  

(vi) $<a> r \psi_n = \bigvee_n <a> r \bigwedge_{i=1}^n \psi_i$.  

(vii) $<a> r \& \psi_n = \bigvee_{s > r} \bigwedge_{i=1}^n s \& \psi_i$.  

A proof of (iv) uses properties of integral, and a proof of (vi) and (vii) uses the $\sigma$-additivity.

2.4 The restricted probability estimations.

We shall define as PPDL$>0$ a language where the only allowed probabilistic estimation (the "diamond") is $>0$ and the logic is finite. We shall see later that this restriction affects the decidability of PPDL. We shall call PPDL$>r$ a finite language where the only numbers which appear in the estimations and probabilistic choice are rational numbers. In such a language the formulas are constructive objects, and we can talk about decidability of its validity problem.
2.5 The (un)decidability of different versions of PPDL.

We shall present here a table which contains the (un)decidability results about the PPDLs defined in this section. The proofs will follow. The + means decidability of validity problem. The - means the undecidability of validity problem and inexistence of a recursive proof system (the set of valid formulas is not recursively enumerable).

<table>
<thead>
<tr>
<th></th>
<th>-NE(^{-})</th>
<th>NE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-SW</td>
<td>SW</td>
</tr>
<tr>
<td>-CH</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>CH</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>-CH</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>GH</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Here "NE" means 'with nesting of probability estimations' \((z=1)\), "SW" means 'with global assignments' \((x=1)\) and "CH" means 'with probabilistic choice' \((y=1)\). DPPDL is the logic proposed in the work of Y. Feldman. In this logic only deterministic regular programs are allowed, and probabilistic estimations are allowed only on events which are boolean combinations of propositional variables. In spite of this, the estimations may be defined by a formula with arithmetic operations and quantifiers over real numbers. We can easily extend this logic by allowing nesting and probabilistic choice, if we deal with only discrete models. In the case of "NE" all combinations of "SW" and "CH" give the same decidability results. DPPDL deals with discrete probabilities only.
PDL formulas and programs are built from the propositional and program variables by allowing finite boolean operations for formulas, composition, nondeterministic choice and star (nondeterministic iteration) for programs and \(<\>("diamond") without probability (see [FL]).

**Proposition 3.1.** There is a translation \( tr \) from PDL to PPDL\(>0 \) such that \( tr \) is the identity on the propositional and program variables, commutes with the boolean operations and composition of programs(\(.)\).

**Proof:** We translate \((\alpha \lor \beta)\) by \((r_1 \alpha^{tr} \lor r_2 \beta^{tr})\) for some randomly chosen \( r_1, r_2 > 0, r_1 + r_2 \leq 1 \).

\(<\alpha>\phi\) is translated by \(<\alpha^{tr}>0\phi^{tr}\).

Finally, we translate \( \alpha^* \) by \((q_i := \text{true}; \text{while } q_i \ (r_1 \alpha^{tr} \lor r_2 q_i := \text{false})\), where \( q_i \) is some new global variable.

To verify that \( tr \) preserves variables and commutes with the boolean and regular connectives is now verified easily by the reader. Note that \( tr \) does not depend on \( r_1 \) and \( r_2 \).

**Proposition 3.2.** The translation \( tr \) preserves validity, i.e. for every formula \( \phi \) of PDL \( \phi \) is a valid PDL formula iff \( \phi^{tr} \) is a valid PPDL\(>0 \) formula.

**Proof:** The proof uses the finite model property of PDL and its axiomatization (see [Ga], [FL]). We have two cases to consider:

1. \( \not \vDash_{PDL} \phi \). Then there is finite model \( M \models \neg \phi \). From such \( M \) we construct a natural probabilistic model \( \hat{M} \) with the same domain and predicate values, and for the program variables we define

\[
\# a_j(J)(U) = \frac{\text{the number of } J, \text{ such that } R_a(I)(J) \text{ and } J \in U}{|D|}
\]

(here a \( \sigma \)-algebra is the algebra of all subsets of the domain \( D \)). Then
we prove that $M \models \varphi$ iff $M \models \varphi^r$ by induction on $\varphi$, and then we have $M \not\models \varphi^r$, and $\not\models_{Pr} \varphi^r$.

2. $\models_{PPDL} \varphi$. Then it is provable in some axiom system (see [Ga]), and one can prove by induction on this proof sequence that $\models_{Pr} \varphi^r$.

Note that this result is in contrast to [Ko79] where it was stated that on the first order level nondeterministic and probabilistic semantics are incompatible.

**Corollary 3.3.** The logic PPDL$> _ 0$ is decidable.

*Proof:* This follows from the decidability of PDL [Fl.] and the fact that every formula $\psi$ from PPDL$> _ 0$ is of the form $\varphi^r$ where $\varphi$ is obtained from $\psi$ simply by dropping the probability values $r_1$ and $r_2$.

4. **Discrete and finite models of PPDL.**

Note that if we want some computability we have to deal with finite models and finite formulas of PPDL. Here we shall do it. We assume that $\sigma$ is some finite signature for PPDL.

**Definition 4.1.** We say that a model $M$ of PPDL is a finite model if its domain is finite, and $M$ is a finite rational model if it is finite and all the values $f_\alpha(I)(U)$ for $I \in D$, $U \in S$, $\alpha$ from $\sigma$ are rational numbers.

We shall see that it is not so complicated to check satisfiability of a finite formula in a finite rational model. A stronger results was independently proved in [PM].

**Proposition 4.2.** There exists a polynomial algorithm which decides $I \models M \varphi$ for a finite rational model $M$, a state $I$ of $M$ and a formula $\varphi$ from PPDL$> _ r$.

*Proof.* We construct an iterative algorithm which builds the sets $U_\varphi$ and the
functions $f_\alpha$ for all the subformulas $\psi$ and program terms $\alpha$ from $\varphi$. The only interesting case is a program term $\alpha = (\text{while } \psi \beta)$, where $U_\varphi$ and $f_\psi$ are already known. It is easy to see that for $I \in U_\varphi$ $f_\alpha(I)(\{I\}) = 1$ and $f_\alpha(I)(\{J\}) = 0$ for $J \neq I$.

For $I \in U_\varphi$ we use the well known facts from the theory of finite Markov chains (see [KS]). We construct a Markov chain from the transition probability $f_\psi$. First of all we make all the states from $U_\varphi$ absorbing. At this step we should know that $f_\alpha(I)(\cdot) = 0$ for every ergodic states from $U_\varphi$. We delete all these states. We add one new absorbing state in order to make the living probability 1.

After that we can already compute the probabilities to go from a transient state $I$ from $U_\varphi$ to an absorbing state $J - \{b_\mu\} = NR$, where the matrix $\begin{bmatrix} I & 0 \\ R & Q \end{bmatrix}$ is the transition matrix of the chain ($I$ is the unit matrix) and $N = (I - Q)^{-1}$ is the fundamental matrix (see [KS]). All these operations may be done in polynomial time.

Now we shall show that all the PPDL$_{\forall \exists \forall \forall}$, i.e. without nesting have the finite model property. In other words their validity problem is not as complicated as one might think. The proof will be long, but the idea is trivial. We use the fact that a regular program is a limit of finite programs. The same idea was also used in [Fe,PM] to prove their similar results.

Here we should deal with some countable signature $\sigma'$ consisting of program variables $\alpha_1, \ldots, \alpha_n$. $N^{<\omega}$ is a set of finite sequences of natural numbers.

**Definition 4.3.** We say that a model of PPDL is a quasi-tree model if its domain $D = \bigcup_{\mu \in N^{<\omega}} D_\mu, \{D_\mu\}$ are disjoint, $D_\Lambda$ is not empty ($\Lambda$ is the empty sequence) and for $I \in D_\mu, t \in N$ we have.
We say that a set $D_A$ is a root of the quasi-tree model.

**Proposition 4.4.** For a model $M$ and a state $I$ from $M$ there exists a quasi-tree model $M'$ and a state $I'$ from the root of $M'$ such that for every formula $\varphi$ from PPDL$_\sigma$,

$$I \models \varphi \iff I' \models \varphi.$$  

**Proof.** The proof is similar to the proof of existence of tree model for PDL. We construct a model $M'$ as follows:

- $D' = D \times N^{<\omega}$,
- $D_\mu = D \times \{\mu\}$,
- $S'$ is a $\sigma$-algebra generated by $S$ on $D$ and the algebra of all subsets on $N^{<\omega}$.
- $P'_\lambda = P \times N^{<\omega}$,
- $f_{\alpha}(<J,\mu>)(U) = f_{\alpha}(J)(\pi_D(D_\mu \cap U))$, where $\pi_D$ is a projection from the product on $D$.

We take $I' = <I,A>$ and prove easily that $M'$ is a quasi-tree model and $I'$ belongs to the root. After that we prove by induction (cf. theorem 2.3.2) on $\varphi$ that for every $J \in D, \mu \in N^{<\omega}$

$$J \models \varphi \iff <J,\mu> \models \varphi.$$  

The interesting case of induction is $\varphi = \alpha_J \land \varphi$. Then $U_\varphi = U_\varphi \times N^{<\omega}$ by the inductive assumption, and $f_{\alpha}(<J,\mu>)(U_\varphi) = f_{\alpha}(J)(\pi_D(D_\mu \cap U_\varphi)) = f_{\alpha}(J)(U_\varphi)$.

Then we prove the Proposition by the Théorème 2.3.2.

**Definition 4.5.** For a quasi-tree model $M = \{D,S,\}$ and a natural number $n$ we define a submodel of depth $n$ $M^n = \{D^n,S^n,\}$ as a model generated by sequences from $N^{<\omega}$ of length $\leq n$. Note that this $M^n$ will be also a quasi-tree.
Proposition 4.6. For a quasi-tree model $M$, a state $I \in D_A$, a program term $a$ without "diamonds" (probabilistic estimations), a set $U \in S$ the following holds:

a) $f_a(I)(U \cap D^n) \leq f_a(I)(U \cap D^{n+1})$ for $n \geq 0$;

b) $\lim_{n \to \infty} f_a(I)(U \cap D^n) = f_a(I)(U)$.

The proof is by induction on $\alpha$ and uses the fact that

$$(D^0 \cap U) \cup \bigcup_{n=0}^{\infty} ((D^{n+1}\cap U) \setminus (D^n \cap U)) = U.$$

Proposition 4.7 (finite, depth property). For a quasi-tree model $M$, a formula $\varphi$ from $PPDL_{xy0}$ and a state $I \in D_A$, if $I \models \varphi$, then for some $n \geq 0$ $I \models \varphi$.

Proof. We prove from the Prop. 4.6 that for every positive appearance of $<\alpha > \lambda \psi$ in $\varphi$ if $I \models <\alpha > \lambda \psi$ then for each sufficiently large $n$ $I \models <\alpha > \lambda \psi$, and for every negative appearance of $<\alpha > \lambda \psi$ in $\varphi$ if $I \models -<\alpha > \lambda \psi$ then for each $n$ $I \models -<\alpha > \lambda \psi$.

Definition 4.8. A quasi-tree model $M$ is a tree model if $M^n$ is finite for every $n$.

Proposition 4.9. For a model $M'$ and a state $I$ from $M$ there exists a tree model $M'$ and a state $I'$ from the root of $M'$ such that for every formula $\varphi$ from $PPDL_{xy0}$

$I \models \varphi$ iff $I' \models \varphi$.

Proof. Assume that the finite signature $\sigma$ consists of the program variables $\alpha_1, \ldots, \alpha_m$ and the propositional variables $p_1, \ldots, p_k$. We define a set of formulas $F_\sigma = \{ \nu \nu_1 p_1 : \nu_1 \text{ is } 0 \text{ or nothing} \}$ and a set of program terms $A_\sigma = \{ (true?; \alpha_4; \varphi_1; \ldots; \alpha_q; \varphi_d) : l \geq 0, \alpha_4 \text{ are the program variables, } \varphi_j \in F_\sigma \}$. We construct a model $M'$ as follows:

$D' = A_\sigma$. 
PPDL

\[ S' = 2^{D'} \]

\[(\text{true?}) \models p_i \iff I \models p_i, \]

\[(\ldots \alpha_i; \psi_i?) \models p_i \text{ if } p_i \text{ appears positively in } \psi_i, \]

\[ f_{\alpha_i}(\alpha)((\alpha;\alpha_i;\psi_j)) = \frac{f_{\alpha_i}^H(I)(D)}{f_{\alpha_i}^U(I)(D)} \text{ and } = 0 \text{ in all other cases. We assume here } 0 = 0. \]

This model is a tree model.

After that we take \( I' = \text{true?} \) and prove that for \( \alpha \) from \( A_0 \)

\[ f_{\alpha_i}^U(I)(D) = f_{\alpha_i}^H(I')(D'). \]

Then we can already prove the Proposition using the fact that for a program term \( \beta \) without \( <> \)

\[ f_{\beta}(I)(D) = \sum_{s=1}^{n} \tau_s f_{\alpha_s}(I)(D), \text{ where } \alpha_s \in A_s \]

(each formula \( <\beta;\psi> \lambda \beta \) is equivalent to \( <\beta;\psi> \lambda \text{true} \), so the equivalence for

\[ f_{\beta}(I)(D) \text{ is sufficient}. \]

**Proposition 4.10.** For a formula \( \varphi \) from PPDL\( _{\text{xy}} \) if \( \varphi \) is satisfiable then it is satisfiable in a finite (tree) model.

**Proof.** If \( \varphi \) is satisfiable, then by the Prop. 4.9 it is satisfiable in the root of some tree model \( M \). By the Prop. 4.7 it is satisfiable in the root of \( M^n \) for some \( n \geq 0 \).

It will be a finite model by the Definition 4.8.

Now we have to be able to change irrational probabilities in a finite model saving a satisfiability of a formula.

**Definition 4.11.** For a finite model \( M \), a program variable \( \alpha \) from the signature \( \sigma \), two states \( I, J \) of a model and a number \( \varepsilon > 0 \) we define a finite model \( M(I, J, \alpha, \varepsilon) \) as a model with the same domain as in \( M \), where

\[ f_{\alpha}^{H(I,J,\alpha)}(I)(\{J\}) = f_{\alpha}^{H}(I)(\{J\}) - \varepsilon. \]

All other probabilities and the values of propositional variables are the same.
**Proposition 4.12.** For a finite model $M$, a program variable $\alpha$ and a program term $\beta$ without $<>$ from the signature $\sigma$, four states $I, J, K, L$ of a model with $f_{\alpha}(I)(J) = p > 0$ we have:

a) $f_{\beta}(M[i,j,\alpha\sigma]) (K)(\{L\}) = f_{\beta}(M[i,j,\alpha\sigma]) (K)(\{L\})$ for $0 \leq \epsilon_2 \leq \epsilon_1 \leq p$.

b) $\lim_{\epsilon \to 0} f_{\beta}(M[i,j,\alpha\sigma]) (K)(\{L\}) = f_{\beta}(K)(\{L\})$.

**Proof.** The proof is similar to the proof of Prop. 4.6 and uses the fact that $f_{\beta}(K)(\{L\})$ is a positive (may be infinite) algebraic expression containing $f_{\alpha}(I)(J)$.

Now we can already prove the existence of finite rational model.

**Theorem 4.13.** (Finite model property for PPDL without nesting)

For a formula $\varphi$ from $PPDL_{xy0}$ if $\varphi$ is satisfiable then it is satisfiable in a finite rational (tree) model.

**Proof.** If $\varphi$ is satisfiable, then by the Prop. 4.10 it is satisfiable in some finite (tree) model, and so by the Prop. 4.12 we can change slightly the probabilities in this model making it a finite rational (tree) model and saving the truth of $\varphi$.

## 5. Coding Arithmetics in the logic with quantifiers over probabilities

In this section we establish the undecidability of the natural extension of the decidable Propositional Probabilistic Dynamic Logic appearing in [Fe].

The logic proposed in the work of Y. Feldman is the logic of real numbers with deterministic regular programs and probabilistic estimations on events which are boolean combinations of propositional variables. In this logic only discrete probabilities are allowed and therefore it has the countable model property, as pointed out in [Fe,PM]. If we allow in this logic the more complicated events (we can do it if we allow only discrete probabilities) - we shall call this logic "DPPDL with nesting".
**Theorem 5.1.** DPPDL with nesting is undecidable. In fact it is of the same degree of undecidability as Second Order Arithmetic.

**Proof:** We can define in DPPDL with nesting the natural numbers in the following way:

we define a formula \( \varphi(x) = \{\text{while } \neg(\alpha\{Fr(P)=x\}) \alpha\{Fr\text{true}\} > 0 \} \) (here \( Fr \) is a frequency and \( \beta\psi \) is a truth of \( \psi \) after \( \beta \)). After that the formula

\[
\varphi(1) \land \forall x (x > 1 \implies \neg \varphi(x)) \land \forall x \exists y ((x \times y + y = x \land x > 0) \implies (\varphi(x) = \varphi(y)))
\]

defines the natural numbers, because if this formula is satisfiable there will be \( \varphi(x) \land x > 0 \) if and only if \( \frac{1}{x} \) is a natural number. But the theory of natural numbers is undecidable.

To show that it is not more complicated as Second Order Arithmetic we observe that every formula of DPPDL with nesting can be recursively translated into a rather simple recursive formula of the fpf's, cf. theorem 2.3.2. Because of the countable (discrete) model property of DPPDL such a formula can be coded in Second-Order Arithmetic.

### 6. Undecidability and unexistence of recursive axiomatization.

Here we shall show that the validity problem of \( PPDL>r \) is undecidable. We shall show also that the set of valid formulas of \( PPDL>r \) is not recursively enumerable, and then there is no recursive axiomatization for the \( PPDL>r \) (as in the case of first order predicate calculi of finite models).

**Definition 6.1.** Following [Paz] we define a probabilistic automaton over finite alphabet \( \Sigma \) as \( A = \{\pi, S, A(\sigma)_{\sigma \in \Sigma}, F\} \), where \( S \) is the finite set of states, \( \pi \) is some initial distribution on \( S \), \( A(\sigma) \) is a Markov matrix \( |S| \times |S| \) associated with \( \sigma \in \Sigma \), and \( F \subseteq S \) is a set of terminating states. We define also a P-event as in [Paz] - for a probabilistic automaton \( A \) with rational probabilities and a rational number \( \lambda \) we
define an event $T(A,\lambda) = \{ \omega \in \Sigma : f^A(\omega) > \lambda \}$.

**Proposition 6.2.** For a P-event $T(A,\lambda)$ from the definition 6.1 we can construct recursively a formula $\varphi$ such that $\models \varphi$ iff $T(A,\lambda) = \lambda$.

**Proof.** If for a P-event $T(A,\lambda) S = s_1, \ldots, s_n, \pi = (\pi_1, \ldots, \pi_n), \Sigma = \sigma_1, \ldots, \sigma_k, A(\sigma_i) = (\tau^*_i)$, we take the propositional variables $s_1, \ldots, s_n$ for the states, $Q$ for the program termination and $P_1, \ldots, P_k$ for the letters of alphabet, a program variable $\alpha$ and write the following expressions of PPDL:

$$
\xi = \begin{cases}
\text{when} & \ldots \\
P_i \land S_i := \text{false} ; (\tau^*_i S_i := \text{true} \lor \ldots \lor r_{1n} S_n := \text{true} ) ; \\
& \ldots \\
\text{otherwise false?} & \text{(here } 1 \leq l \leq k, 1 \leq i \leq n). 
\end{cases}
$$

This $\xi$ defines one execution step of a probabilistic automaton $A$. It changes states (the values of $S_i$) depending on the chosen letter of the alphabet ($P_i$). The operator *when* does not appear in our language, but it may be easily expressed by the *if else*.

$$
\beta = (S_1 := \text{false} \ldots ; S_n := \text{false} ; (\pi_1 S_1 := \text{true} \lor \ldots \lor \pi_n S_n := \text{true} ) ; \text{while } Q(\alpha; \xi))
$$

This $\beta$ defines the execution of a probabilistic automaton $A$, in the beginning the initial state was chosen with right distribution, after that the program $\alpha$ works to some letter from the alphabet ($P_i$) and (possibly) the end of the word ($-Q$), and the changing of states is done by the program $\xi$.

$$
\psi = -<\alpha^*> 0 V_{1 \leq i < k} P_i \land P_{i_2} - \text{see the Proposition 3.2 for the definition of } \alpha^*.
$$

This $\psi$ says that $\alpha$ almost always does not choose two letters in
PPDL

the same time.

And now we can already define

\[ \varphi = \psi \equiv \neg \beta \wedge \bigvee_{\alpha_i \in \mathcal{P}} S_i. \]

We want to prove that this \( \varphi \) defines what we want. First of all, if \( T(A, \lambda) \neq \varphi \), then for some word \( \omega \) from the alphabet \( f^A(\omega) > \lambda \). Then we construct a model of PPDL consisting of the prefixes of the word \( \omega \) (they will be the states of the model). The value of \( P_1 \) will be true if and only if the last letter of a word is \( \alpha_i \). The program variable \( \alpha \) will add one letter to a word (if it is possible) with probability 1. And the value of \( Q \) will be false only on the word \( \omega \). Then we prove that in this model \( \Lambda \models \neg \varphi \) (\( \Lambda \) is the empty word):

Otherwise, suppose that in some model \( M' \) for some state \( I \models \neg \varphi \). Then we define a function \( g \) on the words from \( \Sigma^* \) (an 'event' from [Paz]):

for a word \( \omega = \alpha_1 \ldots \alpha_m \) we define a program term

\[ a_\omega \equiv (Q?; \alpha; P_1; \ldots Q?; \alpha; P_m; \neg Q?), \]

and then we define

\[ g(\omega) = f a_\omega(I)(\mathcal{D}). \]

We prove that \( \sum_{\omega \in \Sigma^*} g(\omega) \leq 1 \), and that,

\[ \lambda < f_p(I)(\bigcup_{\alpha_i \in \mathcal{P}} U_{\alpha_i}) = \sum_{\omega \in \Sigma^*} g(\omega) \times f^A(\omega). \]

So there exists a word \( \omega \) such that \( f^A(\omega) > \lambda \).

Q.E.D.

Note: Here we used global assignments to propositional variables in order to express emptiness of P-event. If we don't want assignments, we shall have to be more sophisticated:
Proposition 6.3. For a P-event \( T(A,\lambda) \) from the definition 6.1 we can construct recursively a formula \( \varphi \) without assignments and probabilistic choice (from PPDL\(_{001}\rightarrow r\) ) such that \( \mathcal{W} \models \varphi \iff T(A,\lambda) = \emptyset \).

Proof. The construction and the proof are very similar to those from the Proposition 9.2. First of all we note that the common case of the problem of emptiness can be reduced to the case of \( m_1 = 1, m_2 = 0 \) for \( i \neq 1 \), \( \neg S_1 \in F \) (add some new state \( S_1 \) and as the first step pass from it to some other state \( S_{i+1} \) with probability \( m_i \)). Because we do not want assignments we shall use the program variable \( \alpha \) as for changing of states, as for choosing of the letters of the alphabet. We define

\[
\beta = (\text{while } Q \alpha)
\]

\[
\psi = [\alpha^*] \land \neg (P_1 \land P_2) \land \neg (S_i \land S_j) \land (\forall S_i) \land (\forall P_i) \land \\
\land (S_i \land P_i \supset \neg \alpha \land S_j)
\]

Here \( [\alpha^*] \varphi \) means 'for iterations of \( \alpha \) there will be almost always \( \varphi \). We have some difficulties here because the expression of \( ^* \) demands assignments, but we can always change an expression \( [\alpha^*] \varphi \) by a formula \( \neg <\text{while } \neg \alpha > \text{true} \).

\[
\varphi = (\psi \land S_i) \supset <\beta > \land \bigvee_{i \in F} S_i.
\]

The proof is similar to the proof of 6.2. If \( T(A,\lambda) \neq \emptyset \), we construct the same model as in 6.2, but we take as \( \alpha \) the program term \( (\alpha;\xi) \) from the 6.2. For \( J \models \neg \varphi \) we define

\[
\alpha_{\omega} = (Q?;\alpha;P_1?; \cdots ;Q?;\alpha;P_m?; \neg Q?),
\]

\[
g(\omega) = f_{\alpha_{\omega}}(I)(D).
\]

And now we prove, as in 6.2, that \( \sum_{\omega \in \mathcal{B}^*} g(\omega) \leq 1 \), and that

\[
\lambda < f_{\beta}(I)(\bigcup_{i \in F} U_i) \leq \sum_{\omega \in \mathcal{B}^*} g(\omega) \cdot f^A(\omega)
\]

So there exists a word \( \omega \) such that \( f^A(\omega) > \lambda \).
Proposition 6.4. For a P-event $T(A,\lambda)$ from the definition 6.1 we can construct recursively a formula $\varphi$ without assignments and nesting (from $PPDL_{>010}$) such that $\models_{PPDL} \varphi \iff T(A,\lambda) = \emptyset$. The construction and the proof are standard. The formulas and the program terms we shall build are similar to those used in Proposition 6.2. We define

$$
\xi \triangleq \begin{cases}
\text{when} & 
\begin{align*}
P_i & \land S_i : (r^i_1(a_{i1};S_i) \cup \cdots \cup r^i_n(a_{in};S_n)); \\
\end{align*}
\end{cases}
$$

otherwise false? , where $1 \leq i \leq k, 1 \leq j \leq n$.

After that we define

$$
\beta \triangleq (a_1 \cup \cdots \cup a_n)
$$

and

$$
\psi \triangleq \bigwedge_{1 \leq i < j \leq n} [\beta^*] - (S_i \land S_j) \land \bigwedge_{1 \leq i < j \leq k} [\beta^*] - (P_i \land P_j)
$$

$$
\varphi \triangleq (\psi \land S_1) \lor \langle \text{while \if\xi \lambda \forall \text{S}_1 \lor S_i \rangle \times_{E} \text{S}_i.
$$

The proof is similar to the proof in the Proposition 6.3.

Theorem 6.5. The set of valid formulas of $PPDL_{>}$ is not recursively enumerable. Moreover, the set of valid formulas of $PPDL_{>1}$ and the set of valid formulas of $PPDL_{>2}$ are not recursively enumerable also.

Proof. First of all we can see that the set of P-events generating empty language is not recursively enumerable: the set of P-events generating nonempty language is easily recursively enumerable (search a word $\omega$ with $f^A(\omega) > \lambda$), and the problem of emptiness is not decidable (see [Paz]). After that we prove the
Theorem directly from the Propositions 6.3 and 6.4.

**Corollary 6.6.** The validity problem of $PPDL >_{\mathbb{R}^1_{10}}$ is exactly $\Pi^3_2$.

It follows immediately from the Th. 4.13 (existence of finite rational model), Prop. 4.2 (recursivity of truth of a finite formula in a finite model) and Th. 6.5 (undecidability of validity problem).

Now we have an easy

**Corollary 6.7.** There is no recursive proof system (axioms, inference rules etc.) for $PPDL >_{\mathbb{R}}$ (for $PPDL >_{\mathbb{R}^1_{12}}$, for $PPDL >_{\mathbb{R}^2_{y1}}$).

**Corollary 6.8.** The DPPDL enriched by probabilistic choice is undecidable. Moreover, there is no recursive proof system for this logic.

**Proof.** We can interpret recursively the $PPDL >_{\mathbb{P}10}$ in this logic.

The interesting question is what happens if we take a sublogic of $PPDL >_{\mathbb{R}}$ which is less expressive than $PPDL >_{\mathbb{R}^1_{12}}$ or $PPDL >_{\mathbb{R}^2_{y1}}$. Sometimes we can prove decidability of such logic.

**Proposition 6.9.** The intersection of $PPDL >_{\mathbb{R}^1_{12}}$ with $PPDL >_{\mathbb{R}^2_{y1}}$, i.e., the logic $PPDL >_{\mathbb{R}^10}$ is decidable.

**Proof.** This logic is recursively expressible in the Feldman's Decidable Propositional Probabilistic Dynamic Logic (see [Fe]).

**Appendix**

*Undecidability of the logic of inequalities*

The decidable logic proposed in the work of D. Kozen ([Ko83]) is the logic of a single inequality $f \leq g$ where $f$ and $g$ are terms built from measurable functions which are characteristic functions of sets or measurable function transformers.
defined by program.

We shall show that a validity problem of boolean combination of inequalities (really a conjunction of finite number of inequalities) is undecidable. We shall interpret the validity problem of \( PPDL_{\text{bool}} \) in the logic of inequalities. For interpreting of the nesting it is sufficient to define in this logic \( A = \langle x \rangle \lambda \varphi \) for some new propositional variable \( A \) (\( x \) and \( \varphi \) do not contain "diamonds"). We define this by the following two inequalities:

1) \( B \times \langle x \rangle \varphi \leq \lambda B \) - it gives us \( B = 1 \circ \langle x \rangle \lambda \varphi \).
2) \( B + \langle x \rangle \varphi \leq \lambda 1 \) - gives us \( B = 0 \circ \langle x \rangle \lambda \varphi \).

If we change in these definitions \( B \) to \( \neg A \) we should have the desired equivalence between \( A \) and \( \langle x \rangle \lambda \varphi \).

REFERENCES