AN ARRAY ASSIGNMENT FOR PROPOSITIONAL
DYNAMIC LOGIC

by

J.A. Makowsky* and M.L. Tiomkin

Technical Report #234
March 1982

* The first author was supported by Swiss National Science Foundation
Grant No. 82.820.0.80.
ABSTRACT

We propose an extension of propositional dynamic logic which allows a propositional version of an array assignments. In this logic many notions like equivalence of programs, looping and finitely branching are expressible on a propositional level. In fact we show that the resulting logic is equivalent in expressive power to first order logic augmented by a device to expressive transitive closures. In other words, it is (modulo extra predicate symbols) equivalent to first order dynamic logic. Not surprisingly, therefore, the validity problem for this extension is $\Pi_1$-complete.
1. INTRODUCTION

The major issue in logic of programs is finding a language appropriate for reasoning about programs. Usually, we would like a language which is sufficiently powerful to enable one to express in a natural way the kinds of properties one would like to prove about programs, such as correctness, termination, equivalence, various kinds of nontermination, properties of converse programs and many more, yet one would want to focus on logics which are tractable or at least (theoretically) decidable. With that in mind, Fisher and Ladner [FL] introduced propositional dynamic logic (PDL), a logic inspired by model logic. It was shown to have a decision procedure complete in deterministic exponential time and many other decidable properties. Recently various expansions of PDL have been studied. Halpern and Reif [FR] got better decision procedures for deterministic and well-structured programs, and Harel, Pnueli and Stavi [HPS] showed that the border line between decidable and undecidable is very close to PDL by showing that allowing some simple context free programming language rather than regular languages gives us already a \( \Pi_1 \)-complete undecidability.

In this paper we study another extension of PDL by adding two propositional versions of 'array assignments'. Not surprisingly this system is far from decidability. On the other hand, we can express in this system almost all properties of programs with unknown structure mentioned before. In fact, in some precise sense, this system embeds first order dynamic logic.

In PDL the labelling of states may be very useful for program checking. The unique possibility to do this in PDL is the using of
propositional variables. Assignments to propositional variables appear in [MN] in order to express the '*' by 'while'. In their paper they use a 'global assignment' (simultaneously in all states) which does not extend the expressive power by much and is not suited for labelling of states. In this paper we introduce a new kind of atomic programs 'local assignments' which, in the case of first order DL, is similar to array assignment for predicates. Both program types not only map states into states, but rather models into models preserving the domain (the set of states), but changing some predicate (1-st order case) or value of propositional variable respectively. More precisely a global propositional assignment sets the value of some propositional variable (predicate) globally to true or false, whereas local assignment does so only in the current state.

Using local and global assignments we can express a lot of readily definable properties of PDL models which are not expressible in PDL such as equivalence of two programs in current state, looping of finitely branching programs, being finitely branching etc. Note that truth of these properties in a state depends only on the states accessible from it. So will be truth of formulas involving global and local assignments. Commonly, all 'programming' notions of PDL must be 'local', i.e. not depending on 'external' states.

The authors are very grateful to Professors A. Pnueli (Weizmann Institute, Rehovot) and J. Stavi (Bar-Ilan University, Ramat-Gan) for their fruitful participating in studying the problem.
2. LANGUAGE DEFINITION

The language of PDL + GLA (PDL + global and local assignment) is similar to PDL, but we extend it by converse operator $\alpha^-$ for programs (see [Str]) and by new atomic programs for local and global assignment to predicates. We use here $p, q, r$ for predicates (propositional variables) and $\alpha, \beta, \gamma, \ldots$ for program variables. Formally the set of formulas ($\phi, \psi, \ldots$) and program terms ($\alpha, \beta, \gamma, \ldots$) are defined by simultaneous induction as follows:

(i) true, false are formulas,

(ii) if $p$ — propositional variable, then $p$ is a formula,

(iii) if $\phi, \psi$ — formulae, then $\sim\phi, (\phi \& \psi), (\phi \lor \psi), (\phi \rightarrow \psi)$ are formulae,

(iv) if $\alpha$ — program variable, then $\alpha$ is a program term,

(v) if $\alpha, \beta$ — program terms, $\phi$ — formula, then $(\alpha; \beta), (\alpha \cup \beta), \alpha^*, \alpha^-, \phi?$ are program terms,

(vi) if $\alpha$ — program term, $\phi$ — formula, then $<\alpha>\phi, [\alpha]\phi$ are formulae,

(vii) if $p$ — propositional variable, then $p:=true, p:=false$ (global assignments), $p+true, p+false$ (local assignments) are program terms.
3. MODELS, SATISFIABILITY IN MODEL

For PDL+GLA we use a model definition of PDL. A model is some

\[ M = \{M, R_{a_1}, \ldots, R_{a_k}, P_1, \ldots, P_n\} \]

where \( M \neq \emptyset \) is a domain (set of states),

\[ R_{a_i} \subseteq M^2 \] is an assignment for program variable \( a_i \), and \( P_j \subseteq M \) is an

assignment for predicate symbol \( p_j \). Now we define for formula \( \phi \) of

PDL+GLA consisting of only program variables \( a_1, \ldots, a_k \) and predicate

symbols \( p_1, \ldots, p_n \), model \( M \) and state \( I \in M \) satisfiability of \( \phi \)

in \( M \) for state \( I \) \( M \models \phi \) as follows:

(i) \( \neg I \models \phi \iff I \models \neg \phi \)

(ii) \( \phi \lor \psi \models I \iff \phi \models I \text{ or } \psi \models I \)

(iii) \( \chi \land \psi \models I \iff \chi \models I \text{ and } \psi \models I \)

\[ M \models [\alpha; \beta] \phi \iff I \models [\alpha][\beta] \phi \]

\[ M \models [\alpha \cup \beta] \phi \iff I \models (\langle \alpha \rangle \phi \lor \langle \beta \rangle \phi) \]

\[ M \models [\alpha^*] \phi \iff \exists n \geq 0 \ I \models [\alpha] \ldots [\alpha] \phi \]

\[ M \models [\alpha^?] \phi \iff \forall J < I,J \in R_{a_i} \text{ and } J \models \phi \]

\[ M \models [\alpha_i] \phi \iff \forall J < I,J \in R_{a_i} \text{ if } <J,I> \in R_{a_i} \text{ then } J \models \phi \]

\[ M \models \langle \phi \rangle \iff I \models [\phi] \iff I_M \models [\phi] \]

\[ M \models \langle \phi \rangle \iff I \models [\phi] \iff I_M \models [\phi] \]

\[ M \models \langle \phi \rangle \iff I \models [\phi] \iff I_M \models [\phi] \]

\[ M \models \langle \phi \rangle \iff I \models [\phi] \iff I_M \models [\phi] \]

\[ M \models \langle \phi \rangle \iff I \models [\phi] \iff I_M \models [\phi] \]

\[ M \models \langle \phi \rangle \iff I \models [\phi] \iff I_M \models [\phi] \]

where \( M(P | S) \) is the model \( M \) with a set \( S \) instead of \( P \).
For converse of the programs which are not program variables, we change all \((a^*)^-\) by \((a^-)^*\), \((a;\beta)^-\) by \((\beta^-;a^-)\), \((a\cup\beta)^-\) by \((a^-\cup\beta^-)\), \(a^-\) by \(a\), \(\phi?^-\) by \(\phi?\), \((p+\text{false})^-\) by \((\neg p;(p+\text{true})\cup\text{true}?))\), \((p+\text{true})^-\) by \((p?;(p+\text{false})\cup\text{true}?))\) and \((p:=S)\) by \(\text{true}?\).

**Note:** Here our converse is precise only for the programs without global assignments, but it doesn't matter for our purposes. We really need only a fact, that a relation of such converse is some extension of a converse of a program relation. The problem is that we cannot save a previous value of propositional variable after global assignment to it.
4. EQUVALENCE OF PROGRAMS AS RELATIONS ON STATES

First of all note that in PDL program equivalence is not definable because PDL with program equivalence is undecidable.

We define for propositional variable \( p \) a program term \( \text{plocltrue} \cdot (p := \text{false}; p := \text{true}) \), which assigns to \( p \) a logical value 'true' in a current state and 'false' in all other states.

For the program terms \( \alpha \) and \( \beta \) we want to express in a state \( I \) the following notion: \( \{ J : <I, J> \in R_\alpha \} = \{ J : <I, J> \in R_\beta \} \). We define this as follows:

\[
(\alpha \equiv \beta) \quad \text{plocltrue} (\alpha \cdot \text{plocltrue} \cdot \alpha; \beta; p < q) \land (\beta \cdot \text{plocltrue} \cdot \beta; p < q)
\]

(here \( p \) and \( q \) are different propositional variables which do not appear in \( \alpha, \beta \)).

For the proof that it expresses a notion defined above we need only the fact that \( R_\alpha \), \( R_\beta \) are auxiliary extensions of converses of \( R_\alpha \), \( R_\beta \) and don't need a precise notion of converse.
5. SOME INTERESTING NOTIONS OF PROGRAM LOGIC EXPRESSIBLE IN PDL + GLA

Here we will write in PDL + GLA some useful definitions of program properties which are not expressible in PDL.

P, Q are new propositional variables which don't appear in a (program term). I is a current state

a) Unique(a) \[ P \land Q \rightarrow \exists I,J \in E \land R_a \]

A result state of a computation from I is uniquely defined. It is not expressible in PDL because by adding to a domain new state J' with the same relations as on J we make a program a not unique in a state I. In spite of this we preserve a satisfiability of every PDL formula in state J.

b) FB(a) \[ P \land Q \rightarrow (a \land Q \land P) \rightarrow \exists I,J \in E \land R_a \]

Finite - a is finitely branching. It is not expressible in PDL because \( \neg \text{FB}(a) \) holds only in infinite models, whereas for PDL we have always finite model.

c) Circle(a) \[ P \land Q \rightarrow \exists I,J \in E \land R_a \]

There is a circlic computation of a iterations from a current state. Here we have also a satisfiability of \( ([a^*] \land true) \land \neg \text{Circle}(a) \) only in infinite models.

d) Inf(a) \[ \neg \text{FB}(a) \]

A is infinitely branching in a current state (see b) above).

In all these definitions a may be also auxiliary extension of R\( \alpha \) converse. We can take any such a not consisting of P, Q for a-d).
6. DEFINABILITY OF LOOP(α)

Here we wish to define looping of a program consisting of only finitely branching not looping atomic programs (usual case of a program consisting of assignments and tests).

We cannot define 'looping' of a program by expression with program variable in dynamic logic, because looping of α depends on syntax of a program term, and not only on Rα (see [MW]), so we define a formula loop(α) by external induction on a term α.

For a program variable α or for an assignment we have not looping by our initial assumptions. The unique nontrivial case of induction is loop(α*). We define it as:

\[ \text{loop}(α^*) (\langle α^* \rangle \text{loop}(α)) \lor \text{Circle}(α) \lor \text{Inf}(α^*). \]

For Circle and Inf we choose new propositional variables which do not appear in α. For a proof see [MW] (1-st order dynamic logic) - if \( \langle α^* \rangle \text{loop}(α) \), then \( \text{loop}(α^*) \), if \([α^*] \sim \text{loop}(α)\), then a tree of α computation is finitely branching, and then

\[ \text{loop}(α^*) \leftrightarrow \text{Circle}(α) \lor \text{Inf}(α^*). \]

In all definitions above we used a very strange for program logic notion of a program converse α*. Now we will show that this notion is definable, and all our definitions may be done without converse (but with the aid of additional program variables).
7. LOCAL DEFINABILITY OF α− (for α without assignments; we really need converse only on program variables, see Ch. 6)

If we have the programs α1, ..., αn and want to express "β ≡ α− in all states accessible from a current state" we can write it as follows:

\[ \text{Conv}(\alpha, \beta, \alpha_1, \ldots, \alpha_n) \cdot \left[ (\alpha \cup \beta \cup \alpha_1 \cup \ldots \cup \alpha_n)^* ; \text{Plocttrue}] ([\alpha] < [\beta] \cdot P) \land \right. \]
\[ \left. ([\beta] < [\alpha] \cdot P) \right) \Rightarrow " \text{In all accessible from current state by } \alpha, \beta, \alpha_1, \ldots, \alpha_n \text{ states } K, J < K, J > \in R_\alpha \text{ iff } < J, K > \in R_\beta". \]

After that we prove that if β is a variable, α does not consist of assignments and does not consist of P, then

(i) \[ \vdash \text{Conv}(\alpha, \alpha^-, \alpha_1, \ldots, \alpha_n) \]

(ii) For each formula \( \psi(\alpha, \alpha^-, \alpha_1, \ldots, \alpha_n) \), which does not consist of P

\[ \vdash \text{Conv}(\alpha, \beta, \alpha_1, \ldots, \alpha_n) \Rightarrow \left( \psi(\alpha, \beta, \alpha_1, \ldots, \alpha_n) \equiv \psi(\alpha, \alpha^-, \alpha_1, \ldots, \alpha_n) \right). \]

So, we may use this definition Conv(α, β, α1, ..., αn) for all formulas over α, α1, ..., αn in PDL+GLA (see above a representation of a converse of α by converses of its variables).
8. REPRESENTATION OF NATURAL NUMBERS AND EXPRESSION OF ARITHMETICS

RECURSIVELY - (giving $\Delta^1_1$ as a lower bound of complexity of a set of valid formulas).

We will write such a formula ($a$ is a program variable):

$\text{Nat } (a) \equiv \neg \text{Circle } (a) \land \exists [a^*] \text{ Unique } (a) \Rightarrow

\Rightarrow \{J: <I,J> \in R_{a^*}\}$, has a structure of natural numbers with $R_a$ as successor relation.

We can also create auxiliary numbers by a program:

$N(a,Q) = (\text{Ploctrue; } a^*; Q\text{loctrue; } a^*; P?) -$

where $Q$ will be a random number, and $P$ is a variable for returning to the initial state "zero".

Such $Q$ represents a number of iterations of $a$, that we need for reaching true value of $Q$ from the initial state.

It may be shown that here we can compute all recursive functions (it is enough to compute "+" and "*"), we can also express quantifications:

$\exists n A(n)$ will be translated to $<N(a,Q)> "\text{translation of } A(x)"

with $Q$ in a place of $x$".

We can also see that a statement $A$ of arithmetics is true iff in PDL+CLA

$\models \text{Nat } (a) \supset "\text{translation of } A"$. 
9. EQUIVALENCE OF PDL+ GLA TO FIRST ORDER CALCULI WITH TRANSITIVE CLOSURE OPERATOR

Here we consider 1-st order predicate calculi with an additional operator of transitive closure:

\[ \text{TC } \vec{u} \vec{z} \ (\varphi(\vec{u}, \vec{z}, \ldots)) \ (\vec{x}, \vec{y}) \Rightarrow \]

"There is a natural number \( n > 1 \), \( n \) vectors of elements \( \vec{s}^1, \ldots, \vec{s}^n \) such that \( \vec{s}^1 \downarrow \vec{r}, \vec{s}^n \downarrow \vec{r} \) and for every positive \( i < n \) holds \( \varphi(\vec{s}^i, \vec{s}^{i+1}, \ldots) \)."

After that we can prove the following:

**Proposition 1**: (J. Stavi [1981]). There is recursive embedding of 1-st order calculi with only two-place predicates and transitive closure operator \( \text{TC} \) (PC2 + TC) into our PDL+ GLA with the addition of such program "a", that \( R_\alpha = M^2 \) (there is a computation of "a" from each state to each state).

Note, that if we drop "recursive" as a requirement for the embedding then full first order dynamic logic with recursive programs can be embedded into PDL+ GLA using the framework of results of [MPa], [Ma], cf. Section 10.

For a proof we construct for a formula \( \varphi \) of PC2 + TC its translation \( \varphi' \) to PDL+ GLA+ \( \{a\} \). We choose for each two-place predicate of \( \varphi \) \( R_1 \) some program variable of PDL \( a_1 \), and for each variable \( x \) of \( \varphi \) a propositional variable \( P_x \) of PDL (before that we change variables in each \( \text{TCu}z(\varphi(\vec{u}, \vec{x}, \ldots))(\vec{x}, \vec{y}) \) in order to have \((\vec{u} \cup \vec{z}) \cap (\vec{x} \cup \vec{y}) = \emptyset \).

For a model \( M \) of PC2 + TC we define a model \( M' \) of PDL+GLA+\( \{a\} \) with the same domain, \( (R_1)_M' = (R_1)_M \), and \( (P_x)_M' = \{(x)_M\} \) - an assignment for \( x \) in \( M \).
For a variable \( x \) we choose
\[
(x)M \in (P)_M \text{ if } (P)_M = 1,
\]
and choose something for \( x \) in other case, so our \( M \) is not unique.

After that for \( \varphi \in \text{Fm (PC2+TC)} \) we define its translation
\[
\varphi' \in \text{Fm (PDL+GLA+\{a\})} \text{ by induction on } \varphi:
\]
(1) \( (R_1(x,y))' \triangleq <a;P_x,?;a_1>_y \)
(2) \( (x=y)' \triangleq <a;P_x,?>_y \)
(3) \( \neg \varphi' \triangleq \neg(\varphi'), (\varphi \lor \psi)' \triangleq (\varphi') \lor (\psi'), (\varphi \land \psi)' \triangleq (\varphi') \land (\psi') \)
(4) \( (\exists x \varphi(x,...))' \triangleq <a;P_{loctrue}>(\varphi(x,...))' - \text{ a satisfiability does not depend on initial value of } P_x \)
(5) For \( \forall x \) we only change in (iv) \( \iff \) to \([\] \).
(6) \( (TCzu(\psi(z,u,...))(x,y))' \triangleq <z:=x; (u:=?;(\psi(z,u,...))')?;z:=u')^k \)

After that we can prove by induction on formula \( \varphi \), that
(1) \( M \models \varphi \Rightarrow M' \models \varphi' \) (M model of PC2+TC);
(2) if for each \( x \) - free variable of \( \varphi \), \( (P_x)_M = 1 \), then
\[
M \models \varphi \Rightarrow M \models \varphi' \text{ (M model of PDL+GLA+\{a\}).}
\]

And so, if \( \varphi \) - sentence of PC2+TC, we have
\[
\text{PC2 + TC} \models \varphi \iff \text{PDL + GLA + \{a\} } \models \varphi'
\]
Q.E.D.
Note: We can express locally a notion of a "common program 'a'".

We define for program variables $a, a_1, \ldots, a_n$ and propositional variable $P$:

$$\text{COM}(a, a_1, \ldots, a_n) \equiv ((a_1 \cup \ldots \cup a_n) \land \text{Ploctrue};(a_1 \cup \ldots \cup a_n) \land \alpha) <(a)P$$

and prove that if in model $M$ holds $I \models \text{COM}(a, a_1, \ldots, a_n)$ then

$${M}^{I \models \text{COM}(a, a_1, \ldots, a_n)}$$

$${M}^{I \models \text{COM}(a, a_1, \ldots, a_n)}$$

is an elementary equivalent submodel of $M$ for formulas on only program variables $a, a_1, \ldots, a_n$, and in this submodel $R_\alpha$ is equal to $R_\alpha$ (see definition of common program 'a' above).

So if $\text{PDL+GLA+\{a\}} \models \psi(a, a_1, \ldots, a_n)$ then

$\text{PDL+GLA} \models \text{COM}(a, a_1, \ldots, a_n) \models \psi(a|a)$

and trivially if $\text{PDL+GLA} \models \text{COM}(a, a_1, \ldots, a_n) \models \psi(a|a)$ then

$\text{PDL+GLA+\{a\}} \models \psi(a, a_1, \ldots, a_n)$

because $\text{PDL+GLA+\{a\}} \models \text{COM}(a, a_1, \ldots, a_n)$.

Hence, we have for a sentence $\varphi$ of PC2+TC

$$\text{PC2+TC} \models \varphi \iff \text{PDL+GLA} \models \text{COM}(a, a_1, \ldots, a_n) \models \varphi(a|a)$$

- translation from PC2+TC into PDL+GLA, and really in the proposition above it is enough PDL+GLA without "common program" 'a'.

Note: We define for program variables $a, a_1, \ldots, a_n$ and propositional variable $P$:

$$\text{COM}(a, a_1, \ldots, a_n) \equiv ((a_1 \cup \ldots \cup a_n) \land \text{Ploctrue};(a_1 \cup \ldots \cup a_n) \land \alpha) <(a)P$$

and prove that if in model $M$ holds $I \models \text{COM}(a, a_1, \ldots, a_n)$ then

$${M}^{I \models \text{COM}(a, a_1, \ldots, a_n)}$$

is an elementary equivalent submodel of $M$ for formulas on only program variables $a, a_1, \ldots, a_n$, and in this submodel $R_\alpha$ is equal to $R_\alpha$ (see definition of common program 'a' above).

So if $\text{PDL+GLA+\{a\}} \models \psi(a, a_1, \ldots, a_n)$ then

$\text{PDL+GLA} \models \text{COM}(a, a_1, \ldots, a_n) \models \psi(a|a)$

and trivially if $\text{PDL+GLA} \models \text{COM}(a, a_1, \ldots, a_n) \models \psi(a|a)$ then

$\text{PDL+GLA+\{a\}} \models \psi(a, a_1, \ldots, a_n)$

because $\text{PDL+GLA+\{a\}} \models \text{COM}(a, a_1, \ldots, a_n)$.

Hence, we have for a sentence $\varphi$ of PC2+TC

$$\text{PC2+TC} \models \varphi \iff \text{PDL+GLA} \models \text{COM}(a, a_1, \ldots, a_n) \models \varphi(a|a)$$

- translation from PC2+TC into PDL+GLA, and really in the proposition above it is enough PDL+GLA without "common program" 'a'.
Proposition 2: There is recursive embedding of PDL+GLA into PC2+TC.

Proof: For each formula \( \phi \) of PDL+GLA we build its translation \( \bar{\phi} \) into PC2+TC with additional predicates \( S(x,y) \) \((y=x+1)\) and \( \text{Mem}_1(s,y), \text{Mem}_2(x,y) \)

\((y=(x)_1, y=(x)_2 - \text{pair enumeration})\) and a constant \( x_0 \) (zero).

Arithmetics and pair enumeration are finitely axiomatizable in PC2+TC:

\[ \forall x \exists ! y \text{Mem}_1(x,y) \& \text{Mem}_2(y,z) \]

\[ \text{Right}(S(x,y)) \rightleftharpoons (\forall z \exists ! x \exists ! y \text{Mem}_1(x,z) \& \text{Mem}_2(y,z)) \]

\[ \& (\forall xy \exists ! z \text{Mem}_1(x,z) \& \text{Mem}_2(y,z)) \& (\forall x \text{TCuv}(S(u,v))(x_0,x) \Rightarrow \exists ! y S(x,y)) \]

\[ \forall xy (\text{TCuv}(S(u,v))(x_0,x) \& S(x,y) \Rightarrow \sim \text{TCuv}(S(u,v))(y,x)) \]

In such a system we can also interprete all arithmetics:

\[ N(x) \rightleftharpoons \text{TCuv}(S(u,v))(x_0,x) \]

\[ \text{Plus}(x,y,z) \rightleftharpoons N(x) \& \text{TCuv}(u_1u_2,v_1v_2)(S(u_1,v_1) \& S(u_2,v_2))(x_0,x;y,z) - x+y=z \]

\[ \text{Mult}(x,y,z) \rightleftharpoons N(x) \& \text{TCuv}(u_1u_2,v_1v_2)(S(u_1,v_1) \& \text{Plus}(x,u_2,v_2))(x_0,x_0;y,z) - x\cdot y=z \]

and all recursive functions we need.

We also define arrays enumeration by pair enumeration. We define

\[ \langle x_1, \ldots, x_n \rangle \text{ as } \langle n, \langle \ldots \langle x_1, x_2, x_3 \rangle \ldots \rangle x_n \rangle. \]

Here we write \( x = \langle y, z \rangle \) instead of \( \text{Mem}_1(y,x) \& \text{Mem}_2(z,x) \) and arithmetic expressions instead of formulae with \( S \) and \( \text{TC} \).

It will be \( \langle x_1, \ldots, x_n, x_{n+1} \rangle = \langle n+1, \langle x_1, \ldots, x_n \rangle, x_{n+1} \rangle \). Then

\[ \text{Arrlength}(x, \ell) \rightleftharpoons \exists v \text{TCuv}(y,m;z,n(\exists w z = \langle y, w \rangle \& n=m+1)(v,1;x,\ell)) - x \text{ is an array and } \ell \text{ is its length}. \]

\[ \text{Array}(x) \rightleftharpoons \exists \ell x = \langle \ell, y \rangle \& \text{Arrlength}(y, \ell) - x \text{ is array}, \]

\[ \text{Length}(x, \ell) \rightleftharpoons \exists y x = \langle \ell, y \rangle \& \text{Arrlength}(y, \ell) - \ell \text{ is a length of } x, \]

\[ \text{Member}(x,y,k) \rightleftharpoons \text{Array}(x) \& \exists z \langle z, \ell \rangle \& \exists uv \text{TCuv}(m;v,1),n \]

\[ (\exists w. s = \langle t, w \rangle \& N(n) \& m=m+1)(z, \ell; u, k) \& \]

\[ \& ((k \geq \ell \& u = \langle v, y \rangle)(v(k+1) = u=y)) - y \text{ is } k\text{-th element of } x, \]

\[ \text{Technion - Computer Science Department - Technical Report CS0234 - 1982} \]
Cut(x,y,k) \quad Array(x) \& \exists z \& v: x = \langle z, v \rangle \& y = \langle k, v \rangle \&
\& TC \ s, m; t, n (\exists w s = \langle t, w \rangle \& n \geq 1 \& m = n+1) (z, \ell; v, k) -
- "y is a cutting array of k first elements of x".

Now we consider for each propositional variable $P$ of PDL+GLA
a predicate $P(x)$ of PC2 and for each program variable $a$ a predicate
$R_a(x, y)$ in addition to $S(x)$, $x_0$, Mem_1, Mem_2 and $D(x)$ (the domain of
states). In our $\emptyset$ we relativize all variables $x, y, z$ by $D(\cdot)$ and
all variables $i, k, \ell, m, n$ by $N(\cdot)$.

We define $P_i(x), \Box \phi \psi, \Box \phi, \neg \phi, \neg \neg \phi$.

If a program term $\alpha$ consists of atomic programs and tests $\alpha_1, \ldots, \alpha_m,$
it is equivalent to some recursive PR program
$\bigcup_{i=1}^{\infty} Q_{i-1}^i, \ldots, Q_{n-1}^i$ -
- union of compositions of atomic programs. We consider a recursive
function $f_\alpha$, which gives us for a number $i$ and array $\langle k_1^i, \ldots, k_n^i \rangle$.

For empty program we shall add to $\alpha_1^i, \ldots, \alpha_m^i$ $\alpha_{m+1}^i \quad$ true?

Then $\emptyset(x) \quad \exists y, n(y) = x \& \text{length}(y) = \text{length}(f_\alpha(n)) + 1 \&
\& \forall i \leq \text{length}(f_\alpha(n)) \left( \forall a \& f_\alpha^k(n, y, i) \& \phi((y) \text{length}(f_\alpha(n)) + 1(P_j \mid P_j^a(y, f_\alpha(n), \cdot)) \right)

(the changing is performed for all $P_j$ from $\alpha$),

where

$$
\begin{cases}
R_{\alpha_k}((y)_i, (y)_{i+1}) & \text{if } \alpha_k \text{ is program variable,} \\
\left( (y)_i = (y)_{i+1} \right) & \text{if } \alpha_k \text{ is assignment; and if } \alpha_k \not= \psi?
\end{cases}
$$

$P_j^a(y, n, x) : "n belongs to the image of $f_\alpha$, and for each $i$
if $i$ is a last appearance of a code of "$P_j := $" in $n$ and it is $P_j := S$, then if for every $i < k \leq \text{length}(n)$ $(n)_k$ is not $P_j$ or $(y)_k \neq x$, then $S$; for maximal $k$ s.t. $(n)_k$ is $P_j + S_k$ and $(y)_k = x$, $k > i$ we have $S_k$. If there is no appearance of $P_j :=$ in $n$, then if for every $k \leq \text{length}(n)$ $(n)_k$ is not $P_j$ or $(y)_k \neq x$, we have $P(x)$; for maximal $k$ s.t. $(n)_k$ is $P_j + S_k$ and $(y)_k = x$, we have $S_k$.

Now we prove:

(i) For each model $N = <D, P_1, \ldots, P_k, R_{a_1}, \ldots, R_{a_n}>$ of PDL+GLA we consider such a modification $\bar{N}$ of $N$ — add to $D$ a set of natural numbers. For every new element it will be $\sim P_j(n)$ and $\sim R_j(n, \cdot)$, $\sim R_j(\cdot, n)$; we define a predicate $D(x)$ $x \in D$, a standard predicate $S(x, y) - y = x + 1$ for numbers, and false for others, $x_0 = 0$ and find some pair enumeration $\text{Mem}_1, \text{Mem}_2$, for $\text{DVN}$ — here we use external choice axiom — for $m \geq x_0$, $m^2 = m$.

Thus, for a formula $\varphi$ of PDL+GLA we have

$$\bar{N} \models \text{Right}(S, x_0, \text{Mem}_1, \text{Mem}_2) \& \exists x D(x)$$ and

$$N \models \varphi \Rightarrow \bar{N} \models \forall x (D(x) \Rightarrow \bar{\varphi}(x)).$$

(ii) For a model $M = <D, P_1, \ldots, P_k, R_{a_1}, \ldots, R_{a_n}, S, \text{Mem}_1, \text{Mem}_2>$ of PC2+TC of a signature of $\bar{\varphi}$ we consider a same model for PDL+GLA with a domain $D = \{x : D(x)\}$ and then if

$$M \models \text{Right}(S, x_0, \text{Mem}_1, \text{Mem}_2) \& \exists x D(x)$$

then

$$M_{\text{PC2+TC}} \models \forall x \bar{\varphi}(x) \Rightarrow M_{\text{PDL+GLA}} \models \varphi$$

so

$$\models_{\text{PDL+GLA}} \varphi \Rightarrow \models_{\text{PC2+TC}} (\text{Right}(S, x_0, \text{Mem}_1, \text{Mem}_2) \& \exists x D(x)) \Rightarrow \forall x (D(x) \Rightarrow \bar{\varphi}(x)).$$
Thus, we have a recursive (PR) interpretation of PDL+GLA in PC2+TC, D, S, Mem₁, Mem₂, with 4 new additional predicates.

Q.E.D.

**Theorem:** PDL+GLA is equivalent in its expressive power to PC2+TC.

**Proof:** See Propositions 1 and 2.
10. RELATION TO FIRST ORDER DYNAMIC LOGIC

By a classical result in Model Theory [cf. Ma] PC2+TC is not compact, but expressible in \( L^{ck}_{\omega_1 w} \), the recursive infinitary logic, and all non-compact sublogics thereof are equivalent in the following weaker sense:

Two logics \( L_1, L_2 \) are \underline{AP-equivalent} if they are intertransmittable via additional predicates. All the first order dynamic logics in [Ma] are AP-equivalent, as is shown in [Ma].

Hence, we have as reformulation of the previous theorem:

Theorem: PDL+GLA is AP-equivalent to first order dynamic logic.
REFERENCES


